Machine Learning, Statistiques et Programmation

Version 0.1.407

Xavier Dupré

oct. 27, 2018

Table des matières

CHAPITRE 1

Introduction

J'ai commencé ce site un jour de nuit blanche, après avoir codé toute la journée et échangé avec mes collègues via des codes review. Les mails qui tombent sur mon portable, le énième commentaire sur la position des espaces dans mon code, une énorme angoisse m'étreignit alors. Les lignes de code défilaient comme les blocs de Tétris dans ma tête. Je me revois alors après une nuit blanche sur ce jeu, plus excité encore qu'après avoir du café à la fontaine. Je me suis dit qu'il fallait que j'arrête et que je remette un peu de sens dans tout ça.

Derrière le code, il y a les algorithmes et derrière encore des idées. Plein d'idées magnifiques, plein d'intuitions. Un code efficace n'a rien à voir avec sa lisibilité. C'est même souvent le contraire et il ne sera jamais lisible. Mais l'idée soujacente, elle, lorsqu'on l'a comprise, elle devient très clair. On n'a même plus besoin de l'écrire.

Il faudra probablement quelques années avant que ce site ne devienne conséquent, voire exhaustif. Il faut bien commencer quelque part. J'aurais pu écrire des pages wikipédia mais je n'aurais pas pu y mettre du code, des notebooks. J'ai commencé ce travail en latex lors de ma thèse, je me suis aperçu qu'il me restait quelques démonstrations à terminer.

Et en français... J'enseigne en français et puis l'offre est tellement riche en anglais.

Installation

Les exemples de codes sont disponibles sous la forme d'un module python et des *notebooks* (page 299) accessibles sur le site.

```
pip install mlstatpy
```

Je n'ai pas la prétention d'être exhaustif. Vous la trouverez dans la documentation des librairies de machine learning qui implémentent la plupart des algorithmes performants connus. Allez voir scikit-learn [1] pour connaître ce qui marche. La documentation rend obsolète n'importe quel livre au bout de six mois.

1. http://scikit-learn.org/stable/

CHAPITRE 2

Clustering

2.1 k-means

Dénomination française : algorithme des centres mobiles.

2.1.1 Principe

Les centres mobiles ou nuées dynamiques sont un algorithme de classification *non supervisée*. A partir d'un ensemble de points, il détermine pour un nombre de classes fixé, une répartition des points qui minimise un critère appelé *inertie* ou variance *intra-classe*.

Algorithme A1 : centre mobile, k-means

On considère un ensemble de points :

$$(X_i)_{1 \leqslant i \leqslant P} \in \left(\mathbb{R}^N\right)^P$$

A chaque point est associée une classe : $(c_i)_{1 \leqslant i \leqslant P} \in \{1, ..., C\}^P$. On définit les barycentres des classes : $(G_i)_{1 \leqslant i \leqslant C} \in \left(\mathbb{R}^N\right)^C$.

Initialisation

L'initialisation consiste à choisir pour chaque point une classe aléatoirement dans $\{1, ..., C\}$. On pose $t = 0$.

Calcul des barycentres

for k in $1..C$

$G_k^t \longleftarrow \sum_{i=1}^P X_i \mathbf{1}_{\left\{c_i^t = k\right\}} \sum_{i=1}^P \mathbf{1}_{\left\{c_i^t = k\right\}}$

Calcul de l'inertie

$$\begin{array}{rcl} I^t & \longleftarrow & \sum_{i=1}^P d^2\left(X_i, G_{c_i^t}^t\right) \\ t & \longleftarrow & t+1 \end{array}$$

if $t > 0$ et $I_t \sim I_{t-1}$

arrêt de l'algorithme

Attribution des classes

for in $1..P$

$c_i^{t+1} \longleftarrow \arg\min_k d\left(X_i, G_k^t\right)$

où $d\left(X_i, G_k^t\right)$ est la distance entre X_i et G_k^t

Retour à l'étape du calcul des barycentres jusqu'à convergence de l'inertie I^t.

Théorème T1 : convergence des k-means

Quelque soit l'initialisation choisie, la suite $(I_t)_{t \geqslant 0}$ construite par l'algorithme des *k-means* (page 3) converge.

La démonstration du théorème nécessite le lemme suivant.

Lemme L1 : inertie minimum

Soit $(X_1, \ldots, X_P) \in \left(\mathbb{R}^N\right)^P$, P points de $\mathbb{R}^N$, le minimum de la quantité $Q\left(Y \in \mathbb{R}^N\right)$:

$$Q(Y) = \sum_{i=1}^P d^2(X_i, Y) \tag{2.1}$$

est atteint pour $Y = G = \frac{1}{P} \sum_{i=1}^P X_i$ le barycentre des points $(X_1, \ldots, X_P)$.

Soit $(X_1, \ldots, X_P) \in \left(\mathbb{R}^N\right)^P$, P points de $\mathbb{R}^N$.

$$\begin{aligned} \sum_{i=1}^P \overrightarrow{GX_i} = \overrightarrow{0} \quad &\Longrightarrow \quad \sum_{i=1}^P d^2(X_i, Y) = \sum_{i=1}^P d^2(X_i, G) + P\, d^2(G, Y) \\ &\Longrightarrow \quad \arg\min_{Y \in \mathbb{R}^N} \sum_{i=1}^P d^2(X_i, Y) = \{G\} \end{aligned}$$

On peut maintenant démontrer le théorème. L'étape d'attribution des classes consiste à attribuer à chaque point le barycentre le plus proche. On définit J_t par :

$$J^{t+1} = \sum_{i=1}^{P} d^2\left(X_i, G^t_{c_i^{t+1}}\right) \tag{2.2}$$

On en déduit que :

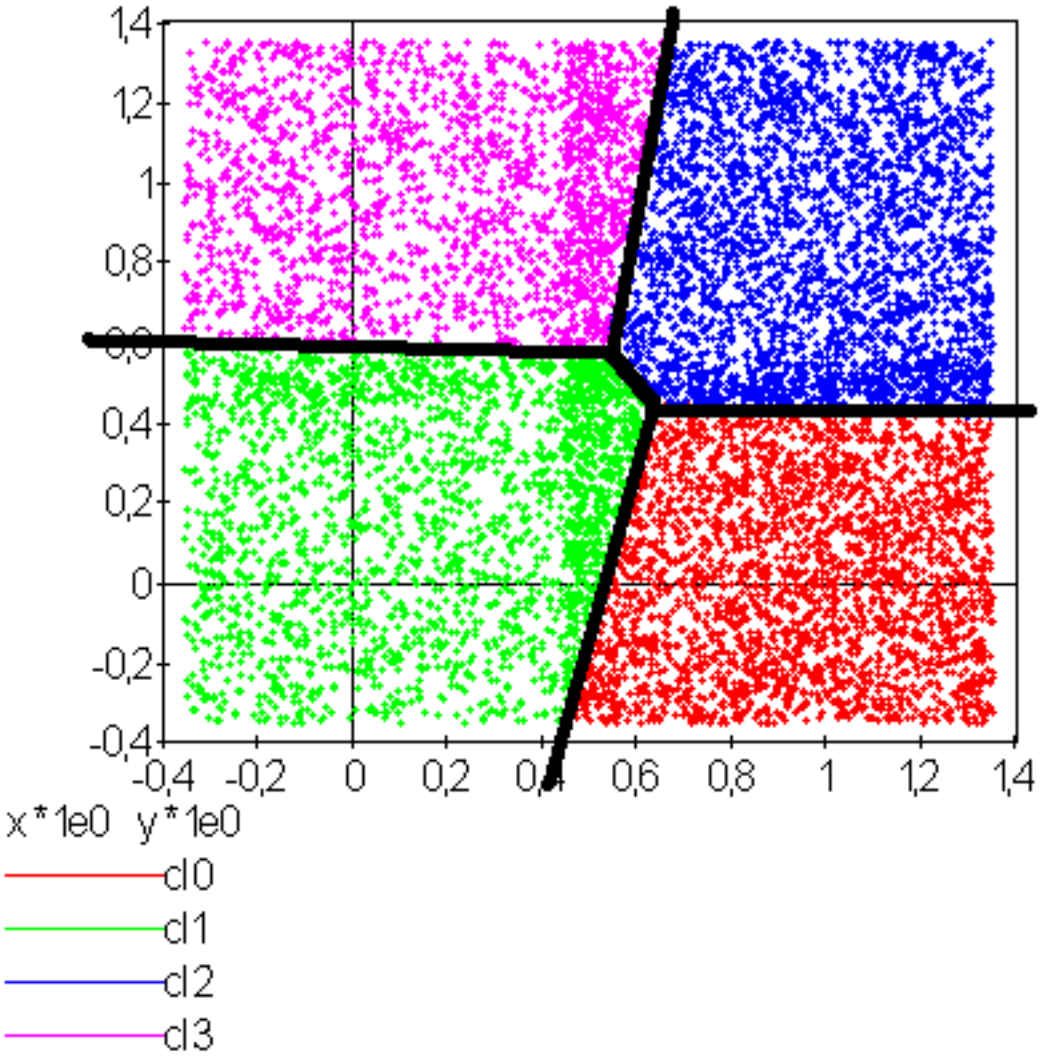
$$J^{t+1} = \sum_{i, c_i^t \neq c_i^{t+1}} d^2\left(X_i, G^t_{c_i^{t+1}}\right) + J^{t+1} \sum_{i, c_i^t = c_i^{t+1}} d^2\left(X_i, G^t_{c_i^{t+1}}\right) \tag{2.3}$$

$$J^{t+1} \leqslant \sum_{i, c_i^t \neq c_i^{t+1}} d^2\left(X_i, G^t_{c_i^t}\right) + \sum_{i, c_i^t = c_i^{t+1}} d^2\left(X_i, G^t_{c_i^t}\right) \tag{2.4}$$

$$J^{t+1} \leqslant I^t \tag{2.5}$$

Le lemme précédent appliqué à chacune des classes $\{1, \ldots, C\}$, permet d'affirmer que $I^{t+1} \leqslant J^{t+1}$. Par conséquent, la suite $(I_t)_{t \geqslant 0}$ est décroissante et minorée par 0, elle est donc convergente.

L'algorithme des centres mobiles cherche à attribuer à chaque point de l'ensemble une classe parmi les C disponibles. La solution trouvée dépend de l'initialisation et n'est pas forcément celle qui minimise l'inertie intra-classe : l'inertie finale est un minimum local. Néanmoins, elle assure que la partition est formée de classes convexes : soit c_1 et c_2 deux classes différentes, on note C_1 et C_2 les enveloppes convexes des points qui constituent ces deux classes, alors $\overset{o}{C}_1 \cap \overset{o}{C}_2 = \emptyset$. La figure suivante présente un exemple d'utilisation de l'algorithme des centres mobiles. Des points sont générés aléatoirement dans le plan et répartis en quatre groupes.



C'est une application des centres mobiles avec une classification en quatre classes d'un ensemble aléatoire de points plus dense sur la partie droite du graphe. Les quatre classes ainsi formées sont convexes.

Homogénéité des dimensions

Les coordonnées des points $(X_i) \in \mathbb{R}^N$ sont généralement non homogènes : les ordres de grandeurs de chaque dimension sont différents. C'est pourquoi il est conseillé de centrer et normaliser chaque dimension. On note : $\forall i \in \{1, \cdots, P\},\ X_i = (X_{i,1}, \ldots, X_{i,N})$:

$$\begin{aligned} g_k &= (EX)_k = \frac{1}{P}\sum_{i=1}^{P} X_{i,k} \\ v_{kk} &= \left(E\left(X - EX\right)^2\right)_{kk} = \left(EX^2\right)_{kk} - g_k^2 \end{aligned}$$

Les points centrés et normalisés sont :

$$\forall i \in \{1, \cdots, P\},\ X'_i = \left(\frac{x_{i,1} - g_1}{\sqrt{v_{11}}}, ..., \frac{x_{i,N} - g_N}{\sqrt{v_{NN}}}\right)$$

L'algorithme des centres mobiles est appliqué sur l'ensemble $(X'_i)_{1 \leqslant i \leqslant P}$. Il est possible ensuite de décorréler les variables ou d'utiliser une distance dite de Malahanobis [2] définie par $d_M(X,Y) = X\,M\,Y'$ où Y' désigne la transposée de Y et M est une matrice symétrique définie positive. Dans le cas de variables corrélées, la matrice $M = \Sigma^{-1}$ où Σ^{-1} est la matrice de variance-covariance des variables aléatoires $(X_i)_i$.

2.1.2 Améliorations de l'initialisation

K-means++

L'article *[Arthur2007]* (page **??**) montre que l'initialisation aléatoire n'est pas efficace et est sensible aux outliers ou points aberrants. L'étape d'initialisation est remplacée par la suivante :

Algorithme A2 : initialisation k-means++

Cette étape d'initialisation viendra remplacer celle définie dans l'algorithme *k-means* (page 3). On considère un ensemble de points :

$$X = (X_i)_{1 \leqslant i \leqslant P} \in \left(\mathbb{R}^N\right)^P$$

A chaque point est associée une classe : $(c_i)_{1 \leqslant i \leqslant P} \in \{1, ..., C\}^P$.

Pour k centres, on choisit C_1 au hasard dans l'ensemble X. Pour les suivants :

1. $k \leftarrow 2$
2. On choisit aléatoirement $G_k \in X$ avec la probabilité $P(x) = \frac{D_{k-1}(x)^2}{\sum_{x \in X} D_{k-1}(x)^2}$
3. $k \leftarrow k + 1$
4. On revient à l'étape 2 jusqu'à ce que $k = C$.

La fonction D_k est définie par la distance du point x au centre G_l choisi parmi les k premiers centres. $D_k(x) = \min_{1 \leqslant l \leqslant k} d(x - G_l)$.

La suite de l'algorithme *k-means++* reprend les mêmes étapes que *k-means* (page 3).

Cette initilisation éloigne le prochain centre le plus possibles des centres déjà choisis. L'article montre que :

Théorème T2 : Borne supérieure de l'erreur produite par k-means++

On définit l'inertie par $J_(X) = \sum_{i=1}^{P} \min_G d^2(X_i, G)$. Si J_{OPT} définit l'inertie optimale alors $\mathbb{E}J(X) \leqslant 8(\ln C + 2)J_{OPT}(X)$.

La démonstration est disponible dans l'article *[Arthur2007]* (page **??**).

K-means||

L'article *[Bahmani2012]* (page **??**) propose une autre initialisation que *K-means++* (page 7) mais plus rapide et parallélisable.

Algorithme A3 : initialisation k-means||

Cette étape d'initialisation viendra remplacer celle définie dans l'algorithme *k-means* (page 3). On considère un ensemble de points :

$$X = (X_i)_{1 \leqslant i \leqslant P} \in \left(\mathbb{R}^N\right)^P$$

2. https://fr.wikipedia.org/wiki/Distance_de_Mahalanobis

A chaque point est associée une classe : $(c_i)_{1 \leqslant i \leqslant P} \in \{1, ..., C\}^P$.
Pour k centres, on choisit $G = \{G_1\}$ au hasard dans l'ensemble X.

on répète $O(\ln D(G, X))$ fois :

$G' \leftarrow$ échantillon aléatoire issue de X de probabilité $p(x) = l \frac{D(G,x)^2}{\sum_x D(G,x)^2}$
$G \leftarrow G \cup G'$

La fonction $D(G, x)$ est définie par la distance du point x au plus proche centre $g \in G$: $D(g, x) = \min_{g \in G} d(x - g)$. Cette étape ajoute à l'ensemble des centres G un nombre aléatoire de centres à chaque étape. L'ensemble G contiendra plus de C centres.

1. Pour tout $g \in G$, on assigne le poids $w_g = card\{y | d(x, y) < \min_{h \in G} d(x, h)\}$
2. On clusterise l'ensemble des points G en C clusters (avec un k-means classique par exemple)

Au lieu d'ajouter les centres un par un comme dans l'algorithme *k-means++* (page 7), plusieurs sont ajoutés à chaque fois, plus l est grand, plus ce nombre est grand. Le tirage d'un échantillon aléatoire consiste à inclure chaque point x avec la probabilité $p(x) = l \frac{D(G,x)^2}{\sum_x D(G,x)^2}$.

2.1.3 Estimation de probabilités

A partir de cette classification en C classes, on construit un vecteur de probabilités pour chaque point $(X_i)_{1 \leqslant i \leqslant P}$ en supposant que la loi de X sachant sa classe c_X est une loi normale multidimensionnelle. La classe de X_i est notée c_i. On peut alors écrire :

$$\begin{aligned}
\forall i \in \{1, \cdots, C\}, & \\
G_i &= E\left(X \mathbb{1}_{\{c_X = i\}}\right) = \frac{\sum_{k=1}^P X_k \mathbb{1}_{\{c_k = i\}}}{\sum_{k=1}^P \mathbb{1}_{\{c_k = i\}}} \\
V_i &= E\left(XX' \mathbb{1}_{\{c_X = i\}}\right) = \frac{\sum_{k=1}^P X_k X_k' \mathbb{1}_{\{c_k = i\}}}{\sum_{k=1}^P \mathbb{1}_{\{c_k = i\}}} \\
\mathbb{P}(c_X = i) &= \sum_{k=1}^P \mathbb{1}_{\{c_k = i\}} \\
f(X | c_X = i) &= \frac{1}{(2\pi)^{\frac{N}{2}} \sqrt{\det(V_i)}} e^{-\frac{1}{2}(X - G_i)' V_i^{-1} (X - G_i)} \\
f(X) &= \sum_{k=1}^P f(X | c_X = i) \mathbb{P}(c_X = i)
\end{aligned}$$

On en déduit que :

$$\mathbb{P}(c_X = i | X) = \frac{f(X | c_X = i) \mathbb{P}(c_X = i)}{f(X)}$$

La densité des obervations est alors modélisée par une mélange de lois normales, chacune centrée au barycentre de chaque classe. Ces probabilités peuvent également être apprises par un réseau de neurones classifieur où servir d'initialisation à un algorithme EM [3].

3. https://fr.wikipedia.org/wiki/Algorithme_esp%C3%A9rance-maximisation

2.1.4 Sélection du nombre de classes

Critère de qualité

L'algorithme des centres mobiles effectue une classification non supervisée à condition de connaître au préalable le nombre de classes et cette information est rarement disponible. Une alternative consiste à estimer la pertinence des classifications obtenues pour différents nombres de classes, le nombre de classes optimal est celui qui correspond à la classification la plus pertinente. Cette pertinence ne peut être estimée de manière unique, elle dépend des hypothèses faites sur les éléments à classer, notamment sur la forme des classes qui peuvent être convexes ou pas, être modélisées par des lois normales multidimensionnelles, à matrice de covariances diagonales, ... Les deux critères qui suivent sont adaptés à l'algorithme des centres mobiles. Le critère de Davies-Bouldin [4] (voir *[Davies1979]* (page **??**)) est minimum lorsque le nombre de classes est optimal.

$$DB = \frac{1}{C} \sum_{i=1}^{C} \max_{i \neq j} \frac{\sigma_i + \sigma_j}{d(C_i, C_j)} \tag{2.6}$$

Avec :

C	nombre de classes
σ_i	écart-type des distances des observations de la classe i
C_i	centre de la classe i

Le critère de Goodman-Kruskal [5] (voir *[Goodman1954]* (page **??**)) est quant à lui maximum lorsque le nombre de classes est optimal. Il est toutefois plus coûteux à calculer.

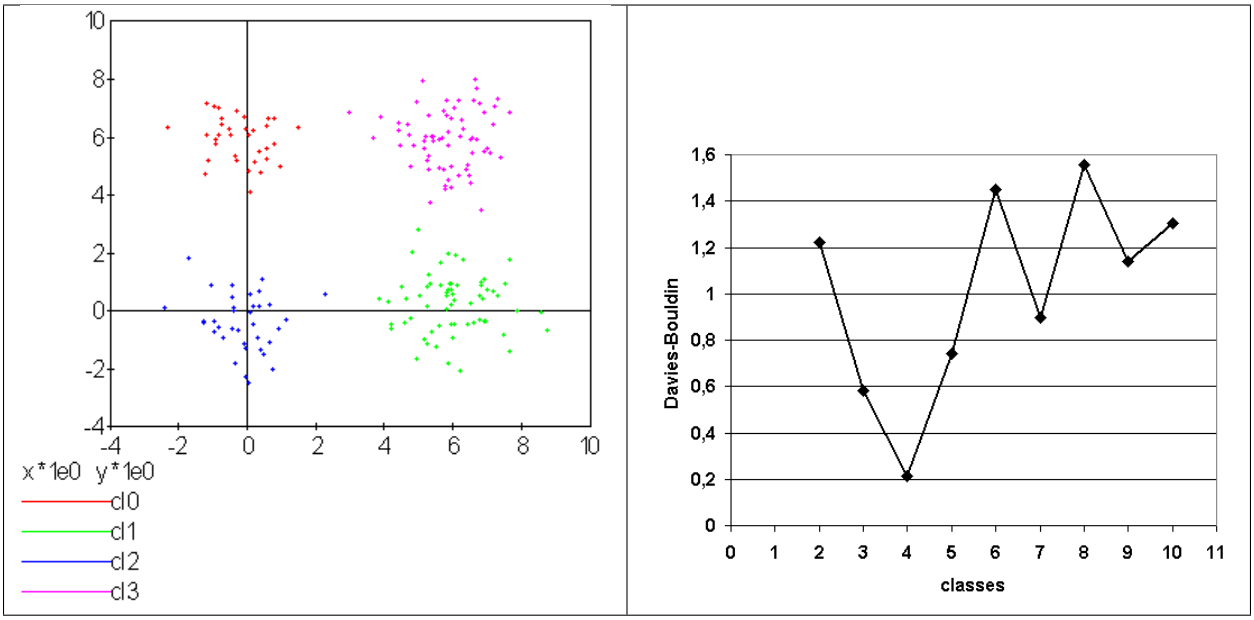
$$GK = \frac{S^+ - S^-}{S^+ + S^-} \tag{2.7}$$

Avec :

$$\begin{aligned} S^+ &= \{(q,r,s,t) \,|\, d(q,r) < d(s,t)\} \\ S^- &= \{(q,r,s,t) \,|\, d(q,r) < d(s,t)\} \end{aligned}$$

Où (q,r) sont dans la même classe et (s,t) sont dans des classes différentes.

4. https://en.wikipedia.org/wiki/Davies%E2%80%93Bouldin_index
5. https://en.wikipedia.org/wiki/Goodman_and_Kruskal%27s_gamma



Classification en quatre classes : nombre de classes sélectionnées par le critère de Davies-Bouldin dont les valeurs sont illustrées par le graphe apposé à droite.

Maxima de la fonction densité

L'article *[Herbin2001]* (page **??**) propose une méthode différente pour estimer le nombre de classes, il s'agit tout d'abord d'estimer la fonction densité du nuage de points qui est une fonction de $\mathbb{R}^n \longrightarrow \mathbb{R}$. Cette estimation est effectuée au moyen d'une méthode non paramètrique telle que les estimateurs à noyau (voir *[Silverman1986]* (page **??**)) Soit $(X_1, \ldots, X_N)$ un nuage de points inclus dans une image, on cherche à estimer la densité $f_H(x)$ au pixel x :

$$\hat{f}_H(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\det H} K\left(H^{-1}(x - X_i)\right)$$

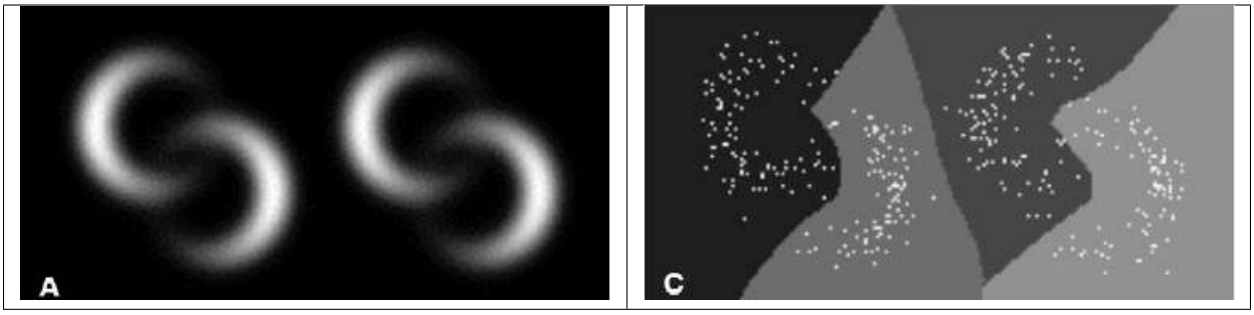
Où :

$$K(x) = \frac{1}{(2\pi)^{\frac{d}{2}}} e^{-\frac{\|x\|^2}{2}}$$

H est un paramètre estimée avec la règle de Silverman. L'exemple utilisé dans cet article est un problème de segmentation d'image qui ne peut pas être résolu par la méthode des nuées dynamiques puisque la forme des classes n'est pas convexe, ainsi que le montre la figure suivante. La fonction de densité f est seuillée de manière à obtenir une fonction $g : \mathbb{R}^n \longrightarrow \{0, 1\}$ définie par :

$$g(x) = \mathbb{1}_{\{f(x) \geqslant s\}}$$

L'ensemble $g^{-1}(\{1\}) \subset \mathbb{R}^n$ est composée de N composantes connexes notées $(C_1, \ldots, C_N)$, la classe d'un point x est alors l'indice de la composante connexe à la laquelle il appartient ou la plus proche le cas échéant.



Exemple de classification non supervisée appliquée à un problème de segmentation d'image, la première figure montre la densité obtenue, la seconde figure illustre la classification obtenue, figure extraite de *[Herbin2001]* (page **??**). Cette méthode paraît néanmoins difficilement applicable lorsque la dimension de l'espace vectoriel atteint de grande valeur. L'exemple de l'image est pratique, elle est déjà découpée en région représentées par les pixels, l'ensemble $g^{-1}(\{1\})$ correspond à l'ensemble des pixels x pour lesquels $f(x) \geqslant s$.

Décroissance du nombre de classes

L'article *[Kothari1999]* (page **??**) propose une méthode permettant de faire décroître le nombre de classes afin de choisir le nombre approprié. L'algorithme des centres mobiles proposent de faire décroître l'inertie notée I définie pour un ensemble de points noté $X = (x_1, \ldots, x_N)$ et K classes. La classe d'un élément x est notée $C(x)$. Les centres des classes sont notés $Y = (y_1, \ldots, y_K)$. L'inertie de ce nuage de points est définie par :

$$I = \sum_{x \in X} \left\| x - y_{C(x)} \right\|^2$$

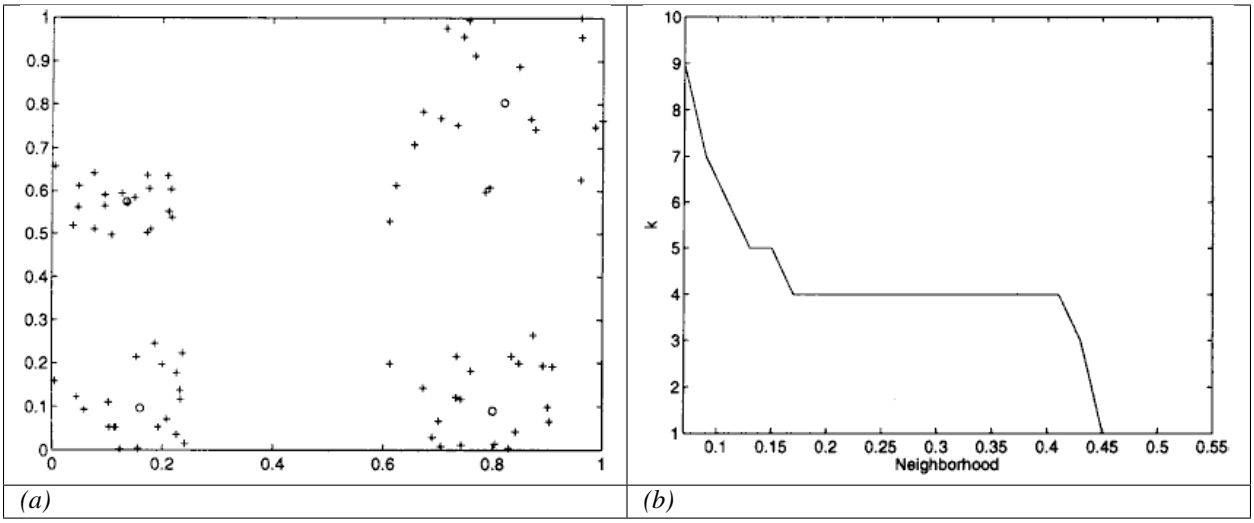
On définit tout d'abord une distance $\alpha \in \mathbb{R}^+$, puis l'ensemble $V(y, \alpha) = \{z \in Y | d(y, z) \leqslant \alpha\}$, $V(y, \alpha)$ est donc l'ensemble des voisins des centres dont la distance avec y est inférieur à α. L'article *[Kothari1999]* (page **??**) propose de minimiser le coût $J(\alpha)$ suivant :

$$J(\alpha) = \sum_{x \in X} \left\| x - y_{C(x)} \right\|^2 + \sum_{x \in X} \sum_{y \in V\left(y_{C(x)}, \alpha\right)} \lambda(y) \left\| y - y_{C(x)} \right\|^2$$

Lorsque α est nul, ce facteur est égal à l'inertie : $I = J(0)$ et ce terme est minimal lorsqu'il y a autant de classes que d'éléments dans X. Lorsque α tend vers l'infini, $J(\alpha) \rightarrow J(\infty)$ où :

$$J(\infty) = \sum_{x \in X} \left\| x - y_{C(x)} \right\|^2 + \sum_{x \in X} \sum_{y \in Y} \lambda(y) \left\| y - y_{C(x)} \right\|^2$$

Ici encore, il est possible de montrer que ce terme $J(\infty)$ est minimal lorsqu'il n'existe plus qu'une seule classe. Le principe de cette méthode consiste à faire varier le paramètre α, plus le paramètre α augmente, plus le nombre de classes devra être réduit. Néanmoins, il existe des intervalles pour lequel ce nombre de classes est stable, le véritable nombre de classes de l'ensemble X sera considéré comme celui correspondant au plus grand intervalle stable.



(a) *(b)*

Evolutation du nombre de classes en fonction du paramètre α lors de la minimisation du critère $J(\alpha)$, figure extraite de *[Kothari1999]* (page **??**). La première image représente le nuage de points illustrant quatre classes sans recouvrement. La seconde image montre que quatre classes est l'état le plus longtemps stable lorsque α croît.

Le coût $J(\alpha)$ est une somme de coût dont l'importance de l'un par rapport à l'autre est contrôle par les paramètres $\lambda(y)$. Le problème de minimisation de $J(\alpha)$ est résolu par l'algorithme qui suit. Il s'appuie sur la méthode des multiplicateurs de Lagrange.

Algorithme A4 : sélection du nombre de classes

(voir *[Kothari1999]* (page **??**)) Les notations sont celles utilisés dans les paragraphes précédents. On suppose que le paramètre α évolue dans l'intervalle $[\alpha_1, \alpha_2]$ à intervalle régulier α_t. Le nombre initial de classes est noté K et il est supposé surestimer le véritable nombre de classes. Soit $\eta \in]0,1[$, ce paramètre doit être choisi de telle sorte que dans l'algorithme qui suit, l'évolution des centres y_k soit autant assurée par le premier de la fonction de coût que par le second.

initialisation

$$\alpha \longleftarrow \alpha_1$$

On tire aléatoirement les centres des K classes $(y_1, \ldots, y_K)$.

préparation

On définit les deux suites entières $\left(c_1^1, \ldots, c_K^1\right)$, $\left(c_1^2, \ldots, c_K^2\right)$, et les deux suites de vecteur $\left(z_1^1, \ldots, z_K^1\right)$, $\left(z_1^2, \ldots, z_K^2\right)$.

$$\begin{array}{rcl} \forall k, \quad c_k^1 & = & 0 \\ \forall k, \quad c_k^2 & = & 0 \\ \forall k, \quad z_k^1 & = & 0 \\ \forall k, \quad z_k^2 & = & 0 \end{array}$$

calcul des mises à jour

for i in $1..N$

Mise à jour d'après le premier terme de la fonction de coût $J(\alpha)$.

$w \longleftarrow \underset{1 \leqslant l \leqslant K}{\arg\min} \; \|x_i - y_l\|^2$

$z_w^1 \longleftarrow z_w^1 + \eta (x_i - y_w)$

$c_w^1 \longleftarrow c_w^1 + 1$

Mise à jour d'après le second terme de la fonction de coût $J(\alpha)$

for v in $1..k$
    if $\|y_v - y_w\| < \alpha$
        $z_v^2 \longleftarrow z_v^2 - (y_v - y_w)$
        $c_v^2 \longleftarrow c_v^2 + 1$

for v in $1..k$
    $\lambda_v \longleftarrow \frac{c_v^2 \|z_v^1\|}{c_v^1 \|z_v^2\|}$
    $y_v \longleftarrow y_v + z_v^1 + \lambda_v z_v^2$

convergence

Tant que l'étape précédente n'a pas convergé vers une version stable des centres, y_k, retour à l'étape précédente. Sinon, tous les couples de classes (i, j) vérifiant $\|y_i - y_j\| > \alpha$ sont fusionnés : $\alpha \longleftarrow \alpha + \alpha_t$. Si $\alpha \leqslant \alpha 2$, retour à l'étape de préparation.

terminaison

Le nombre de classes est celui ayant prévalu pour le plus grand nombre de valeur de α.

2.1.5 Extension des nuées dynamiques

Classes elliptiques

La version de l'algorithme des nuées dynamique proposée dans l'article *[Cheung2003]* (page **??**) suppose que les classes ne sont plus de forme circulaire mais suivent une loi normale quelconque. La loi de l'échantillon constituant le nuage de points est de la forme :

$$f(x) = \sum_{i=1}^{N} p_i \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{\det \Sigma_i}} exp\left(-\frac{1}{2}(x-\mu_i)' \Sigma_i^{-1}(x-\mu_i)\right)$$

Avec $sum_{i=1}^{N} p_i = 1$. On définit :

$$G(x, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{\det \Sigma}} exp\left(-\frac{1}{2}(x-\mu)' \Sigma^{-1}(x-\mu)\right)$$

L'algorithme qui suit a pour objectif de minimiser la quantité pour un échantillon $(X_1, \ldots, X_K)$:

$$I = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{1}_{\left\{i = \underset{1 \leqslant j \leqslant N}{\arg\max} G(X_k, \mu_j, \Sigma_j)\right\}} \ln\left[p_i G(X_k, \mu_i, \Sigma_i)\right]$$

Algorithme A5 : nuées dynamiques généralisées

Les notations sont celles utilisées dans ce paragraphe. Soient η, η_s deux réels tels que $\eta > \eta_s$. La règle préconisée par l'article *[Cheung2003]* (page **??**) est $\eta_s \sim \frac{\eta}{10}$.

initialisation

$t \longleftarrow 0$. Les paramètres $\left\{p_i^0, \mu_i^0, \Sigma_i^0 | 1 \leqslant i \leqslant N\right\}$ sont initialisés grâce à un algorithme des *k-means* (page 3) ou *FSCL* (page 17). $\forall i,\ p_i^0 = \frac{1}{N}$ et $\beta_i^0 = 0$.

récurrence

Soit X_k choisi aléatoirement dans $(X_1, \ldots, X_K)$.

$$i = \underset{1 \leqslant i \leqslant N}{\arg\min}\ G\left(X_k, \mu_i^t, \Sigma_i^t\right)$$

for i in $1..N$

$$\mu_i^{t+1} = \mu_i^t + \eta\ \left(\Sigma_i^t\right)^{-1}\ \left(X_k - \mu_i^t\right)$$
$$\beta_i^{t+1} = \beta_i^t + \eta\ \left(1 - \alpha_i^t\right)$$
$$\Sigma_i^{t+1} = \left(1 - \eta_s\right)\ \Sigma_i^t + \eta_s\ \left(X_k - \mu_i^t\right)\left(X_k - \mu_i^t\right)'$$

for i in $1..N$

$$p_i^{t+1} = \frac{e^{\beta_i^{t+1}}}{\sum_{j=1}^{N} e^{\beta_j^{t+1}}}$$

$t \longleftarrow t + 1$

terminaison

Tant que $\underset{1 \leqslant i \leqslant N}{\arg\min}\ G\left(X_k, \mu_i^t, \Sigma_i^t\right)$ change pour au moins un des points X_k.

Lors de la mise à jour de Σ^{-1}, l'algorithme précédent propose la mise à jour de Σ_i alors que le calcul de $G\left(., \mu_i, \Sigma_i\right)$ implique Σ_i^{-1}, par conséquent, il est préférable de mettre à jour directement la matrice Σ^{-1} :

$$\left(\Sigma_i^{t+1}\right)^{-1} = \frac{\left(\Sigma_i^t\right)^{-1}}{1 - \eta_s}\left[I - \frac{\eta_s\left(X_k - \mu_i^t\right)\left(X_k - \mu_i^t\right)'\left(\Sigma_i^t\right)^{-1}}{1 - \eta_s + \eta_s\left(X_k - \mu_i^t\right)'\left(\Sigma_i^t\right)^{-1}\left(X_k - \mu_i^t\right)}\right]$$

Rival Penalized Competitive Learning (RPCL)

L'algorithme suivant développé dans *[Xu1993]* (page **??**), est une variante de celui des centres mobiles. Il entreprend à la fois la classification et la sélection du nombre optimal de classes à condition qu'il soit inférieur à une valeur maximale à déterminer au départ de l'algorithme. Un mécanisme permet d'éloigner les centres des classes peu pertinentes de sorte qu'aucun point ne leur sera affecté.

Algorithme A6 : RPCL

Soient $(X_1, \ldots, X_N)$, N vecteurs à classer en au plus T classes de centres $(C_1, \ldots, C_T)$. Soient deux réels α_r et α_c tels que $0 < \alpha_r \ll \alpha_c < 1$.

initialisation

Tirer aléatoirement les centres $(C_1, \ldots, C_T)$.

for j in $1..C$

$$n_j^0 \longleftarrow 1$$

calcul de poids

Choisir aléatoirement un point X_i.

for j in $1..C$

$$\gamma_j = \frac{n_j}{\sum_{k=1}^{C} n_k}$$

for j in $1..C$

$$u_j = \begin{array}{l} 1 \text{ si } j \in \arg\min\limits_{k} \left[\gamma_k \, d\left(X_i, C_k\right)\right] \\ \text{-1 si } j \in \arg\min\limits_{j \neq k} \left[\gamma_k \, d\left(X_i, C_k\right)\right] \\ 0 \text{ sinon} \end{array}$$

mise à jour

for j in $1..C$

$$C_j^{t+1} \longleftarrow C_j^t + \left\{ \begin{array}{ll} \alpha_c \left(X_i - C_j\right) & \text{si } u_j = 1 \\ -\alpha_r \left(X_i - C_j\right) & \text{si } u_j = -1 \\ 0 & \text{sinon} \end{array} \right.$$

$$n_j^t + \left\{ \begin{array}{ll} 1 & \text{si } u_j = 1 \\ 0 & \text{sinon} \end{array} \right.$$

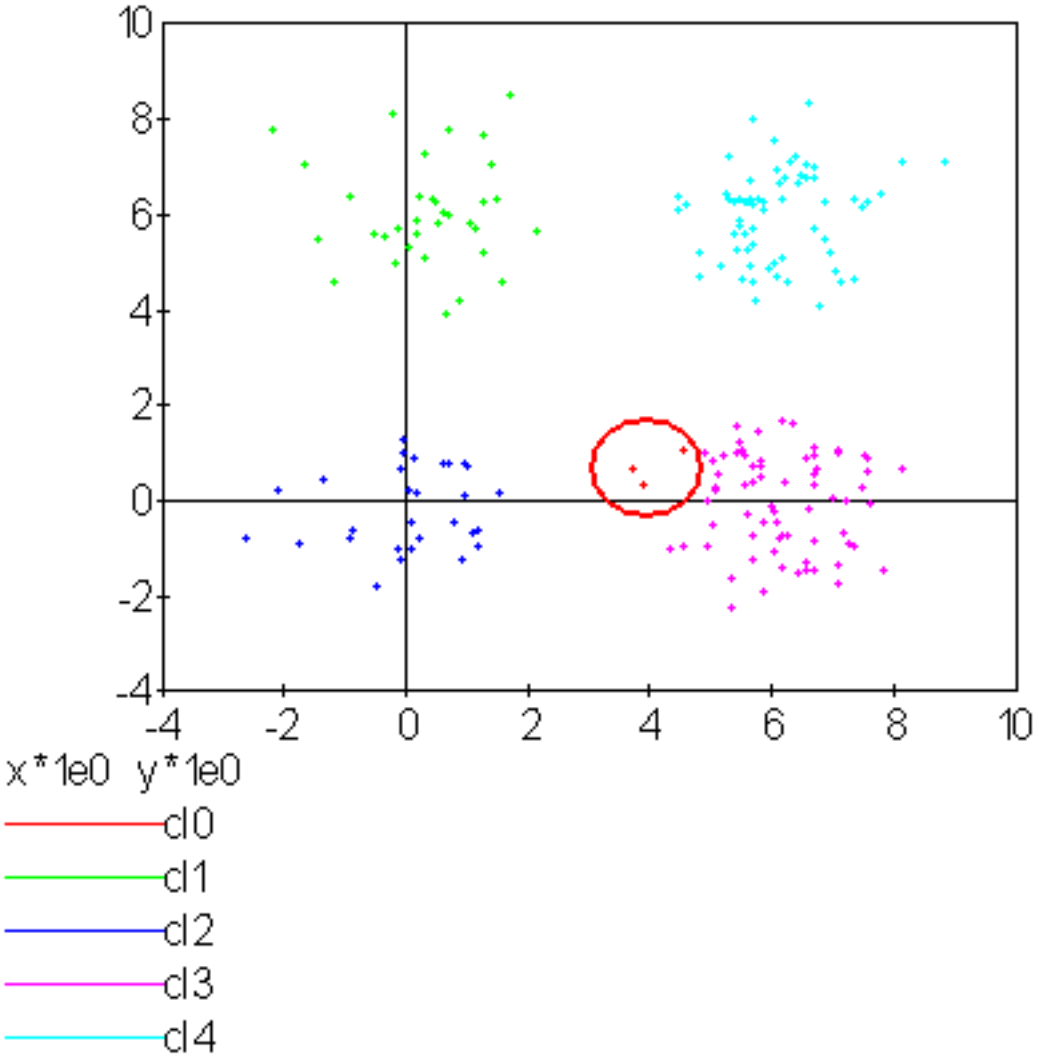
$t \longleftarrow t + 1$

terminaison

S'il existe un indice j pour lequel $C_j^{t+1} \neq C_j^t$ alors retourner à l'étape de calcul de poids ou que les centres des classes jugées inutiles ont été repoussés vers l'infini.

Pour chaque point, le centre de la classe la plus proche en est rapproché tandis que le centre de la seconde classe la plus proche en est éloigné mais d'une façon moins importante (condition $\alpha_r \ll \alpha_c$). Après convergence, les centres des classes inutiles ou non pertinentes seront repoussés vers l'infini. Par conséquent, aucun point n'y sera rattaché.

L'algorithme doit être lancé plusieurs fois. L'algorithme RPCL peut terminer sur un résultat comme celui de la figure suivante où un centre reste coincé entre plusieurs autres. Ce problème est moins important lorsque la dimension de l'espace est plus grande.



Application de l'algorithme *RPCL* (page 14) : la classe 0 est incrusté entre les quatre autres et son centre ne peut se ńăfaufileră" vers l'infini.

RPCL-based local PCA

L'article *[Liu2003]* (page **??**) propose une extension de l'algorithme *RPCL* (page 14) et suppose que les classes ne sont plus de forme circulaire mais suivent une loi normale quelconque. Cette méthode est utilisée pour la détection de ligne considérées ici comme des lois normales dégénérées en deux dimensions, la matrice de covariance définit une ellipse dont le grand axe est très supérieur au petit axe, ce que montre la figure suivante. Cette méthode est aussi présentée comme un possible algorithme de squelettisation.

Figure extraite de *[Liu2003]* (page **??**), l'algorithme est utilisé pour la détection de lignes considérées ici comme des lois normales dont la matrice de covariance définit une ellipse dégénérée dont le petit axe est très inférieur au grand axe. Les traits fin grisés correspondent aux classes isolées par l'algorithme RPCL-based local PCA.

On modélise le nuage de points par une mélange de lois normales :

$$f(x) = \sum_{i=1}^{N} p_i \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{\det \Sigma_i}} exp\left(-\frac{1}{2}(x-\mu_i)' \Sigma_i^{-1}(x-\mu_i)\right)$$

Avec $\sum_{i=1}^{N} p_i = 1$.

On suppose que le nombre de classes initiales N surestime le véritable nombre de classes. L'article *[Liu2003]* (page **??**) s'intéresse au cas particulier où les matrices de covariances vérifient $\Sigma_i = \zeta_i I + \sigma_i \phi_i \phi_i'$ avec $\zeta_i > 0,\ \sigma_i > 0,\ \phi_i' \phi_i = 1$.

On définit également :

$$G(x, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{\det \Sigma}} exp\left(-\frac{1}{2}(x-\mu)' \Sigma^{-1}(x-\mu)\right)$$

L'algorithme utilisé est similaire à l'algortihme *RPCL* (page 14). La distance d utilisée lors de l'étape de calcul des poids afin de trouver la classe la plus probable pour un point donné X_k est remplacée par l'expression :

$$d(X_k, classe\, i) = -\ln p_i^t\, G\left(X_k,\, \mu_i^t,\, \Sigma_i^t\right)$$

L'étape de mise à jour des coefficients est remplacée par :

$$x^{t+1} \longleftarrow x^t + \begin{cases} \alpha_c \nabla x^t & \text{si } u_j = 1 \\ -\alpha_r \nabla x^t & \text{si } u_j = -1 \\ 0 & \text{sinon} \end{cases}$$

Où x^t joue le rôle d'un paramètre et est remplacé successivement par p_i^t, μ_i^t, ζ_i^t, σ_i^t, ϕ_i^t :

$$\begin{array}{rcl} \nabla p_i^t & = & -\frac{1}{p_i^t} \\ \nabla \mu_i^t & = & -\left(X_k - \mu_i^t\right) \\ \nabla \zeta_i^t & = & \frac{1}{2}\, tr\left[\left(\Sigma_i^t\right)^{-1}\left(I - \left(X_k - \mu_i^t\right)\left(X_k - \mu_i^t\right)'\left(\Sigma_i^t\right)^{-1}\right)\right] \\ \nabla \sigma_i^t & = & \frac{1}{2}\, \left(\phi_i^t\right)'\left(\Sigma_i^t\right)^{-1}\left(I - \left(X_k - \mu_i^t\right)\left(X_k - \mu_i^t\right)'\left(\Sigma_i^t\right)^{-1}\right)\phi_i^t \\ \nabla \phi_i^t & = & \sigma_i^t\left(\Sigma_i^t\right)^{-1}\left(I - \left(X_k - \mu_i^t\right)\left(X_k - \mu_i^t\right)'\left(\Sigma_i^t\right)^{-1}\right)\phi_i^t \end{array}$$

Frequency Sensitive Competitive Learning (FSCL)

L'algorithme Frequency Sensitive Competitive Learning est présenté dans *[Balakrishnan1996]* (page **??**). Par rapport à l'algorithme des centres mobiles classique, lors de l'estimation des centres des classes, l'algorithme évite la formation de classes sous-représentées.

Algorithme A7 : FSCL

Soit un nuage de points $(X_1, \ldots, X_N)$, soit C vecteurs $(\omega_1, \ldots, \omega_C)$ initialisés de manière aléatoires. Soit $F : (u,t) \in \mathbb{R}^2 \longrightarrow \mathbb{R}^+$ croissante par rapport à u. Soit une suite de réels $(u_1, \ldots, u_C)$, soit une suite $\epsilon(t) \in [0,1]$ décroissante où t représente le nombre d'itérations. Au début $t \leftarrow 0$.

meilleur candidat

Pour un vecteur X_k choisi aléatoirement dans l'ensemble $(X_1, \ldots, X_N)$, on détermine :

$$i^* \in \arg\min \{D_i = F(u_i, t)\ d(X_k, \omega_i)\}$$

mise à jour

$\omega_{i^*}(t+1) \longleftarrow \omega_{i^*}(t) + \epsilon(t)(X_k - \omega_{i^*}(t))$
$t \longleftarrow t+1$
$u_{i^*} \longleftarrow u_{i^*} + 1$

Retour à l'étape précédente jusqu'à ce que les nombres $\frac{u_i}{\sum_i u_i}$ convergent.

Exemple de fonctions pour F, ϵ (voir *[Balakrishnan1996]* (page **??**)) :

$$\begin{aligned} F(u,t) &= u\,\beta e^{-t/T} \text{ avec } \beta = 0,06 \text{ et } 1/T = 0,00005 \\ \epsilon(t) &= \beta\, e^{-\gamma t} \text{ avec } \gamma = 0,05 \end{aligned}$$

Cet algorithme ressemble à celui des cartes topographiques de Kohonen sans toutefois utiliser un maillage entre les neurones (ici les vecteurs ω_i). Contrairement à l'algorithme RPCL, les neurones ne sont pas repoussés s'ils ne sont pas choisis mais la fonction croissante $F(u,t)$ par rapport à u assure que plus un neurone est sélectionné, moins il a de chance de l'être, bien que cet avantage disparaisse au fur et à mesure des itérations.

2.1.6 Bibliographie

2.2 Mélange de lois normales

2.2.1 Algorithme EM

Définition D1 : mélange de lois normales

Soit X une variable aléatoire d'un espace vectoriel de dimension d, X suit un la loi d'un mélange de N lois gaussiennes de paramètres $(\mu_i, \Sigma_i)_{1 \leqslant i \leqslant N}$, alors la densité f de X est de la forme :

$$f(x) = \sum_{i=1}^{N} p_i \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{\det \Sigma_i}}\, exp\left(-\frac{1}{2}(x-\mu_i)' \Sigma_i^{-1}(x-\mu_i)\right) \tag{2.8}$$

$$\tag{2.9}$$

Avec : $\sum_{i=1}^{N} p_i = 1$.

Dans le cas d'une loi normale à valeur réelle $\Sigma = \sigma^2$, l'algorithme permet d'estimer la loi de l'échantillon $(X_1, \ldots, X_T)$, il s'effectue en plusieurs itérations, les paramètres $p_i(0)$, $\mu_i(0)$, $\sigma^2(0)$ sont choisis de manière aléatoire, à l'itération $t+1$, la mise à jour des coefficients est faite comme suit :

$$f_{k,i}(t) = p_i(t) \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{\det \Sigma_i(t)}} \exp\left(-\frac{1}{2}(X_k - \mu_i(t))' \Sigma_i^{-1}(t)(X_k - \mu_i(t))\right) \quad (2.10)$$

$$\overline{f_{k,i}}(t) = \frac{f_{k,i}(t)}{\sum_i f_{k,i}(t)} \quad (2.11)$$

$$p_i(t+1) = \frac{1}{T} \sum_{k=1}^{T} \overline{f_{k,i}}(t) \quad (2.12)$$

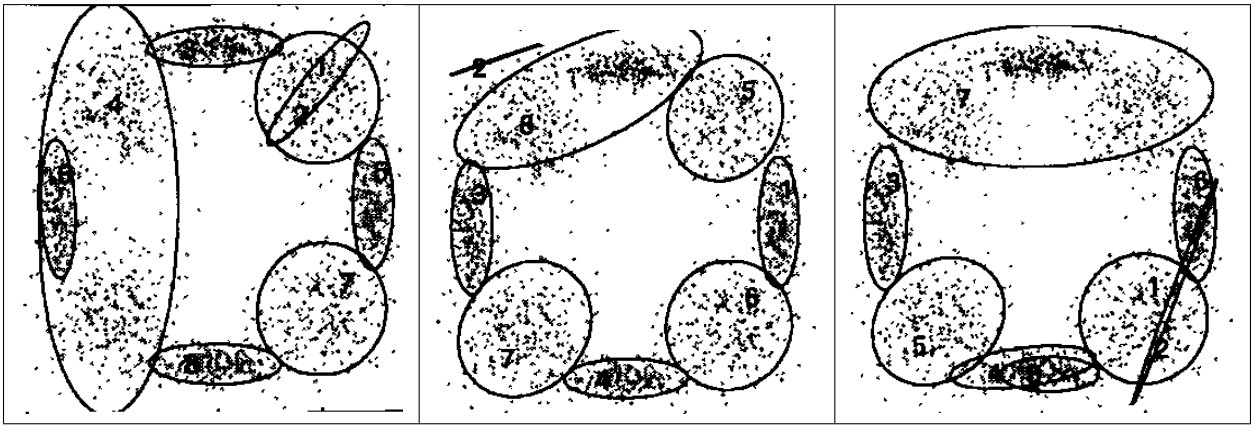
$$\mu_i(t+1) = \left[\sum_{k=1}^{T} \overline{f_{k,i}}(t)\right]^{-1} \sum_{k=1}^{T} \overline{f_{k,i}}(t) X_k \quad (2.13)$$

$$\Sigma_i^2(t+1) = \left[\sum_{k=1}^{T} \overline{f_{k,i}}(t)\right]^{-1} \sum_{k=1}^{T} \overline{f_{k,i}}(t)\ (X_k - \mu_i(t+1))(X_k - \mu_i(t+1))' \quad (2.14)$$

L'estimation d'une telle densité s'effectue par l'intermédiaire d'un algorithme de type Expectation Maximization (EM) [9] (voir *[Dempster1977]* (page **??**)) ou de ses variantes SEM [10], SAEM [11], ... (voir [Celeux1995], *[Celeux1985b]* (page **??**)). La sélection du nombre de lois dans le mélange reste un problème ouvert abordé par l'article *[Biernacki2001]* (page **??**).

2.2.2 Competitive EM algorithm

L'algorithme développé dans l'article *[ZhangB2004]* (page **??**) tente de corriger les défauts de l'algorithme EM. Cette nouvelle version appelée ńăCompetitive EMă" ou CEM s'applique à un mélange de lois - normales en particulier -, il détermine le nombre de classes optimal en supprimant ou en ajoutant des classes.



Figures extraites de *[ZhangB2004]* (page **??**), la première image montre deux classes incluant deux autres classes qui devrait donc être supprimées. La seconde image montre une classe aberrante tandis que la troisième image montre des classes se recouvrant partiellement.

9. https://fr.wikipedia.org/wiki/Algorithme_esp%C3%A9rance-maximisation
10. https://fr.wikipedia.org/wiki/Algorithme_esp%C3%A9rance-maximisation#Algorithme_SEM
11. http://wiki.webpopix.org/index.php/The_SAEM_algorithm_for_estimating_population_parameters

On considère un échantillon de variables aléatoires indépendantes et identiquement distribuées à valeur dans un espace vectoriel de dimension d. Soit X une telle variable, on suppose que X suit la loi du mélange suivant :

$$f\left(X|\theta\right) = \sum_{i=1}^{k} \alpha_i \, f\left(X|\theta_i\right)$$

Avec : $\theta = (\alpha_i, \theta_i)_{1 \leqslant i \leqslant k}$, $\forall i,\ \alpha_i \geqslant 0$ et $\sum_{i=1}^{k} \alpha_i = 1$.

On définit pour une classe m la probabilité $P_{split}(m, \theta)$ qu'elle doive être divisée et celle qu'elle doive être associée à une autre $P_{merge}(m, l, \theta)$. Celles ci sont définies comme suit :

$$P_{split}(m, \theta) = \frac{J\left(m, \theta\right)}{Z\left(\theta\right)} \tag{2.15}$$

$$P_{merge}(m, l, \theta) = \frac{\beta}{J\left(m, \theta\right) Z\left(\theta\right)} \tag{2.16}$$

β est une constante définie par expériences. $J\left(m, \theta\right)$ est défini pour l'échantillon $(x_1, \ldots, x_n)$ par :

$$J\left(m, \theta\right) = \int f_m\left(x, \theta\right) \, \log \frac{f_m\left(x, \theta\right)}{p_m\left(x, \theta_m\right)} \, dx$$

Où : $f_m\left(x, \theta\right) = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{x=x_i\}} \mathbb{P}(m|x_i, \theta)}{\sum_{i=1}^{n} \mathbb{P}(m|x_i, \theta)}$.

La constante $Z\left(\theta\right)$ est choisie de telle sorte que les probabilités $P_{split}(m, \theta)$ et $P_{merge}(m, l, \theta)$ vérifient :

$$\sum_{m=1}^{k} P_{split}(m, \theta) + \sum_{m=1}^{k} \sum_{l=m+1}^{k} P_{merge}(m, l, \theta) = 1$$

L'algorithme EM permet de construire une suite $\hat{\theta}_t$ maximisant la vraisemblance à partir de poids $\hat{\theta_0}$. L'algorithme CEM [12] est dérivé de l'algorithme EM :

Algorithme A1 : CEM

Les notations sont celles utilisées dans les paragraphes précédents. On suppose que la variable aléatoire $Z = (X, Y)$ où X est la variable observée et Y la variable cachée. T désigne le nombre maximal d'itérations.

initialisation

Choix arbitraire de k et $\hat{\theta}_0$.

Expectation

$$Q\left(\theta, \hat{\theta}_t\right) = \mathbb{E}\Big(\log\left[f\left(X, Y|\theta\right)\right] |X, \hat{\theta}_t\Big)$$

Maximization

$$\hat{\theta}_{t+1} = \arg\max_{\theta} Q\left(\theta, \hat{\theta}_t\right)$$

convergence

$t \longleftarrow t + 1$, si $\hat{\theta}_t$ n'a pas convergé vers un maximum local, alors on retourne à l'étape Expectation.

division ou regroupement

Dans le cas contraire, on estime les probabilités $P_{split}(m, \theta)$ et $P_{merge}(m, l, \theta)$ définie par les expressions eq_split_merge. On choisit aléatoirement une division ou un regroupement (les choix les plus probables ayant le plus de chance d'être sélectionnés). Ceci mène au paramètre θ'_t dont la partie modifiée par rapport à $\hat{\theta}_t$ est déterminée de manière aléatoire. L'algorithme EM est alors appliqué aux paramètres θ'_t jusqu'à convergence aux paramètres θ''_t.

12. https://fr.wikipedia.org/wiki/Algorithme_esp%C3%A9rance-maximisation#Algorithme_CEM

acceptation

On calcule le facteur suivant :

$$P_a = \min\left\{\exp\left[\frac{L\left(\theta_t'', X\right) - L\left(\theta_t, X\right)}{\gamma}\right], 1\right\}$$

On génére aléatoirement une variable $u \sim U\left[0, 1\right]$, si $u \leqslant P_a$, alors les paramètres θ_t'' sont validés. $\hat{\theta}_t \longleftarrow \theta_t''$ et retour à l'étape d'expectation. Dans le cas contraire, les paramètres θ_t'' sont refusés et retour à l'étape précédente.

terminaison

Si $t < T$, on retoure à l'étape d'expectation, Sinon, on choisit les paramètres $\theta^* = \hat{\theta}_{t^*}$ qui maximisent l'expression :

$$L\left(\theta^* | X\right) = \log f\left(X|\theta\right) - \frac{N^*}{2}\sum_{i=1}^{k^*}\log\frac{n\alpha_i^*}{12} - \frac{k^*}{2}\log\frac{n}{12} - \frac{k^*(N^*+1)}{2} \tag{2.17}$$

Avec n le nombre d'exemples et N est le nombre de paramètres spécifiant chaque composant.

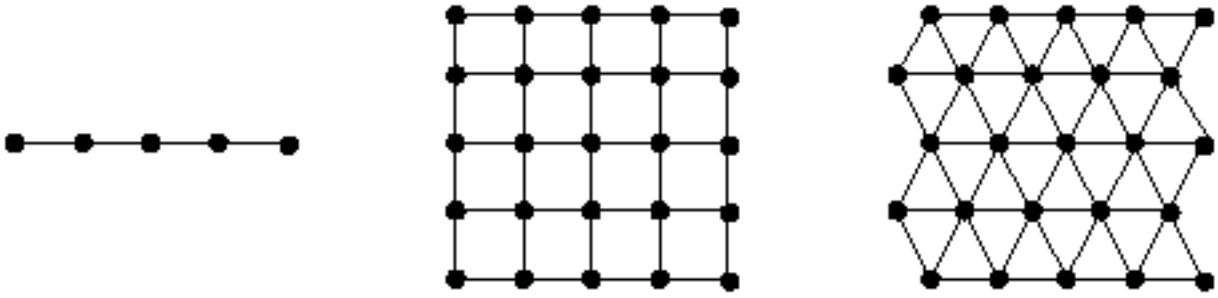
L'article *[ZhangB2004]* (page **??**) prend $\gamma = 10$ mais ne précise pas de valeur pour β qui dépend du problème. Toutefois, il existe un cas supplémentaire où la classe m doit être supprimée afin d'éviter sa convergence vers les extrêmes du nuage de points à modéliser. Si $n\alpha_m < N$, le nombre moyen de points inclus dans une classe est inférieur au nombre de paramètres attribués à cette classe qui est alors supprimée. Cette condition comme l'ensemble de l'article s'inspire de l'article *[Figueiredo2002]* (page **??**) dont est tiré le critère décrit en (ref{classif_cem_cirtere}).

2.2.3 Bibliographie

2.3 Carte de Kohonen

2.3.1 Principe

Les cartes de Kohonen ou Self Organizing Map (SOM) est le terme anglais pour les cartes de Kohonen[13]. (voir *[Kohonen1997]* (page **??**)) sont assimilées à des méthodes neuronales. Ces cartes sont constituées d'un ensemble de neurones $(\mu_1, \ldots, \mu_N)$ lesquels sont reliés par une forme récurrente de voisinage. Les neurones sont initialement répartis selon ce système de voisinage. Le réseau évolue ensuite puisque chaque point de l'espace parmi l'ensemble $(X_1, \ldots, X_K)$ attire le neurone le plus proche vers lui, ce neurone attirant à son tour ses voisins. Cette procédure est réitérée jusqu'à convergence du réseau en faisant décroître l'attirance des neurones vers les points du nuage.



13. https://fr.wikipedia.org/wiki/Carte_auto_adaptative

Trois types de voisinages couramment utilisés pour les cartes de Kohonen, voisinages linéaire, rectangulaire, triangulaire.

Algorithme A1 : cartes de Kohonen (SOM)

Soient $(\mu_1^t, \ldots, \mu_N^t) \in (\mathbb{R}^n)^N$ des neurones de l'espace vectoriel $\mathbb{R}^n$. On désigne par $V(\mu_j)$ l'ensemble des neurones voisins de μ_j pour cette carte de Kohonen. Par définition, on a $\mu_j \in V(\mu_j)$. Soit $(X_1, \ldots, X_K) \in (\mathbb{R}^n)^K$ un nuage de points. On utilise une suite de réels positifs (α_t) vérifiant $\sum_{t \geqslant 0} \alpha_t^2 < \infty$ et $\sum_{t \geqslant 0} \alpha_t = \infty$.

initialisation

Les neurones $\left(\mu_1^0, \ldots, \mu_N^0\right)$ sont répartis dans l'espace $\mathbb{R}^n$ de manière régulière selon la forme de leur voisinage. $t \longleftarrow 0$.

neurone le plus proche

On choisi aléatoirement un points du nuage X_i puis on définit le neurone $\mu_{k^*}^t$ de telle sorte que : $\|\mu_{k^*}^t - X_i\| = \min\limits_{1 \leqslant j \leqslant N} \left\|\mu_j^t - X_i\right\|$.

mise à jour

foreach μ_j^t in $V(\mu_{k^*}^t)$

$$\mu_j^{t+1} \longleftarrow \mu_j^t + \alpha_t \left(X_i - \mu_j^{t+1}\right)$$

$t \longleftarrow t+1$

Tant que l'algorithme n'a pas convergé, retour à l'étape du neurone le plus proche.

L'étape de mise à jour peut être modifiée de manière à améliorer la vitesse de convergence (voir *[Lo1991]* (page **??**)) :

$$\mu_j^{t+1} \longleftarrow \mu_j^t + \alpha_t\, h\left(\mu_j^t, \mu_{k^*}^t\right)\, \mu_k \left(X_i - \mu_j^{t+1}\right)$$

Où h est une fonction à valeur dans l'intervalle $[0,1]$ qui vaut 1 lorsque $\mu_j^t = \mu_{k^*}^t$ et qui décroît lorsque la distance entre ces deux neurones augmente. Une fonction typique est : $h(x,y) = h_0 \exp\left(-\frac{\|x-y\|^2}{2\,\sigma_t^2}\right)$.

Les cartes de Kohonen sont utilisées en analyse des données afin de projeter un nuage de points dans un espace à deux dimensions d'une manière non linéaire en utilisant un voisinage rectangulaire. Elles permettent également d'effectuer une classification non supervisée en regroupant les neurones là où les points sont concentrés. Les arêtes reliant les neurones ou sommets de la cartes de Kohonen sont soit rétrécies pour signifier que deux neurones sont voisins, soit distendues pour indiquer une séparation entre classes.

2.3.2 Carte de Kohonen et classification

L'article *[Wu2004]* (page **??**) aborde le problème d'une classification à partir du résultat obtenu depuis une *carte de Kohonen* (page 22). Plutôt que de classer les points, ce sont les neurones qui seront classés en C classes. Après avoir appliqué l¿*algorithme de Kohonen* (page 22), la méthode proposée dans *[Wu2004]* (page **??**) consiste à classer de manière non supervisée les A neurones obtenus $(\mu_1, \ldots, \mu_A)$. Toutefois, ceux-ci ne sont pas tous pris en compte afin d'éviter les points aberrants. On suppose que $\alpha_{il} = 1$ si le neurone l est le plus proche du point X_i, 0 dans le cas

contraire. Puis on construit les quantités suivantes :

$$\begin{aligned}
\nu_k &= \sum_{i=1}^{N} \alpha_{ik} \\
T_k &= \frac{1}{\nu_k} \sum_{i=1}^{N} \alpha_{ik} X_i \\
\theta(T_k) &= \sqrt{\frac{1}{\nu_k} \sum_{i=1}^{N} \alpha_{ik} \left\| X_i - T_k \right\|^2}
\end{aligned}$$

De plus :

$$\begin{aligned}
\overline{\theta} &= \frac{1}{A} \sum_{k=1}^{A} \theta(T_k) \\
\sigma(\theta) &= \sqrt{\frac{1}{A} \sum_{k=1}^{A} \left(\theta(T_k) - \overline{\theta} \right)^2}
\end{aligned}$$

Si $\nu_k = 0$ ou $\|\mu_k - T_k\| > \overline{\theta} + \sigma(\theta)$, le neurone μ_k n'est pas prise en compte lors de la classification non supervisée. Une fois celle-ci terminée, chaque élément X_i est classé selon la classe du neurone le plus proche.

L'article *[Wu2004]* (page **??**) propose également un critère permettant de déterminer le nombre de classes idéale. On note, $a_{ik} = 1$ si X_i appartient à la classe k, dans le cas contraire, $a_{ik} = 0$. On définit n_k le nombre d'éléments de la classe k, le vecteur moyenne M_k associé à la classe k :

$$\begin{aligned}
n_k &= \sum_{i=1}^{N} a_{ik} \\
M_k &= \frac{1}{n_k} \sum_{i=1}^{N} a_{ik} X_i \\
\sigma^2(M_k) &= \frac{1}{n_k} \sum_{i=1}^{N} a_{ik} \left\| X_i - M_k \right\|^2
\end{aligned}$$

On note au préalable $\sigma = \sqrt{\frac{1}{C} \sum_{k=1}^{C} \sigma^2(M_k)}$. L'article définit ensuite la densité interne pour C classes :

$$D_{int}(C) = \frac{1}{C} \sum_{k=1}^{C} \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ik} a_{jk} \mathbb{1}_{\{\|X_i - X_j\| \leqslant \sigma\}}$$

On définit la distance d^*_{kl} pour $(k,l) \in \{1,\ldots,C\}^2$, cette distance est égale à la distance minimale pour un couple de points, le premier appartenant à la classe i, le second à la classe j :

$$d^*_{kl} = \min \left\{ \|X_i - X_j\| \, | a_{ik} a_{jl} = 1 \right\} = \left\| X^{kl}_{i^*} - X^{kl}_{j^*} \right\|$$

La densité externe est alors définie en fonction du nombre de classes C par :

$$D_{ext}(C) = \sum_{k=1}^{C} \sum_{l=1}^{C} \left[\frac{d_{kl}}{\sigma(k)\,\sigma(l)} \sum_{i=1}^{N} \mathbb{1}_{\{a_{ik}+a_{il}>0\}} \mathbb{1}_{\left\{ \left\| X_i - \frac{X^{kl}_{i^*} + X^{kl}_{j^*}}{2} \right\| \leqslant \frac{\sigma(k)+\sigma(l)}{2} \right\}} \right]$$

L'article définit ensuite la séparabilité en fonction du nombre de classes C :

$$Sep(C) = \frac{1}{D_{ext}(C)} \sum_{k=1}^{C} \sum_{l=1}^{C} d^*_{kl}$$

Enfin, le critère *Composing Density Between and With clusters* noté $CDBw(C)$ est défini par :

$$CDBw(C) = D_{int}(C) * Sep(C)$$

Ce critère est maximal pour un nombre de classes optimal. Outre les résultats de l'article *[Wu2004]* (page **??**) sommairement résumés ici, ce dernier revient sur l'histoire des cartes de Kohonen, depuis leur création *[Kohonen1982]* (page **??**) jusqu'aux derniers développements récents.

2.3.3 Autres utilisation des cartes de Kohenen

On peut les utiliser pour déterminer le plus court chemin passant par tous les noeuds d'un graphe, c'est à dire appliquer Kohonen au problème du voyageur de commerce [14].

2.3.4 Bibliographie

14. http://www.xavierdupre.fr/app/ensae_teaching_cs/helpsphinx3/specials/tsp_kohonen.html

CHAPITRE 3

Bases de Machine Learning

3.1 Réseaux de neurones

Ce chapitre aborde les réseaux de neurones au travers de deux utilisations courantes, la *régression* (page 28) et la *classification* (page 32) et une qui l'est moins, l¿analyse en composantes principales [16] ou *ACP* (page 63) sans oublier les méthodes d'estimation des paramètres qui les composent, à savoir optimisations du premier et second ordre (*Méthodes du premier ordre* (page 46) et *Méthodes du second ordre* (page 46)) ainsi qu'une méthode permettant de supprimer des coefficients inutiles *Sélection de connexions* (page 59).

3.1.1 Définition des réseaux de neurones multi-couches

Les réseaux de neurones multi-couches (ou perceptrons) définissent une classe de fonctions dont l'intérêt est de pouvoir approcher n'importe quelle fonction continue à support compact (voir théorème sur la *densité* (page 35)). Aucun autre type de réseau de neurones ne sera étudié et par la suite, tout réseau de neurones sera considéré comme multi-couches (donc pas les réseau de Kohonen [17]).

Un neurone

Définition D1 : neurone

Un neurone à p entrées est une fonction $f : \mathbb{R}^{p+1} \times \mathbb{R}^p \longrightarrow \mathbb{R}$ définie par :

- $g : \mathbb{R} \longrightarrow \mathbb{R}$
- $W \in \mathbb{R}^{p+1}, W = (w_1, \ldots, w_{p+1})$
- $\forall x \in \mathbb{R}^p,\ f(W, x) = g\left(\sum_{i=1}^{p} w_i x_i + w_{p+1}\right)$ avec $x = (x_1, \ldots, x_p)$

Cette définition est inspirée du neurone biologique, les poids jouant le rôle de synapses, le vecteur x celui des *entrées* et W celui des *coefficients* ou *poids*. Le coefficient w_{p+1} est appelé le *biais* et souvent noté b. La fonction g est appelée *fonction de transfert* ou *fonction de seuil*.

16. https://fr.wikipedia.org/wiki/Analyse_en_composantes_principales
17. https://fr.wikipedia.org/wiki/Carte_auto_adaptative

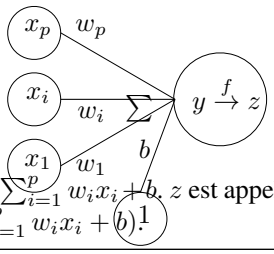


Figure F1 : neurone graphique Le vecteur $(x_1, ..., x_p) \in \mathbb{R}^p$ joue le rôle des *entrées*. y est appelé parfois le *potentiel*. $y = \sum_{i=1}^{p} w_i x_i + b$. z est appelée la sortie du neurone. f est appelée la fonction de transfert ou de seuil. $z = f(y) = f\left(\sum_{i=1}^{p} w_i x_i + b\right)$.

La réprésentation *graphique* (page 26) est plus souvent celle qu'on retient. Ce schéma est également plus proche de sa définition biologique et dissocie mieux les rôles non symétriques des entrées et des poids. Des exemples de fonctions de transfert sont donnés par la table qui suit. Les plus couramment utilisées sont les fonctions linéaire et sigmoïde.

exemples de fonction de transfert ou de seuil	expression
escalier	$1_{[0,+\infty[}$
linéaire	x
sigmoïde entre $[0,1]$	$\frac{1}{1+e^{-x}}$
sigmoïde entre $[-1,1]$	$1 - \frac{2}{1+e^{x}}$
normale	$e^{-\frac{x^2}{2}}$
exponentielle	e^{x}

La plupart des fonctions utilisées sont dérivables et cette propriété s'étend à tout assemblage de neurones, ce qui permet d'utiliser l'algorithme de rétropropagation découvert par *[Rumelhart1986]* (page **??**). Ce dernier permet le calcul de la dérivée ouvre ainsi les portes des méthodes d'optimisation basées sur cette propriété.

Une couche de neurones

Définition D2 : couche de neurones

Soit p et n deux entiers naturels, on note $W \in \mathbb{R}^{n(p+1)} = (W_1, \ldots, W_n)$ avec $\forall i \in \{1, \cdots, n\},\ W_i \in \mathbb{R}^{p+1}$. Une couche de n neurones et p entrées est une fonction :

$$F : \mathbb{R}^{n(p+1)} \times \mathbb{R}^p \longrightarrow \mathbb{R}^n$$

vérfifiant :

— $\forall i \in \{1, \cdots, n\},\ f_i$ est un neurone.
— $\forall W \in \mathbb{R}^{n(p+1)} \times \mathbb{R}^p,\ F(W, x) = (f_1(W_1, x), \ldots, f_n(W_n, x))$

Une couche de neurones représente la juxtaposition de plusieurs neurones partageant les mêmes entrées mais ayant chacun leur propre vecteur de coefficients et leur propre sortie.

Un réseau de neurones : le perceptron

Définition D3 : réseau de neurones multi-couches ou perceptron

Un réseau de neurones multi-couches à n sorties, p entrées et C couches est une liste de couches $(C_1, \ldots, C_C)$ connectées les unes aux autres de telle sorte que :

— $\forall i \in \{1, \cdots, C\}$, chaque couche C_i possède n_i neurones et p_i entrées

— $\forall i \in \{1, \cdots, C-1\},\ n_i = p_{i+1}$, de plus $p_1 = p$ et $n_C = n$

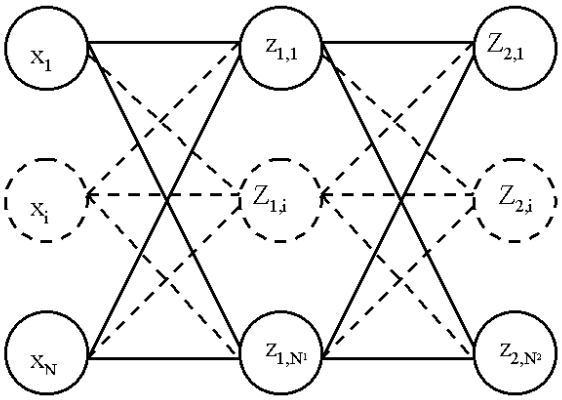
Les coefficients de la couche C_i sont notés $\left(W_1^i, \ldots, W_{n_i}^i\right)$, cette couche définit une fonction F_i. Soit la suite $(Z_i)_{0 \leqslant i \leqslant C}$ définie par :

$$Z_0 \in \mathbb{R}^p$$
$$\forall i \in \{1, \cdots, C\},\ Z_i = F_i\left(W_1^i, \ldots, W_{n_i}^i, Z_{i-1}\right)$$

On pose $M = M = \sum_{i=1}^{C} n_i\,(p_i + 1)$, le réseau de neurones ainsi défini est une fonction F telle que :

$$\begin{array}{rccc} F: & \mathbb{R}^M \times \mathbb{R}^p & \longrightarrow & \mathbb{R}^n \\ & (W, Z_0) & \longrightarrow & Z_C \end{array}$$

Figure F2 : Modèle du perceptron multi-couche (multi-layer perceptron, MLP)



— $(x_1, \ldots, x_p)$: entrées
— C_i nombre de neurones sur la couche i, $C_0 = p$
— $z_{c,i}$ sortie du neurone i, de la couche c, par extension, $z_{0,i} = x_i$
— $y_{c,i}$ potentiel du neurone i de la couche c
— $w_{c,i,j}$ coefficient associé à l'entrée j du neurone i de la couche c,
— $b_{c,i}$ biais du neurone i de la couche c
— $f_{c,i}$ fonction de seuil du neurone i de la couche c

On note W_c la matrice des poids associée à la couche c. De la même manière, B_c est le vecteur des biais associée à la couche c, Z_c, Y_c sont les objets vectoriels correspondant. On considère que les entrées forment la couche C_0 de manière à simplifier les écritures. Ainsi, chaque couche C_i du perceptron a pour entrées les sorties de la couche C_{i-1}. Cette définition est plus facile à illustrer qu'à énoncer (voir *Modèle du perceptron* (page 27)) et rappelle le rôle non symétrique des entrées et des poids. Le mécanisme qui permet de calculer les sorties d'un réseau de neurones sachant ses poids est appelé *propagation*.

Algorithme A1 : Propagation

Cet algorithme s'applique à un réseau de neurones vérifiant la définition du *perceptron* (page 26). Il s'agit de calculer les sorties de ce réseau connaissant ses poids $(w_{c,i,j})$ et ses entrées (x_j).

$Z_c \longleftarrow X$

Vient ensuite le calcul itératif de la suite $(Z_c)_{1 \leqslant c \leqslant C}$:

for c in $1..C$:

$$Y_c \longleftarrow W_c Z_{c-1} + B_c$$
$$Z_c \longleftarrow F(Y_c)$$

Le nombre de couches d'un réseau de neurones n'est pas limité. Les réseaux de deux couches (une couche pour les entrées, une couche de sortie) sont rarement utilisés. Trois couches sont nécessaires (une couche pour les entrées, une couche dite *cachée*, une couche de sortie) pour construire des modèles avec une propriété intéressante de *densité* (page 35).

3.1.2 La régression

Le bruit blanc est une variable aléatoire couramment utilisé pour désigner le hasard ou la part qui ne peut être modélisée dans une régression ou tout autre problème d'apprentissage. On suppose parfois que ce bruit suive une loi normale.

Définition D1 : bruit blanc

Une suite de variables aléatoires réelles $(\epsilon_i)_{1 \leqslant i \leqslant N}$ est un bruit blanc :
- $\exists \sigma > 0, \forall i \in \{1, \cdots, N\}, \ \epsilon_i \sim \mathcal{N}(0, \sigma)$
- $\forall (i,j) \in \{1, \cdots, N\}^2, \ i \neq j \Longrightarrow \epsilon_i \perp\!\!\!\perp \epsilon_j$

Une régression consiste à résoudre le problème suivant :

Problème P1 : Régression

Soient deux variables aléatoires X et Y, l'objectif est d'approximer la fonction $\mathbb{E}(Y|X) = f(X)$. Les données du problème sont un échantillon de points $\{(X_i, Y_i) \, | 1 \leqslant i \leqslant N\}$ et un modèle paramétré avec :math :theta‘ :

$$\forall i \in \{1, \cdots, N\}, \ Y_i = f(\theta, X_i) + \epsilon_i$$

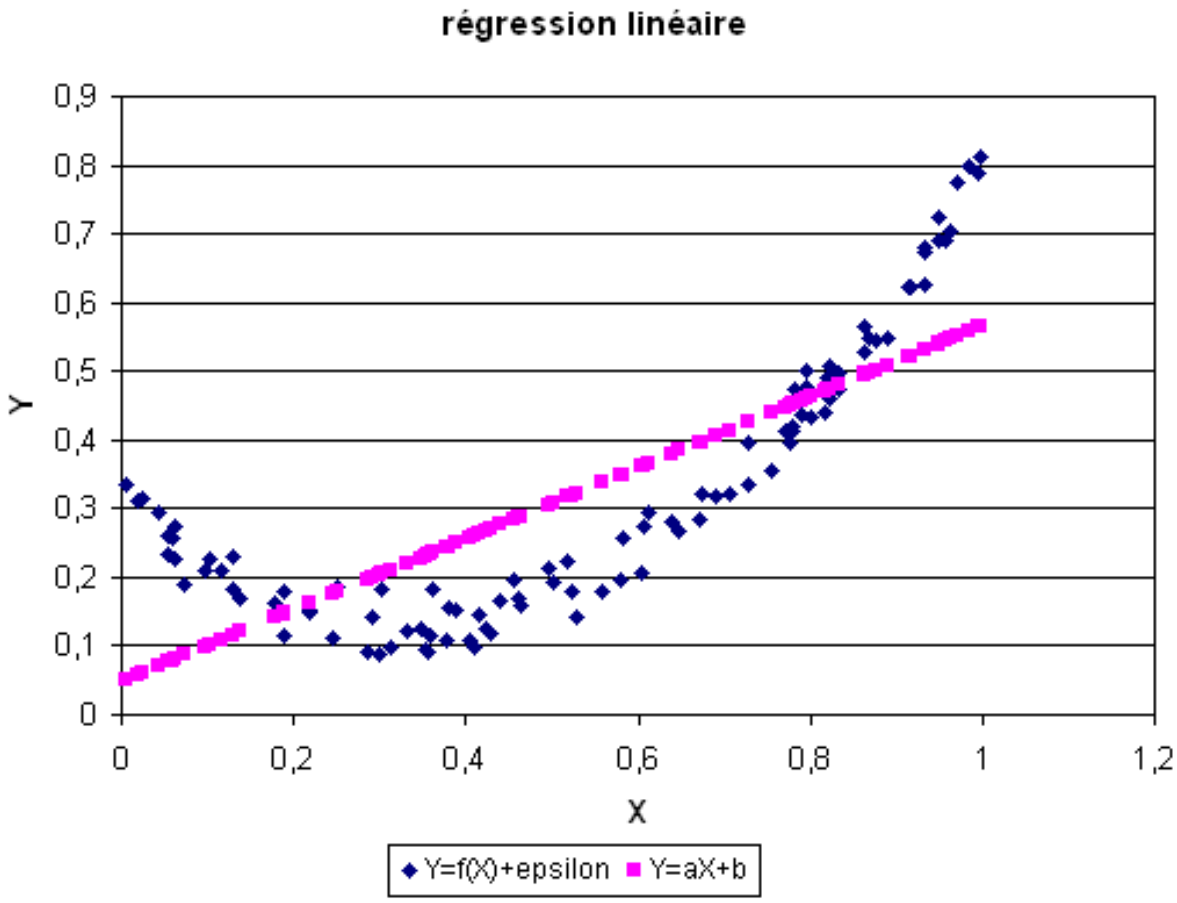
avec $n \in \mathbb{N}$, $(\epsilon_i)_{1 \leqslant i \leqslant N}$ *bruit blanc* (page 28), f est une fonction de paramètre θ.

La fonction f peut être une fonction linéaire, un polynôme, un réseau de neurones... Lorsque le bruit blanc est normal, la théorie de l'estimateur de vraisemblance (voir *[Saporta1990]* (page **??**)) permet d'affirmer que le meilleur paramètre $\hat{\theta}$ minimisant l'erreur de prédiction est :

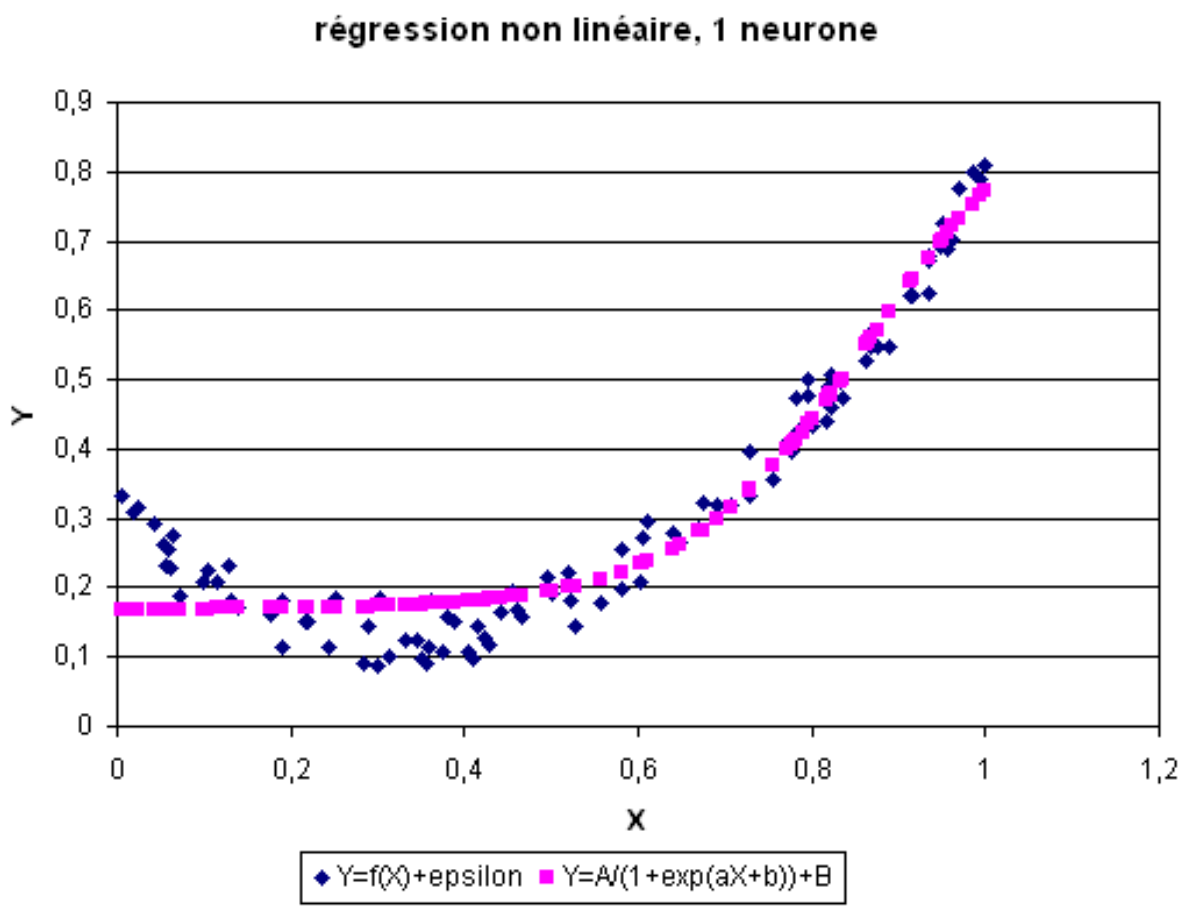
$$\hat{\theta} = \underset{\theta \in \mathbb{R}^p}{\arg\min} \, \mathbb{E}(\theta) = \underset{\theta \in \mathbb{R}^p}{\arg\min} \left[\sum_{i=1}^{N} [Y_i - f(\theta, X_i)]^2 \right]$$

Le lien entre les variables X et Y dépend des hypothèses faites sur f. Généralement, cette fonction n'est supposée non linéaire que lorsqu'une régression linéaire [18] donne de mauvais résultats. Cette hypothèse est toujours testée car la résolution du problème dans ce cas-là est déterministe et aboutit à la résolution d'un système linéaire avec autant d'équations que d'inconnues. Voici ce que ce la donne avec un nuage de points (X_i, Y_i) défini par $Y_i = \frac{3}{2} X_i^2 - X_i + \frac{1}{4} + \epsilon_i$.

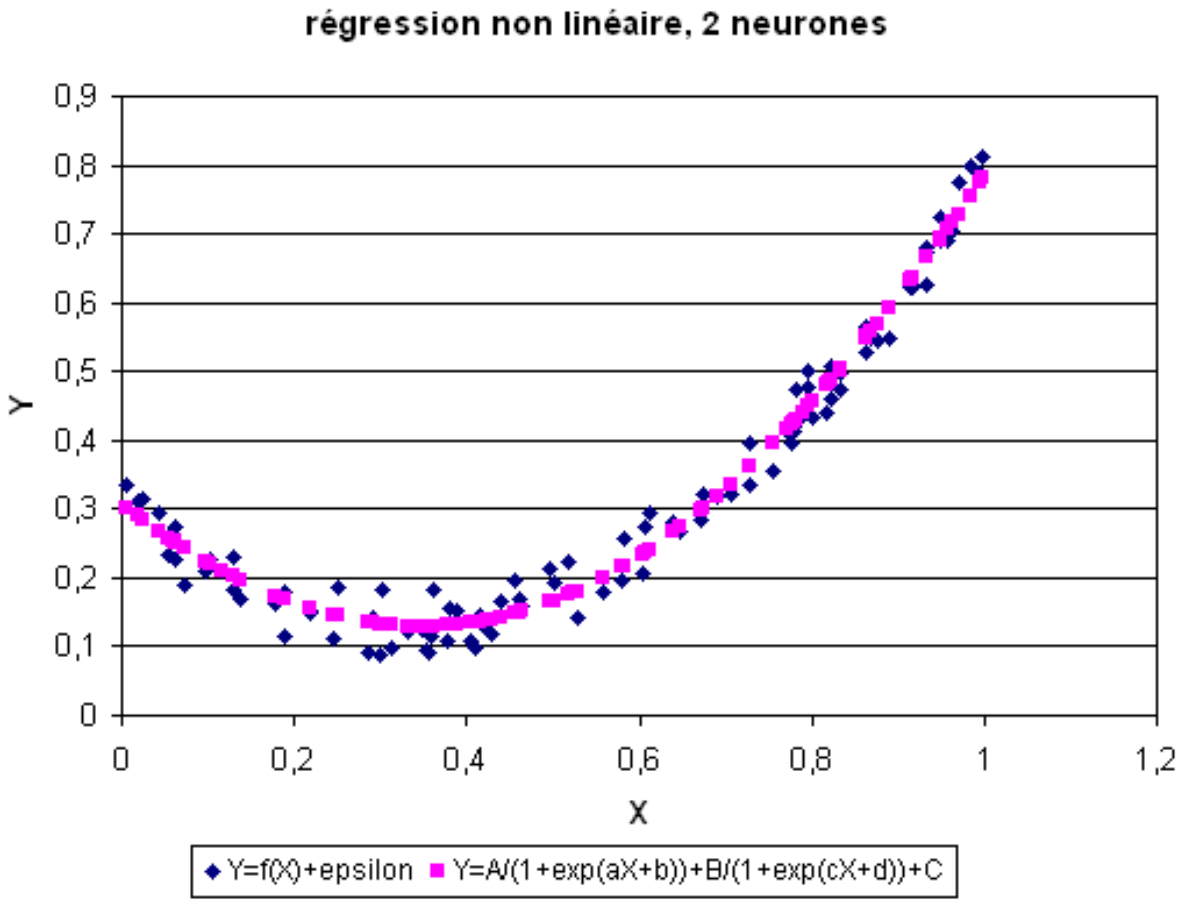
18. https://fr.wikipedia.org/wiki/R%C3%A9gression_lin%C3%A9aire



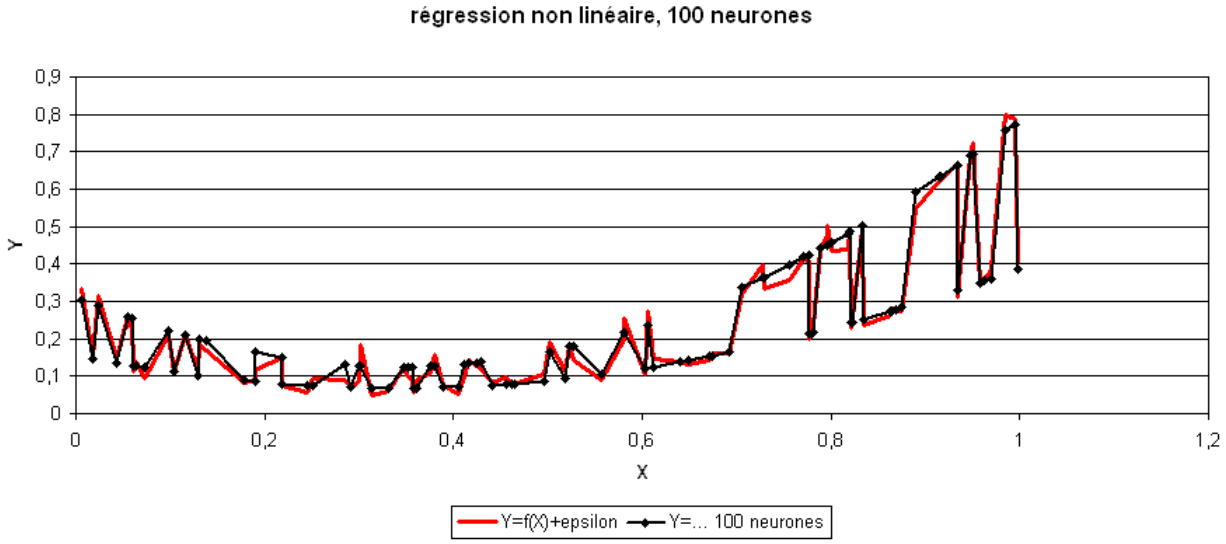
Une fonction non linéaire permet de s'approcher un peu plus de la véritable fonction. Premier cas : f est un réseau avec un neurone sur la couche cachée.



Second cas : f est un réseau avec deux neurones sur la couche cachée.



Troisième cas : f est un réseau avec 100 neurones sur la couche cachée.



L'erreur de prédiction de ce réseau de neurones est très inférieure à celle des modèles précédent, ce modèle a appris par coe ur le nuage de points (X_i, Y_i) sans vraiment ñăcomprendreă" ce qu'il apprenait. Dans le cas d'une régression à cent neurones, le nombre de coefficients du réseau de neurones (301) est largement supérieur au nombre de points (50). Il en résulte que contrairement aux trois précédents cas, la ñărichesseă" du modèle choisi lui permet d'apprendre le ñăhasardă". Lorsque ce cas de figure se présente, on dit que le réseau de neurones a appris coeur, son *pouvoir de*

généralisation est mauvais ou il fait de l¿overfitting [19] (voir aussi Generalization Error [20]). L'erreur minime estimée sur ce nuage de points (ou *base d'apprentissage*) sera considérablement accrue sur un autre nuage de points ou *base de test* suivant la même loi. Cet exemple montre que le choix du réseau de neurones le mieux adapté au problème n'est pas évident. Il existe des méthodes permettant d'approcher l'architecture optimale mais elles sont généralement coûteuses en calcul.

3.1.3 La classification

Comme la régression, la classification consiste aussi à trouver le lien entre une variable X et une variable aléatoire discrète suivant une loi multinomiale [21] Y.

Problème P1 : Classification

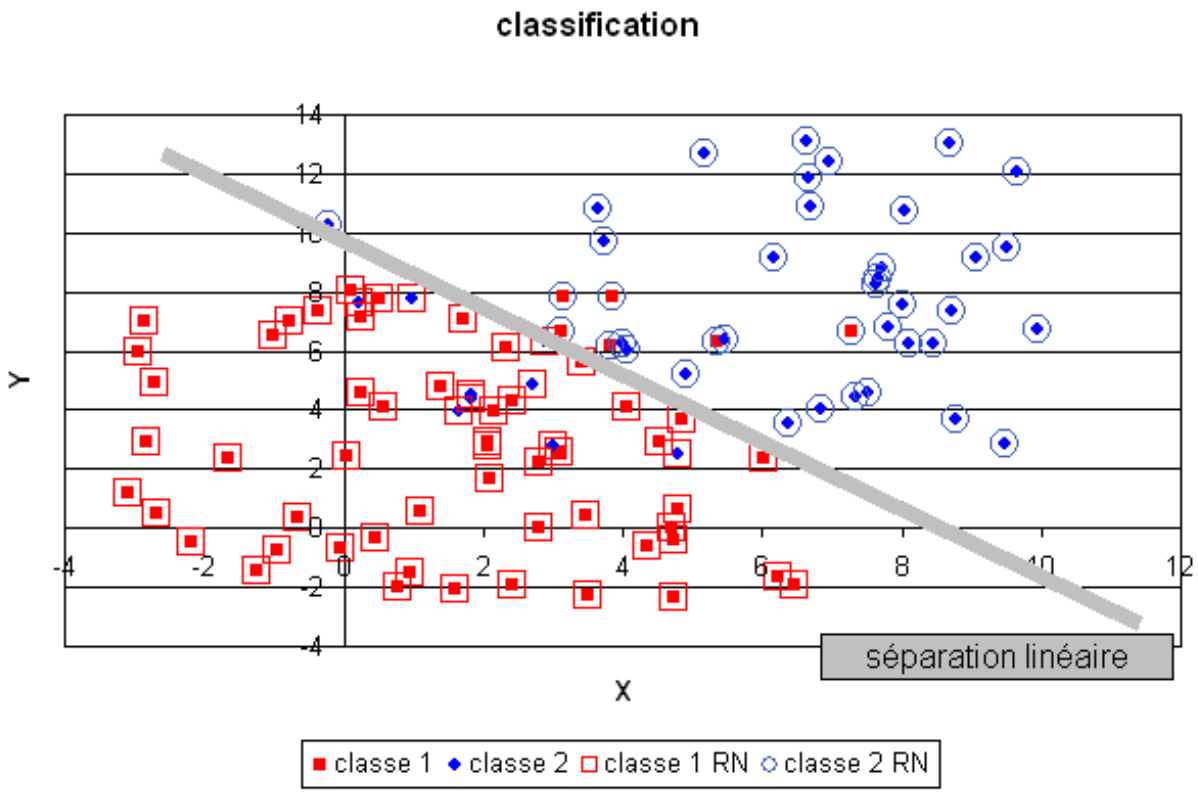
Soit une variable aléatoire X et une variable aléatoire discrète Y, l'objectif est d'approximer la fonction $\mathbb{E}(Y|X) = f(X)$. Les données du problème sont un échantillon de points : $\{(X_i, Y_i)\,|1 \leqslant i \leqslant N\}$ avec $\forall i \in \{1, \ldots, N\}$, $Y_i \in \{1, \ldots, C\}$ et un modèle paramétré avec θ :

$$\forall i \in \{1, \cdots, N\}\,,\ \forall c \in \{1, \cdots, C\}\,,\ \mathbb{P}(Y_i = c|X_i, \theta) = h(\theta, X_i, c)$$

avec $n \in \mathbb{N}$, h est une fonction de paramètre θ à valeur dans $[0, 1]$ et vérifiant la contrainte : $\sum_{c=1}^{C} h(\theta, X, c) = 1$.

Le premier exemple est une classification en deux classes, elle consiste à découvrir le lien qui unit une variable aléatoire réelle X et une variable aléatoire discrète et $Y \in \{0, 1\}$, on dispose pour cela d'une liste :

$$\{(X_i, Y_i) \in \mathbb{R} \times \{0, 1\}\,|1 \leqslant i \leqslant N\}$$

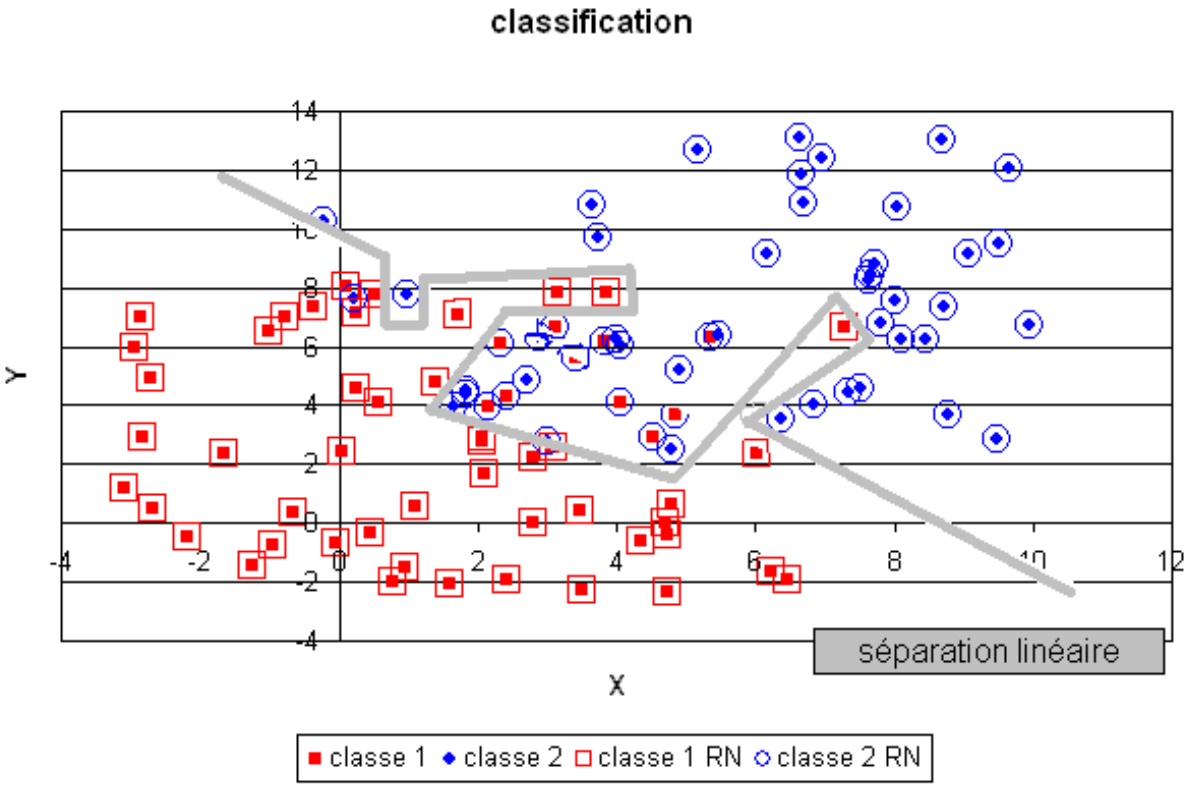


19. https://fr.wikipedia.org/wiki/Surapprentissage
20. https://en.wikipedia.org/wiki/Generalization_error
21. https://fr.wikipedia.org/wiki/Loi_multinomiale

Il n'est pas facile de déterminer directement une fonction h qui approxime $Y|X$ car h et Y sont toutes deux discrètes. C'est pourquoi, plutôt que de résoudre directement ce problème, il est préférable de déterminer la loi marginale $\mathbb{P}(Y = c|X) = f(X, \theta, c)$. f est alors une fonction dont les sorties sont continues et peut être choisie dérivable. Par exemple, f peut être un réseau de neurones dont les sorties vérifient :

$$f(X, 0) + f(X, 1) = \mathbb{P}(0|X) + \mathbb{P}(1|X) = 1$$

Le réseau de neurones utilisé pour cette tâche est légèrement différent du précédent, il sera présenté ultérieurement. Un plan a été divisé en deux demi-plan par une droite délimitant deux classes, le réseau de neurones dont la couche cachée contient deux neurones linéaires, a retrouvé cette séparation malgré les quelques exemples mal classés.



En revanche, un réseau de neurones comportant trop de coefficients aura tendance à apprendre par coeur la classification et les quelques erreurs de classification comme le montre la figure suivante. La séparation produite par le réseau de neurones est de manière évidente non linéaire puisqu'aucune droite ne peut séparer les deux classes déterminées par cette fonction. Cette classe de modèles permet donc de résoudre des problèmes complexes en gardant toutefois à l'esprit, comme dans le cas de la régression, qu'il n'est pas moins de facile de dénicher le bon modèle que dans le cas linéaire.

3.1.4 Démonstration du théorème de la densité des réseaux de neurones

Formulation du problème de la régression

Soient deux variables aléatoires continues $(X, Y) \in \mathbb{R}^p \times \mathbb{R}^q \sim \mathcal{L}$ quelconque, la résolution du problème de *régression* (page 28) est l'estimation de la fonction $\mathbb{E}(Y|X) = F(X)$. Pour cela, on dispose d'un ensemble de points $A =$

$\{(X_i, Y_i) \sim \mathcal{L} | 1 \leqslant i \leqslant N\}$.

Soit $f : \mathbb{R}^M \times \mathbb{R}^p \longrightarrow \mathbb{R}^q$ une fonction, on définit $\forall i \in \{1, \cdots, N\}$, $\widehat{Y_i^W} = f(W, X_i)$. $\widehat{Y_i^W}$ est appelée la valeur prédite pour *X_{i}*. On pose alors $\epsilon_i^W = Y_i - \widehat{Y_i^W} = Y_i - f(W, X_i)$.

Les résidus sont supposés i.i.d. (identiquement et indépendemment distribués) [22], et suivant une loi normale $\forall i \in \{1, \cdots, N\}$, $\epsilon_i^W \sim \mathcal{N}(\mu_W, \sigma_W)$ La vraisemblance d'un échantillon $(Z_i)_{1 \leqslant i \leqslant N}$, où les Z_i sont indépendantes entre elles et suivent la loi de densité $f(z|\theta)$ est la densité du vecteur $(Z_1, \ldots, Z_N)$ qu'on exprime comme suit :

$$\begin{array}{rcl} L(\theta, Z_1, \ldots, Z_N) & = & \prod_{n=1}^{N} f(Z_i|\theta) \\ \Longrightarrow \ln L(\theta, Z_1, \ldots, Z_N) & = & \sum_{n=1}^{N} \ln f(Z_i|\theta) \end{array}$$

La log-vraisemblance de l'échantillon s'écrit $L_W = -\frac{1}{2\sigma_W^2} \sum_{i=1}^{N} \left(Y_i - \widehat{Y_i^W} - \mu_W\right)^2 + N \ln\left(\sigma_W \sqrt{2\pi}\right)$. Les estimateurs du maximum de vraisemblance pour μ_W et σ_W sont (voir *[Saporta1990]* (page **??**)) :

$$\begin{array}{rcl} \widehat{\mu_W} & = & \frac{1}{N} \sum_{i=1}^{N} Y_i - \widehat{Y_i^W} \\ \widehat{\sigma_W} & = & \sqrt{\frac{\sum_{i=1}^{N} \left(Y_i - \widehat{Y_i^W} - \mu_W\right)^2}{N}} \end{array}$$

L'estimateur de $\widehat{Y} = f(W, X)$ désirée est de préférence sans biais ($\mu_W = 0$) et de variance minimum, par conséquent, les paramètres $\overset{*}{W}$ qui maximisent la vraisemblance L_W sont :

$$\begin{array}{rcl} \overset{*}{W} & = & \underset{W \in \mathbb{R}^M}{\arg\min} \sum_{i=1}^{N} \left(Y_i - \widehat{Y_i^W}\right)^2 \\ & = & \underset{W \in \mathbb{R}^M}{\arg\min} \sum_{i=1}^{N} \left(Y_i - f(W, X_i)\right)^2 \end{array} \tag{3.1}$$

Réciproquement, on vérifie que si W^* vérifie l'équation *(1)* (page **??**) alors l'estimateur défini par f est sans biais Il suffit pour s'en convaincre de poser $g = f + \alpha$ avec $\alpha \in \mathbb{R}$ et de vérifier que la valeur optimale pour α est $\alpha = -\frac{1}{N} \sum_{i=1}^{N} Y_i - f(W, X_i)$. L'estimateur minimise la vraisemblance L_W. Cette formule peut être généralisée en faisant une autre hypothèse que celle de la normalité des résidus (l'indépendance étant conservée), l'équation *(1)* (page **??**) peut généralisée par *(2)* (page **??**).

$$\begin{array}{rcl} \overset{*}{W} & = & \underset{W \in \mathbb{R}^M}{\arg\min} \sum_{i=1}^{N} e\left(Y_i - \widehat{Y_i^W}\right) \\ & = & \underset{W \in \mathbb{R}^M}{\arg\min} \sum_{i=1}^{N} e\left(Y_i - f(W, X_i)\right) \end{array} \tag{3.2}$$

Où la fonction $e : \mathbb{R}^q \in \mathbb{R}$ est appelée fonction d'erreur.

Densité des réseaux de neurones

L'utilisation de réseaux de neurones s'est considérablement développée depuis que l'algorithme de rétropropagation a été trouvé (*[LeCun1985]* (page **??**), *[Rumelhart1986]* (page **??**), *[Bishop1995]* (page **??**)). Ce dernier permet d'estimer la dérivée d'un réseau de neurones en un point donné et a ouvert la voie à des méthodes classiques de résolution pour des problèmes d'optimisation tels que la régression non linéaire.

Comme l'ensemble des fonctions polynômiales, l'ensemble des fonctions engendrées par des réseaux de neurones multi-couches possède des propriétés de *densité* (page 35) et sont infiniment dérivables. Les réseaux de neurones comme les polynômes sont utilisés pour modéliser la fonction f de l'équation *(2)* (page **??**). Ils diffèrent néanmoins sur certains points

Si une couche ne contient que des fonctions de transfert bornées comme la fonction sigmoïde, tout réseau de neurones incluant cette couche sera aussi borné. D'un point de vue informatique, il est préférable d'effectuer des calculs avec des

22. https://fr.wikipedia.org/wiki/Variables_ind%C3%A9pendantes_et_identiquement_distribu%C3%A9es

valeurs du même ordre de grandeur. Pour un polynôme, les valeurs des termes de degré élevé peuvent être largement supérieurs à leur somme.

Un autre attrait est la symétrie dans l'architecture d'un réseau de neurones, les neurones qui le composent jouent des rôles symétriques (corollaire *familles libres* (page 39). Pour améliorer l'approximation d'une fonction, dans un cas, il suffit d'ajouter un neurone au réseau, dans l'autre, il faut inclure des polynômes de degré plus élevé que ceux déjà employés.

Théorème T1 : densité des réseaux de neurones (Cybenko1989)

[Cybenko1989] (page **??**) Soit E_p^q l'espace des réseaux de neurones à p entrées et q sorties, possédant une couche cachée dont la fonction de seuil est une fonction sigmoïde $\left(x \to 1 - \frac{2}{1+e^x}\right)$, une couche de sortie dont la fonction de seuil est linéaire Soit F_p^q l'ensemble des fonctions continues de $C \subset \mathbb{R}^p \longrightarrow \mathbb{R}^q$ avec C compact muni de la norme $\|f\| = \sup_{x \in C} \|f(x)\|$ Alors E_p^q est dense dans F_p^q.

La démonstration de ce théorème nécessite deux lemmes. Ceux-ci utilisent la définition usuelle du produit scalaire sur $\mathbb{R}^p$ défini par $(x,y) = (x_1,\ldots,x_p,y_1,\ldots,y_p) \in \mathbb{R}^{2p} \longrightarrow \langle x,y \rangle = \sum_{i=1}^{p} x_i y_i$. et la norme infinie : $x = (x_1,\ldots,x_p) \in \mathbb{R}^p \longrightarrow \|x\| = \max_{i \in \{1,\cdots,p\}} x_i$. Toutes les normes sont équivalentes [23] sur $\mathbb{R}^p$.

Corollaire C1 : approximation d'une fonction créneau

Soit $C \subset \mathbb{R}^p,\ C = \{(y_1,\ldots,y_p) \in \mathbb{R}^p \,|\forall i \in \{1,\cdots,p\},\ 0 \leqslant y_i \leqslant 1\}$, alors :

$$\forall \varepsilon > 0,\ \forall \alpha > 0,\ \exists n \in \mathbb{N}^*,\ \exists (x_1,\ldots,x_n) \in (\mathbb{R}^p)^n,\ \exists (\gamma_1,\ldots,\gamma_n) \in \mathbb{R}^n \text{ tels que } \forall x \in \mathbb{R}^p,$$

$$\left| \sum_{i=1}^{n} \frac{\gamma_i}{1 + e^{\langle x_i, x \rangle + b_i}} - \mathbb{1}_{\{x \in C\}} \right| \leqslant 1$$

$$\text{et} \quad \inf_{y \in Fr(C)} \|x - y\| > \alpha \Rightarrow \left| \sum_{i=1}^{n} \frac{\gamma_i}{1 + e^{\langle x_i, x \rangle + b_i}} - \mathbb{1}_{\{x \in C\}} \right| \leqslant \varepsilon$$

Démonstration du corollaire

Partie 1

Soit h la fonction définie par : $h(x) = \left(\frac{1}{1+e^{-kx}}\right)^p$ avec $p > 0$ et $0 < \epsilon < 1$. A α, ϵ fixé, $0 < \epsilon < 1$, on cherche k tel que :

$$\begin{aligned}
\epsilon &= h(\alpha) = \left(\frac{1}{1+e^{-k\alpha}}\right)^p \\
\Longrightarrow \quad \epsilon^{-\frac{1}{p}} &= 1 + e^{-k\alpha} \\
\Longrightarrow \quad \epsilon^{-\frac{1}{p}} - 1 &= e^{-k\alpha} \\
\Longrightarrow \quad \ln\left(\epsilon^{-\frac{1}{p}} - 1\right) &= -k\alpha \\
\Longrightarrow \quad k &= -\frac{\ln\left(\epsilon^{-\frac{1}{p}} - 1\right)}{\alpha} = k_0(\epsilon,\alpha,p)
\end{aligned}$$

Partie 2

Soit $\alpha > 0$ et $1 \geqslant \varepsilon > 0,\ k > 0$,

On pose $f(y_1,...,y_p) = \prod_{i=1}^{p} \frac{1}{1+e^{-ky_i}} \prod_{i=1}^{p} \frac{1}{1+e^{-k(1-y_i)}}$ d'après sa définition, $0 \leqslant f(y_1,...,y_p) \leqslant 1$.

23. https://fr.wikipedia.org/wiki/Norme_%C3%A9quivalente

Pour $k \geqslant k_0(\epsilon, \alpha, 2p)$ obtenu dans la partie précédente :

$$\inf_{i \in \{1,\ldots,p\}} [\min\{|y_i|, |1-y_i|\}] > \alpha \Longrightarrow \left\| f(y_1, ..., y_p) - \mathbb{1}_{\{x \in C\}} \right\| \leqslant \varepsilon$$

Partie 3

Soit g la fonction définie par :

$$\begin{aligned} g(x) &= \left(\frac{1}{1+e^{-kx}}\right)\left(\frac{1}{1+e^{-k(1-x)}}\right) = \frac{1}{1+e^{-kx}+e^{-k(1-x)}+e^{-k}} \\ &= \frac{1}{1+e^{-kx}+e^{-k}e^{kx}+e^{-k}} = \frac{e^{kx}}{e^{kx}\left(1+e^{-k}\right)+1+e^{-k}e^{2kx}} \end{aligned}$$

La fonction $x \longrightarrow e^{kx}\left(1+e^{-k}\right)+1+e^{-k}e^{2kx}$ est un polynôme en e^{kx} dont le discriminant est positif. Par conséquent la fraction rationnelle $g(x)$ admet une décomposition en éléments simples du premier ordre et il existe quatre réels η_1, η_2, δ_1, δ_2 tels que :

$$g(x) = \frac{\eta_1}{1+e^{kx+\delta_1}} + \frac{\eta_2}{1+e^{kx+\delta_2}}$$

Par conséquent :

$$f(y_1, \ldots, y_p) = \prod_{i=1}^{p} g(y_i) = \prod_{i=1}^{p} \left[\frac{\eta_1^i}{1+e^{ky_i+\delta_1^i}} + \frac{\eta_2^i}{1+e^{ky_i+\delta_2^i}}\right]$$

Il existe $n \in \mathbb{N}$ tel qu'il soit possible d'écrire f sous la forme :

$$f(y) = \sum_{i=1}^{n} \frac{\gamma_i}{1+e^{<x_i,y>+b_i}}$$

Corollaire C2 : approximation d'une fonction indicatrice

Soit $C \subset \mathbb{R}^p$ compact, alors :

$$\forall \varepsilon > 0,\ \forall \alpha > 0,\ \exists (x_1, ..., x_n) \in (\mathbb{R}^p)^n,\ \exists (b_1, ..., b_n) \in \mathbb{R}^n \text{ tels que } \forall x \in \mathbb{R}^p,$$

$$\left| \sum_{i=1}^{n} \frac{\gamma_i}{1+e^{\langle x_i, x\rangle + b_i}} - \mathbb{1}_{\{x \in C\}} \right| \leqslant 1 + 2\varepsilon^2$$

$$\text{et} \quad \inf_{y \in Fr(C)} \|x-y\| > \alpha \Rightarrow \left| \sum_{i=1}^{n} \frac{\gamma_i}{1+e^{\langle x_i, x\rangle + b_i}} - \mathbb{1}_{\{x \in C\}} \right| \leqslant \varepsilon$$

Démonstration du corollaire

Partie 1

Soit $C_1 = \{y = (y_1, ..., y_p) \in \mathbb{R}^p \,|\, \forall i \in \{1, ..., n\},\ 0 \leqslant y_i \leqslant 1\}$ et $C_2^j = \{y = (y_1, ..., y_p) \in \mathbb{R}^p \,|\, \forall i \neq j,\ 0 \leqslant y_i \leqslant 1 \text{ et } 1 \leqslant y_j \leqslant 2\}$

Le premier lemme suggère que la fonction cherchée pour ce lemme dans le cas particulier $C_1 \cup C_2^j$ est :

$$\begin{aligned}
f(y_1,...,y_p) &= \prod_{i=1}^p \frac{1}{1+e^{-ky_i}} \prod_{i=1}^p \frac{1}{1+e^{-k(1-y_i)}} + \\
&\quad \left(\prod_{i\neq j} \frac{1}{1+e^{-ky_i}}\right)\left(\prod_{i\neq j} \frac{1}{1+e^{-k(1-y_i)}}\right) \frac{1}{1+e^{k(1-y_j)}} \frac{1}{1+e^{-k(2-y_j)}} \\
&= \left(\prod_{i\neq j} \frac{1}{1+e^{-ky_i}}\right)\left(\prod_{i\neq j} \frac{1}{1+e^{-k(1-y_i)}}\right) \\
&\quad \left(\frac{1}{1+e^{-ky_j}} \frac{1}{1+e^{-k(1-y_j)}} + \frac{1}{1+e^{k(1-y_j)}} \frac{1}{1+e^{-k(2-y_j)}}\right) \\
&= \left(\prod_{i\neq j} \frac{1}{1+e^{-ky_i}}\right)\left(\prod_{i\neq j} \frac{1}{1+e^{-k(1-y_i)}}\right) \\
&\quad \left[\frac{1}{1+e^{-ky_j}}\left(\frac{1}{1+e^{-k(1-y_j)}} + 1 - 1\right) + \left(1 - \frac{1}{1+e^{-k(1-y_j)}}\right) \frac{1}{1+e^{-k(2-y_j)}}\right]
\end{aligned}$$

Pour $k \geqslant k_0(\epsilon, \alpha, 2p)$, on a :

$$\begin{aligned}
f(y_1,...,y_p) &= \left(\prod_{i\neq j} \frac{1}{1+e^{-ky_i}}\right)\left(\prod_{i\neq j} \frac{1}{1+e^{-k(1-y_i)}}\right) \\
&\quad \left(\frac{1}{1+e^{-ky_j}} + \frac{1}{1+e^{-k(2-y_j)}} + \underbrace{\frac{1}{1+e^{k(1-y_j)}} \frac{1}{1+e^{-ky_j}}}_{\leqslant \varepsilon^2} - \underbrace{\frac{1}{1+e^{-k(1-y_j)}} \frac{1}{1+e^{-k(2-y_j)}}}_{\leqslant \varepsilon^2}\right)
\end{aligned}$$

Par conséquent, il est facile de construire la fonction cherchée pour tout compact connexe par arc.

Partie 2

Si un compact C n'est pas connexe par arc, on peut le recouvrir par une somme finie de compacts connexes par arcs et disjoints $(C_k)_{1\leqslant k\leqslant K}$ de telle sorte que :

$$\forall y \in \bigcup_{k=1}^K C_k, \inf\{\|x-y\|, x \in C\} \leqslant \frac{\alpha}{2}$$

Démontration du théorème de *densité des réseaux de neurones* (page 35)

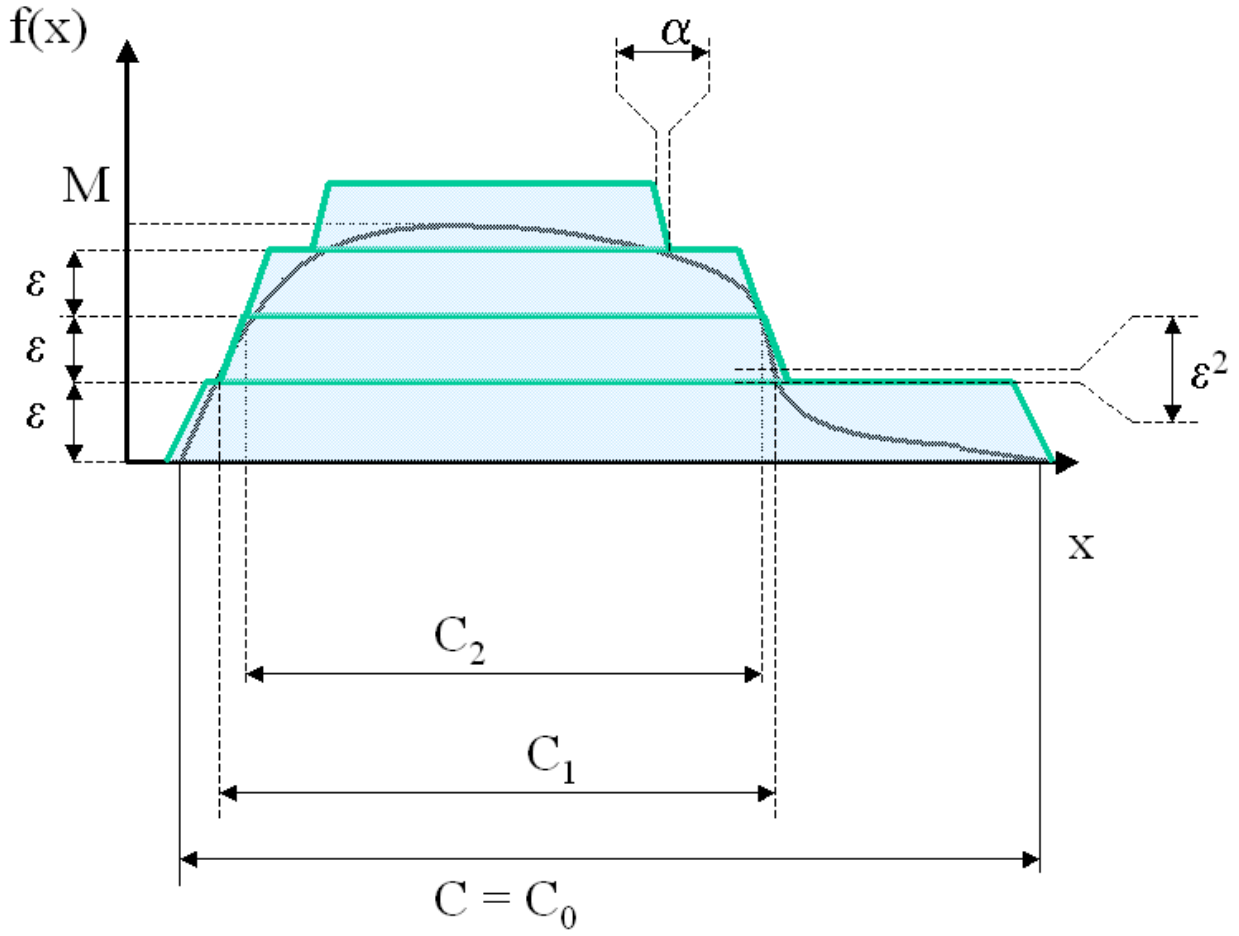
Partie 1

On démontre le théorème dans le cas où $q = 1$. Soit f une fonction continue du compact $C \subset \mathbb{R}^p \to \mathbb{R}$ et soit $\varepsilon > 0$.

On suppose également que f est positive, dans le cas contraire, on pose $f = \underbrace{f - \inf f}_{\text{fonction positive}} + \inf f$.

Si f est nulle, alors c'est fini, sinon, on pose $M = \sup_{x\in C} f(x)$. M existe car f est continue et C est compact (de même, $\inf f$ existe également).

On pose $C_k = f^{-1}([k\varepsilon, M])$. C_k est compact car il est l'image réciproque d'un compact par une fonction continue et $C_k \subset C$ compact.



Par construction, $C_{k+1} \subset C_k$ et $C = \bigcup_{k=0}^{\frac{M}{\varepsilon}} C_k = C_0$ on définit~ :

$$\forall x \in C,\ g_\varepsilon(x) = \varepsilon \sum_{k=0}^{\frac{M}{\varepsilon}} \mathbb{1}_{\{x \in C_k\}}$$

D'où~ :

$$\begin{aligned} f(x) - g_\varepsilon(x) &= f(x) - \varepsilon \sum_{k=0}^{\frac{M}{\varepsilon}} \mathbb{1}_{\{x \in C_k\}} = f(x) - \varepsilon \sum_{k=0}^{\frac{M}{\varepsilon}} \mathbb{1}_{\{f(x) \geqslant k\varepsilon\}} \\ &= f(x) - \varepsilon \left[\frac{f(x)}{\varepsilon} \right] \quad \text{(partie entière)} \\ \text{d'où} \quad & 0 \leqslant f(x) - g_\varepsilon(x) \leqslant \frac{\varepsilon}{4} \end{aligned} \tag{3.3}$$

Comme f est continue sur un compact, elle est uniformément continue sur ce compact :

$$\exists \alpha > 0 \text{ tel que } \forall (x, y) \in C^2,\ \|x - y\| \leqslant \alpha \Longrightarrow |f(x) - f(y)| \leqslant \tfrac{\varepsilon}{2}$$

$$\text{d'où } |f(x) - f(y)| \geqslant \varepsilon \Longrightarrow \|x - y\| > \alpha$$

Par conséquent :

$$\inf \{ \|x - y\| \mid x \in Fr(C_k),\ y \in Fr(C_{k+1}) \} > \alpha$$

D'après le second lemme, on peut construire des fonctions $h_k(x) = \sum_{i=1}^{n} \frac{1}{1+e^{\langle x_i^k, x\rangle + b_i^k}}$ telles que :

$$\left(\left\|h_k(x) - \mathbb{1}_{\{x\in C_k\}}\right\| \leqslant 1\right) \text{ et } \left(\inf_{y\in Fr(C)} \|x-y\| > \frac{\alpha}{2} \Rightarrow \left\|h_k(x) - \mathbb{1}_{\{x\in C_k\}}\right\| \leqslant \varepsilon^2\right)$$

On en déduit que :

$$\begin{aligned}\left|f(x) - \varepsilon\sum_{k=0}^{\frac{M}{\varepsilon}} h_k(x)\right| &\leqslant |f(x) - g_\varepsilon(x)| + \left|g_\varepsilon(x) - \varepsilon\sum_{k=0}^{\frac{M}{\varepsilon}} h_k(x)\right| \\ &\leqslant \varepsilon + \varepsilon^2\left[\frac{M}{\varepsilon}\right] + 2\varepsilon^2 \\ &\leqslant \varepsilon(M+3)\end{aligned}$$

Comme $\varepsilon\sum_{k=1}^{\frac{M}{\varepsilon}} h_k(x)$ est de la forme désirée, le théorème est démontré dans le cas $q=1$.

Partie 2

Dans le cas $q>1$, on utilise la méthode précédente pour chacune des projections de f dans un repère orthonormé de $\mathbb{R}^q$. Il suffit de sommer sur chacune des dimensions.

Ce théorème montre qu'il est judicieux de modéliser la fonction f dans l'équation *(2)* (page **??**) par un réseau de neurones puisqu'il possible de s'approcher d'aussi près qu'on veut de la fonction $\mathbb{E}(Y|X)$, il suffit d'ajouter des neurones sur la couche cachée du réseau. Ce théorème permet de déduire le corollaire suivant :

Corollaire C3 : famille libre de fonctions

Soit F_p l'ensemble des fonctions continues de $C \subset \mathbb{R}^p \longrightarrow \mathbb{R}$ avec C compact muni de la norme : $\|f\| = \sup_{x\in C} \|f(x)\|$

Alors l'ensemble E_p des fonctions sigmoïdes :

$$E_p = \left\{x \longrightarrow 1 - \frac{2}{1+e^{<y,x>+b}} | y\in\mathbb{R}^p \text{ et } b\in\mathbb{R}\right\}$$

est une base de F_p.

Démonstration du corollaire

Le théorème de *densité* (page 35) montre que la famille E_p est une famille génératrice. Il reste à montrer que c'est une famille libre. Soient $(y_i)_{1\leqslant i\leqslant N} \in (\mathbb{R}^p)^N$ et $(b_i)_{1\leqslant i\leqslant N} \in \mathbb{R}^N$ vérifiant : $i\neq j \Longrightarrow y_i \neq y_j$ ou $b_i \neq b_j$. Soit $(\lambda_i)_{1\leqslant i\leqslant N} \in \mathbb{R}^N$, il faut montrer que :

$$\forall x\in\mathbb{R}^p,\ \sum_{i=1}^{N} \lambda_i \left(1 - \frac{2}{1+e^{<y_i,x>+b_i}}\right) = 0 \Longrightarrow \forall i\ \lambda_i = 0 \tag{3.4}$$

C'est évidemment vrai pour $N=1$. La démonstration est basée sur un raisonnement par récurrence, on suppose qu'elle est vraie pour $N-1$, démontrons qu'elle est vraie pour N. On suppose donc $N\geqslant 2$. S'il existe $i\in\{1,\ldots,N\}$ tel que $y_i = 0$, la fonction $x \longrightarrow 1 - \frac{2}{1+e^{<y_i,x>+b_i}}$ est une constante, par conséquent, dans ce cas le corollaire est est vrai pour N. Dans le cas contraire, $\forall i\in\{1,\ldots,N\},\ y_i \neq 0$. On définit les vecteurs $X_i = (x_i, 1)$ et $Y_i = (y_j, b_j)$. On cherche à résoude le système de N équations à N inconnues :

$$\left\{\begin{array}{rcl} \sum_{j=1}^{N} \lambda_j \left(1 - \frac{2}{1+e^{<Y_j,X_1>}}\right) &=& 0 \\ \cdots & & \\ \sum_{j=1}^{N} \lambda_j \left(1 - \frac{2}{1+e^{<Y_j,X_i>}}\right) &=& 0 \\ \cdots & & \\ \sum_{j=1}^{N} \lambda_j \left(1 - \frac{2}{1+e^{<Y_j,X_N>}}\right) &=& 0 \end{array}\right. \tag{3.5}$$

On note le vecteur $\Lambda = (\lambda_i)_{1 \leqslant i \leqslant N}$ et M la matrice :

$$M = (m_{ij})_{1 \leqslant i,j \leqslant N} = \left(1 - \frac{2}{1 + e^{<Y_j, X_i>}}\right)_{1 \leqslant i,j \leqslant N}$$

L'équation *(4)* (page **??**) est équivalente à l'équation matricielle : $M\Lambda = 0$. On effectue une itération du pivot de Gauss. *(4)* (page **??**) équivaut à :

$$\iff \left\{ \begin{array}{ccccccccccc} \lambda_1 m_{11} & + & \lambda_2 & m_{12} & & + & \ldots & + & \lambda_N & m_{1N} & = 0 \\ 0 & + & \lambda_2 & (m_{22}m_{11} - m_{12}m_{21}) & & + & \ldots & + & \lambda_N & (m_{2N}m_{11} - m_{1N}m_{21}) & = 0 \\ \ldots & & & & & & & & & & \\ 0 & + & \lambda_2 & (m_{N2}m_{11} - m_{12}m_{N1}) & & + & \ldots & + & \lambda_N & (m_{NN}m_{11} - m_{1N}m_{N1}) & = 0 \end{array} \right.$$

On note $\Lambda_* = (\lambda_i)_{2 \leqslant i \leqslant N}$ et Δ_*, M_* les matrices :

$$\begin{array}{rcl} M_* & = & (m_{ij})_{2 \leqslant i,j \leqslant N} \\ \Delta_* & = & (m_{1j}\, m_{i1})_{2 \leqslant i,j \leqslant N} \end{array}$$

Donc *(4)* (page **??**) est équivalent à :

$$\iff \left\{ \begin{array}{ccccc} \lambda_1 m_{11} & + & \lambda_2 m_{12} + \ldots + \lambda_N m_{1N} & = 0 \\ 0 & + & (m_{11} M_* - \Delta_*)\, \Lambda_* & = 0 \end{array} \right. \tag{3.6}$$

Il est possible de choisir $X_1(\alpha) = (\alpha x_1, 1)$ de telle sorte qu'il existe une suite $(s_l)_{1 \leqslant l \leqslant N} \in \{-1, 1\}^N$ avec $s_1 = 1$ et vérifiant :

$$\forall j \in (1, \ldots, N), \lim_{\alpha \longrightarrow +\infty} \left[1 - \frac{2}{1 + e^{<Y_j,\, X_1(\alpha)>}}\right] = \lim_{\alpha \longrightarrow +\infty} m_{1j}(\alpha) = s_j$$

On définit :

$$\begin{array}{rcl} U_* & = & (m_{21}, \ldots, m_{N1})' \\ V_* & = & (s_2\, m_{21}, \ldots, s_N\, m_{N1})' \\ \text{et la matrice } L_* & = & (V_*)_{2 \leqslant i \leqslant N} \text{ dont les } N-1 \text{ colonnes sont identiques} \end{array}$$

On vérifie que :

$$\lim_{\alpha \longrightarrow +\infty} \Delta(\alpha) = V_*$$

On obtient, toujours pour *(4)* (page **??**) :

$$\iff \left\{ \begin{array}{ccccc} \lambda_1 m_{11}(\alpha) & + & \lambda_2 m_{12}(\alpha) + \ldots + \lambda_N m_{1N}(\alpha) & = 0 \\ 0 & + & [m_{11}(\alpha)\, M_* - (L_* + (\Delta_*(\alpha) - L_*))]\, \Lambda_* & = 0 \end{array} \right. \tag{3.7}$$

$$\iff \left\{ \begin{array}{ccccc} \lambda_1 m_{11}(\alpha) & + & \lambda_2 m_{12}(\alpha) + \ldots + \lambda_N m_{1N}(\alpha) & = 0 \\ 0 & + & (m_{11}(\alpha)\, M_* - L_*)\, \Lambda_* + (\Delta_*(\alpha) - L_*)\, \Lambda_* & = 0 \end{array} \right.$$

On étudie la limite lorsque $\alpha \longrightarrow +\infty$:

$$\begin{array}{rrcl} & (\Delta_*(\alpha) - L_*) & \underset{\alpha \to +\infty}{\longrightarrow} & 0 \\ \Longrightarrow & (m_{11}(\alpha)\, M_* - L_*)\, \Lambda_* & \underset{\alpha \to +\infty}{\longrightarrow} & 0 \\ \Longrightarrow & (M_* - L_*)\, \Lambda_* & = & 0 \\ \Longrightarrow & M_*\Lambda_* - \left(\sum_{j=2}^N \lambda_j\right) V_* & = & 0 \end{array}$$

Donc :

$$M_*\Lambda_* - \left(\sum_{j=2}^N \lambda_j\right) V_* = 0$$

D'après l'hypothèse de récurrence, *(7)* (page **??**) implique que : $\forall i \in \{2, \ldots, N\}\,,\ \lambda_i = 0$. Il reste à montrer que λ_1 est nécessairement nul ce qui est le cas losque $\alpha \longrightarrow +\infty$, alors $\lambda_1 m_{11}(\alpha) \longrightarrow \lambda_1 = 0$. La récurrence est démontrée.

A chaque fonction sigmoïde du corollaire *famille libre* (page 39) correspond un neurone de la couche cachée. Tous ont des rôles symétriques les uns par rapport aux autres ce qui ne serait pas le cas si les fonctions de transfert étaient des polynômes. C'est une des raisons pour lesquelles les réseaux de neurones ont du succès. Le théorème *densité* (page 35) et le corollaire *famille libre* (page 39) sont aussi vraies pour des fonctions du type exponentielle : $(y, b) \in \mathbb{R}^p \times \mathbb{R} \longrightarrow e^{-(<y,x>+b)^2}$. Maintenant qu'il est prouvé que les réseaux de neurones conviennent pour modéliser f dans l'équation *(2)* (page **??**), il reste à étudier les méthodes qui permettent de trouver les paramètres W^* optimaux de cette fonction.

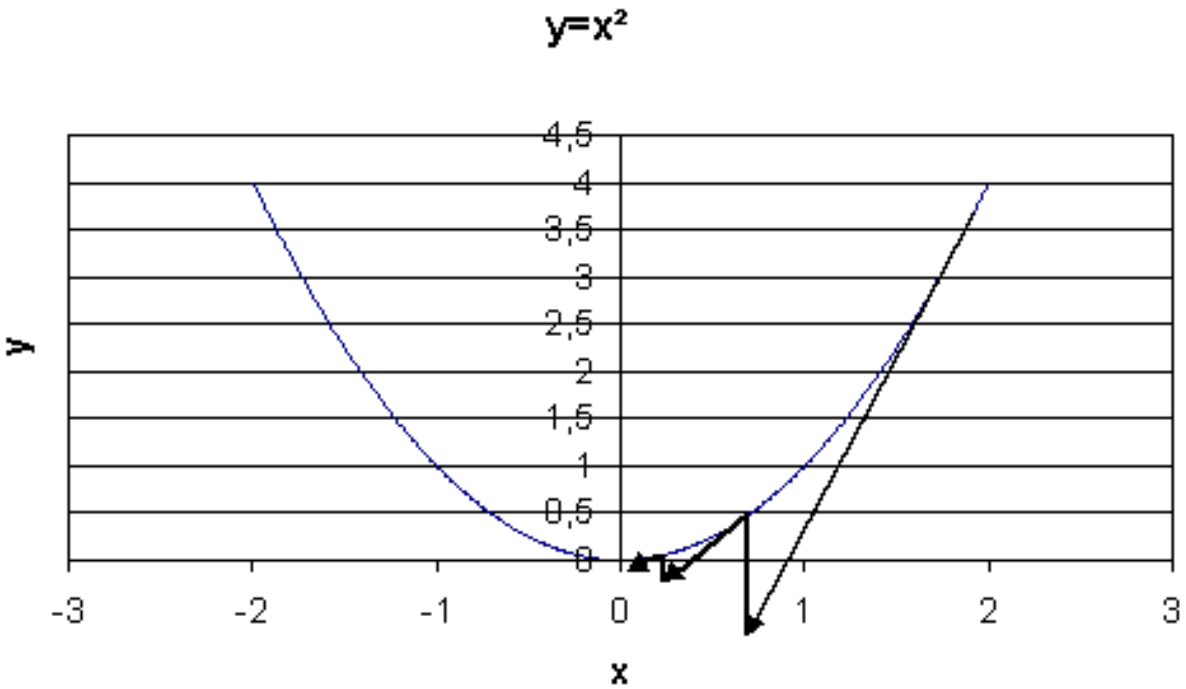
3.1.5 Descente de gradient

Lorsqu'un problème d'optimisation n'est pas soluble de manière déterministe, il existe des algorithmes permettant de trouver une solution approchée à condition toutefois que la fonction à maximiser ou minimiser soit dérivable, ce qui est le cas des réseaux de neurones. Plusieurs variantes seront proposées regroupées sous le terme de descente de gradient. Quelques lectures :

— An overview of gradient descent optimization algorithms [24]
— Implementing a Neural Network from Scratch in Python – An Introduction [25]

Algorithme et convergence

Soit $g : \mathbb{R} \to \mathbb{R}$ une fonction dérivable dont il faut trouver $\overset{*}{x} = \underset{x \in \mathbb{R}}{\arg\min}\ g(x)$, le schéma suivant illustre la méthode de descente de gradient dans le cas où $g(x) = x^2$.



24. http://sebastianruder.com/optimizing-gradient-descent/
25. http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/

On note x_t l'abscisse à l'itération t. On note $\frac{\partial g(x_t)}{\partial x}$ le gradient de $g(x) = x^2$. L'abscisse à l'itération $t+1$ sera $x_{t+1} = x_t - \varepsilon_t \left[\frac{\partial g(x_t)}{\partial x}\right]$. ε_t est le pas de gradient à l'itération t.

On suppose maintenant que g est une fonction dérivable $g : \mathbb{R}^q \to \mathbb{R}$ dont il faut trouver le minimum, le théorème suivant démontre la convergence de l'algorithme de descente de gradient à condition que certaines hypothèses soient vérifiées. Une généralisation de ce théorème est présentée dans *[Driancourt1996]* (page **??**).

Théorème T1 : convergence de la méthode de Newton

[Bottou1991] (page **??**)

Soit une fonction continue $g : W \in \mathbb{R}^M \to \mathbb{R}$ de classe C^1. On suppose les hypothèses suivantes vérifiées :

- **H1** : $\underset{W \in \mathbb{R}^q}{\arg\min}\, g(W) = \{W^*\}$ est un singleton
- **H2** : $\forall \varepsilon > 0,\ \inf_{|W - W^*| > \varepsilon} \left[(W - W^*)'.\nabla g(W)\right] > 0$
- **H3** : $\exists (A, B) \in \mathbb{R}^2$ tels que $\forall W \in \mathbb{R}^p,\ \|\nabla g(W)\|^2 \leqslant A^2 + B^2 \|W - W^*\|^2$
- **H4** : la suite $(\varepsilon_t)_{t \geqslant 0}$ vérifie, $\forall t > 0,\ \varepsilon_t \in \mathbb{R}_+^*$ et $\sum_{t \geqslant 0} \varepsilon_t = +\infty$, $\sum_{t \geqslant 0} \varepsilon_t^2 < +\infty$

Alors la suite $(W_t)_{t \geqslant 0}$ construite de la manière suivante $W_0 \in \mathbb{R}^M$, $\forall t \geqslant 0 : W_{t+1} = W_t - \varepsilon_t \nabla g(W_t)$ vérifie $\lim_{t \to +\infty} W_t = W^*$.

L'hypothèse **H1** implique que le minimum de la fonction g est unique et l'hypothèse **H2** implique que le demi-espace défini par l'opposé du gradient contienne toujours le minimum de la fonction g. L'hypothèse **H3** est vérifiée pour une fonction sigmoïde, elle l'est donc aussi pour toute somme finie de fonctions sigmoïdes que sont les réseaux de neurones à une couche cachée.

Démonstration du théorème

Partie 1

Soit la suite $u_t = \ln\left(1 + \varepsilon_t^2 x^2\right)$ avec $x \in \mathbb{R}$, comme $\sum_{t \geqslant 0} \varepsilon_t^2 < +\infty,\ u_t \sim \varepsilon_t^2 x^2$, on a $\sum_{t \geqslant 0} u_t < +\infty$.

Par conséquent, si $v_t = e^{u_t}$ alors $\prod_{t=1}^{T} v_t \overset{T \to \infty}{\longrightarrow} D \in \mathbb{R}$.

Partie 2

On pose $h_t = \|W_t - W^*\|^2$. Donc :

$$h_{t+1} - h_t = \|W_t - \varepsilon_t \nabla g(W_t) - W^*\|^2 - \|W_t - W^*\|^2 \tag{3.8}$$

Par conséquent :

$$h_{t+1} - h_t = -2\varepsilon_t \underbrace{(W_t - W^*)' \nabla g(W_t)}_{>0} + \varepsilon_t^2 \|\nabla C(W_t)\|^2 \leqslant \varepsilon_t^2 \|\nabla g(W_t)\|^2 \leqslant \varepsilon_t^2 \left(A^2 + B^2 h_t\right)$$

D'où :

$$h_{t+1} - h_t \left(1 + \varepsilon_t^2 B^2\right) \leqslant \varepsilon_t^2 A^2$$

On pose $\pi_t = \prod_{k=1}^{t} \left(1 + \varepsilon_k^2 B^2\right)^{-1}$ alors en multipliant des deux côtés par π_{t+1}, on obtient :

$$\begin{array}{rcl} \pi_{t+1} h_{t+1} - \pi_t h_t & \leqslant & \varepsilon_t^2 A^2 \pi_{t+1} \\ \text{d'où } \pi_{q+1} h_{q+1} - \pi_p h_p & \leqslant & \sum_{t=p}^{q} \varepsilon_t^2 A^2 \pi_{t+1} \leqslant \sum_{t=p}^{q} \varepsilon_t^2 A^2 \Pi \leqslant \sum_{t=p}^{q} \varepsilon_t^2 A^2 \Pi \underset{t \longrightarrow \infty}{\longrightarrow} 0 \end{array}$$

Comme la série $\sum_t (\pi_{t+1} h_{t+1} - \pi_t h_t)$ vérifie le critère de Cauchy, elle est convergente. Par conséquent :

$$\lim_{q \to \infty} \pi_{q+1} h_{q+1} = 0 = \lim_{q \to \infty} \Pi h_{q+1}$$

D'où $\lim_{q \to \infty} h_q = 0$.

Partie 3

La série $\sum_t (h_{t+1} - h_t)$ est convergente car $\Pi h_t \sim \pi_t h_t$. $\sum_{t \geqslant 0} \varepsilon_t^2 \parallel \nabla g(W_t) \parallel^2$ l'est aussi (d'après **H3**).

D'après *(1)* (page **??**), la série $\sum_{t \geqslant 0} \varepsilon_t (W_t - W^*)' \nabla g(W_t)$ est donc convergente. Or d'après les hypothèses **H2**, **H4**, elle ne peut l'être que si :

$$\lim_{t \to \infty} W_t = W^* \tag{3.9}$$

Si ce théorème prouve la convergence de la méthode de Newton, il ne précise pas à quelle vitesse cette convergence s'effectue et celle-ci peut parfois être très lente. Plusieurs variantes ont été développées regroupées sous le terme de méthodes de quasi-Newton dans le but d'améliorer la vitesse de convergence.

Ce théorème peut être étendu dans le cas où la fonction g n'a plus un seul minimum global mais plusieurs minima locaux (*[Bottou1991]* (page **??**)), dans ce cas, la suite (W_t) converge vers un mimimum local. Dans le cas des réseaux de neurones, la fonction à optimiser est :

$$G(W) = \sum_{i=1}^{N} e\left(Y_i, \widehat{Y_i^W}\right) = \sum_{i=1}^{N} e\left(Y_i, f(W, X_i)\right)$$

Dès que les fonctions de transfert ne sont pas linéaires, il existe une multitude de minima locaux, ce nombre croissant avec celui des coefficients.

Calcul du gradient ou *rétropropagation*

Afin de minimiser la fonction G décrite en *(2)* (page **??**), l'algorithme de descente du gradient nécessite de calculer le gradient de cette fonction G qui est la somme des gradients $\frac{\partial e}{\partial W}$ pour chaque couple (X_i, Y_i) :

$$\begin{aligned} \frac{\partial G}{\partial W}(W) &= \sum_{i=1}^{N} \frac{\partial e\left(Y_i, f(W, X_i)\right)}{\partial W} \\ &= \sum_{i=1}^{N} \sum_{k=1}^{C_C} \frac{\partial e\left(Y_i, f(W, X_i)\right)}{\partial z_{C,k}} \frac{\partial z_{C,k}}{\partial W} \end{aligned}$$

Les notations utilisées sont celles de la figure du *perceptron* (page 27). Les résultats qui suivent sont pour $X_i = X$ donné appartenant à la suite (X_i). On remarque tout d'abord que :

$$\begin{aligned} \frac{\partial e}{\partial w_{c,i,j}}(W, X) &= z_{c-1,j} \frac{\partial e}{\partial y_{c,i}}(W, X) \\ \frac{\partial e}{\partial b_{c,i}}(W, X) &= \frac{\partial e}{\partial y_{c,i}}(W, X) \end{aligned}$$

La rétropropagation du gradient consiste donc à calculer les termes : $\frac{\partial e}{\partial y_{.,.}}(W, X)$ puisque le gradient s'en déduit facilement. La dernière couche du réseau de neurones nous permet d'obtenir :

$$\begin{aligned} \frac{\partial e}{\partial y_{C,i}}(W, X) &= \sum_{k=1}^{C_C} \frac{\partial e}{\partial z_{C,k}}(W, X) \frac{\partial z_{C,k}}{\partial y_{C,i}}(W, X) \\ &= \frac{\partial e}{\partial z_{C,i}}(W, X) f'_{c,i}(y_{C,i}) \end{aligned}$$

Pour les autres couches c telles que $1 \leqslant c \leqslant C-1$, on a :

$$\begin{aligned}
\frac{\partial e}{\partial y_{c,i}} &= \sum_{l=1}^{C_{c+1}} \frac{\partial e}{\partial y_{c+1,l}} \frac{\partial y_{c+1,l}}{\partial y_{c,i}} \\
&= \sum_{l=1}^{C_{c+1}} \frac{\partial e}{\partial y_{c+1,l}} \left[\sum_{l=1}^{C_c} \frac{\partial y_{c+1,l}}{\partial z_{c,l}} \underbrace{\frac{\partial z_{c,l}}{\partial y_{c,i}}}_{=0 \; si \; l \neq i} \right] \\
&= \sum_{l=1}^{C_{c+1}} \frac{\partial e}{\partial y_{c+1,l}} \frac{\partial y_{c+1,l}}{\partial z_{c,i}} \frac{\partial z_{c,i}}{\partial y_{c,i}}
\end{aligned}$$

Par conséquent :

$$\frac{\partial e}{\partial y_{c,i}} = \left[\sum_{l=1}^{C_{c+1}} \frac{\partial e}{\partial y_{c+1,l}} w_{c+1,l,i} \right] f'_{c,i}\left(y_{c,i}\right)$$

Cette dernière formule permet d'obtenir par récurrence les dérivées $\frac{\partial e}{\partial y_{.,.}}$ de la dernière couche C à la première et ce, quel que soit le nombre de couches. Cette récurrence inverse de la propagation est appelée *rétropropagation*. Cet algorithme se déduit des équations *(3)* (page **??**), *(4)* (page **??**), *(5)* (page **??**) et *(7)* (page **??**) :

Théorème T2 : rétropropagation

Cet algorithme s'applique à un réseau de neurones vérifiant la définition du *perceptron* (page 26). Il s'agit de calculer sa dérivée par rapport aux poids. Il se déduit des formules *(3)* (page **??**), *(4)* (page **??**), *(5)* (page **??**) et *(7)* (page **??**) et suppose que l'algorithme de *propagation* (page 27) a été préalablement exécuté. On note $y'_{c,i} = \frac{\partial e}{\partial y_{c,i}}$, $w'_{c,i,j} = \frac{\partial e}{\partial w_{c,i,j}}$ et $b'_{c,i} = \frac{\partial e}{\partial b_{c,i}}$.

Initialisation

for i in $1..C_C$
    $y'_{C,i} \longleftarrow \frac{\partial e}{\partial z_{C,i}}(W,X)\, f'_{c,i}\left(y_{C,i}\right)$

Récurrence

for c in $1..C-1$
    for i in $1..C_c$
        $y'_{c,i} \longleftarrow 0$
        for j in $1..C_{c+1}$
            $y'_{c,i} \longleftarrow y'_{c,i} + y'_{c+1,j}\; w_{c+1,j,i}$
        $y'_{c,i} \longleftarrow y'_{c,i}\; f'_{c,i}\left(y'_{c,i}\right)$

Terminaison

for c in $1..C$
    for i in $1..C_c$
        for j in $1..C_{c-1}$
            $w'_{c,i,j} \longleftarrow z_{c-1,j}\; y'_{c,i}$

$$b'_{c,i,j} \longleftarrow y'_{c,i}$$

3.1.6 Apprentissage d'un réseau de neurones

Le terme apprentissage est encore inspiré de la biologie et se traduit par la minimisation de la fonction *(2)* (page **??**) où f est un réseau de neurone défini par un *perceptron* (page 26). Il existe plusieurs méthodes pour effectuer celle-ci. Chacune d'elles vise à minimiser la fonction d'erreur :

$$E(W) = G(W) = \sum_{i=1}^{N} e\left(Y_i - \widehat{Y_i^W}\right) = \sum_{i=1}^{N} e\left(Y_i - f\left(W, X_i\right)\right)$$

Dans tous les cas, les différents apprentissages utilisent la suite suivante (ϵ_t) vérifiant *(1)* (page **??**) et proposent une convergence vers un minimum local.

$$\forall t > 0, \quad \varepsilon_t \in \mathbb{R}_+^* \text{ et } \sum_{t \geqslant 0} \varepsilon_t = +\infty, \quad \sum_{t \geqslant 0} \varepsilon_t^2 < +\infty \tag{3.10}$$

Il est souhaitable d'apprendre plusieurs fois la même fonction en modifiant les conditions initiales de ces méthodes de manière à améliorer la robustesse de la solution.

Apprentissage avec gradient global

L'algorithme de *rétropropagation* (page 44) permet d'obtenir la dérivée de l'erreur e pour un vecteur d'entrée X. Or l'erreur $E(W)$ à minimiser est la somme des erreurs pour chaque exemple X_i, le gradient global $\frac{\partial E(W)}{\partial W}$ de cette erreur globale est la somme des gradients pour chaque exemple (voir équation *(3)* (page **??**)). Parmi les méthodes d'optimisation basées sur le gradient global, on distingue deux catégories :

— Les méthodes du premier ordre, elles sont calquées sur la méthode de Newton [26] et n'utilisent que le gradient.
— Les méthodes du second ordre ou méthodes utilisant un gradient conjugué [27] elles sont plus coûteuses en calcul mais plus performantes puisque elles utilisent la dérivée seconde ou une valeur approchée.

Méthodes du premier ordre

Les méthodes du premier ordre sont rarement utilisées. Elles sont toutes basées sur le principe de la descente de gradient de Newton présentée dans la section *Algorithme et convergence* (page 41) :

Algorithme A1 : optimisation du premier ordre

Initialiation

Le premier jeu de coefficients W_0 du réseau de neurones est choisi aléatoirement.

$$\begin{array}{rcl} t & \longleftarrow & 0 \\ E_0 & \longleftarrow & \sum_{i=1}^{N} e\left(Y_i - f\left(W_0, X_i\right)\right) \end{array}$$

26. https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Newton
27. https://fr.wikipedia.org/wiki/M%C3%A9thode_du_gradient_conjugu%C3%A9

Calcul du gradient

$g_t \longleftarrow \frac{\partial E_t}{\partial W}(W_t) = \sum_{i=1}^{N} e'(Y_i - f(W_t, X_i))$

Mise à jour

$$\begin{array}{rcl} W_{t+1} & \longleftarrow & W_t - \epsilon_t g_t \\ E_{t+1} & \longleftarrow & \sum_{i=1}^{N} e(Y_i - f(W_{t+1}, X_i)) \\ t & \longleftarrow & t+1 \end{array}$$

Terminaison

Si $\frac{E_t}{E_{t-1}} \approx 1$ (ou $\|g_t\| \approx 0$) alors l'apprentissage a convergé sinon retour au calcul du gradient.

La condition d'arrêt peut-être plus ou moins stricte selon les besoins du problème. Cet algorithme converge vers un minimum local de la fonction d'erreur (d'après le théorème de *convergence* (page 42) mais la vitesse de convergence est inconnue.

Méthodes du second ordre

L'algorithme *apprentissage global* (page 45) fournit le canevas des méthodes d'optimisation du second ordre. La mise à jour des coefficients est différente car elle prend en compte les dernières valeurs des coefficients ainsi que les derniers gradients calculés. Ce passé va être utilisé pour estimer une direction de recherche pour le minimum différente de celle du gradient, cette direction est appelée gradient conjugué (voir *[Moré1977]* (page **??**)).

Ces techniques sont basées sur une approximation du second degré de la fonction à minimiser. On note M le nombre de coefficients du réseau de neurones (biais compris). Soit $h : \mathbb{R}^M \rightarrow \mathbb{R}$ la fonction d'erreur associée au réseau de neurones : $h(W) = \sum_i e(Y_i, f(W, X_i))$. Au voisinage de W_0, un développement limité donne :

$$h(W) = h(W_0) + \frac{\partial h(W_0)}{\partial W}(W - W_0) + (W - W_0)' \frac{\partial^2 h(W_0)}{\partial W^2}(W - W_0) + o\|W - W_0\|^2$$

Par conséquent, sur un voisinage de W_0, la fonction $h(W)$ admet un minimum local si $\frac{\partial^2 h(W_0)}{\partial W^2}$ est définie positive strictement.

Rappel : $\frac{\partial^2 h(W_0)}{\partial W^2}$ est définie positive strictement $\iff \forall Z \in \mathbb{R}^N,\ Z \neq 0 \Longrightarrow Z' \frac{\partial^2 h(W_0)}{\partial W^2} Z > 0$.

Une matrice symétrique définie strictement positive est inversible, et le minimum est atteint pour la valeur :

$$W_{\min} = W_0 + \frac{1}{2}\left[\frac{\partial^2 h(W_0)}{\partial W^2}\right]^{-1}\left[\frac{\partial h(W_0)}{\partial W}\right]$$

Néanmoins, pour un réseau de neurones, le calcul de la dérivée seconde est coûteux, son inversion également. C'est pourquoi les dernières valeurs des coefficients et du gradient sont utilisées afin d'approcher cette dérivée seconde ou directement son inverse. Deux méthodes d'approximation sont présentées :

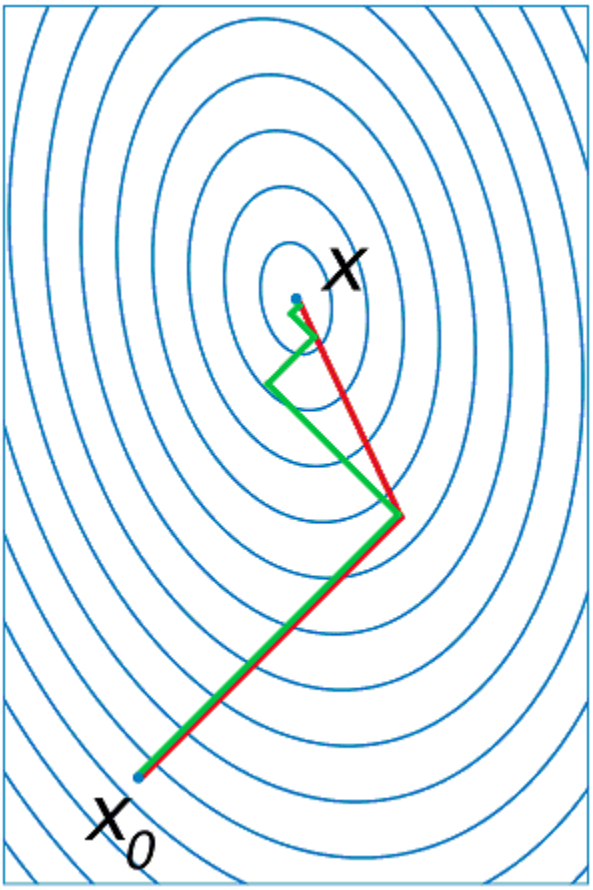
— L'algorithme BFGS (Broyden-Fletcher-Goldfarb-Shano) [28] (*[Broyden1967]* (page **??**), *[Fletcher1993]* (page **??**)), voir aussi les versions L-BFGS [29].

— L'algoritmhe DFP (Davidon-Fletcher-Powell) [30] (*[Davidon1959]* (page **??**), *[Fletcher1963]* (page **??**)).

La figure du *gradient conjugué* (page 47) est couramment employée pour illustrer l'intérêt des méthodes de gradient conjugué. Le problème consiste à trouver le minimum d'une fonction quadratique, par exemple, $G(x, y) = 3x^2 + y^2$. Tandis que le gradient est orthogonal aux lignes de niveaux de la fonction G, le gradient conjugué se dirige plus sûrement vers le minimum global.

Figure F1 : Gradient conjugué

28. https://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%93Shanno_algorithm
29. https://en.wikipedia.org/wiki/Limited-memory_BFGS
30. https://en.wikipedia.org/wiki/Davidon%E2%80%93Fletcher%E2%80%93Powell_formula



Gradient et gradient conjugué sur une ligne de niveau de la fonction $G(x,y) = 3x^2 + y^2$, le gradient est orthogonal aux lignes de niveaux de la fonction G, mais cette direction est rarement la bonne à moins que le point (x,y) se situe sur un des axes des ellipses, le gradient conjugué agrège les derniers déplacements et propose une direction de recherche plus plausible pour le minimum de la fonction. Voir Conjugate Gradient Method [31].

Ces méthodes proposent une estimation de la dérivée seconde (ou de son inverse) utilisée en *(2)* (page **??**). Dans les méthodes du premier ordre, une itération permet de calculer les poids W_{t+1} à partir des poids W_t et du gradient G_t. Si ce gradient est petit, on peut supposer que G_{t+1} est presque égal au produit de la dérivée seconde par G_t. Cette relation est mise à profit pour construire une estimation de la dérivée seconde. Cette matrice notée B_t dans l'algorithme *BFGS* (page 48) est d'abord supposée égale à l'identité puis actualisée à chaque itération en tenant de l'information apportée par chaque déplacement.

Algorithme A2 : BFGS

Le nombre de paramètres de la fonction f est M.

Initialisation

Le premier jeu de coefficients W_0 du réseau de neurones est choisi aléatoirement.

$$\begin{array}{lcl} t & \longleftarrow & 0 \\ E_0 & \longleftarrow & \sum_{i=1}^{N} e\left(Y_i - f\left(W_0, X_i\right)\right) \\ B_0 & \longleftarrow & I_M \\ i & \longleftarrow & 0 \end{array}$$

31. https://en.wikipedia.org/wiki/Conjugate_gradient_method

Calcul du gradient

$$\begin{array}{lll} g_t & \longleftarrow & \frac{\partial E_t}{\partial W}(W_t) = \sum_{i=1}^{N} e'(Y_i - f(W_t, X_i)) \\ c_t & \longleftarrow & B_t g_t \end{array}$$

Mise à jour des coefficients

$$\begin{array}{lll} \epsilon^* & \longleftarrow & \arg\inf_{\epsilon} \sum_{i=1}^{N} e(Y_i - f(W_t - \epsilon c_t, X_i)) \\ W_{t+1} & \longleftarrow & W_t - \epsilon^* c_t \\ E_{t+1} & \longleftarrow & \sum_{i=1}^{N} e(Y_i - f(W_{t+1}, X_i)) \\ t & \longleftarrow & t+1 \end{array}$$

Mise à jour de la marice :math :'B_t'

si $t - i \geqslant M$ ou $g'_{t-1} B_{t-1} g_{t-1} \leqslant 0$ ou $g'_{t-1} B_{t-1} (g_t - g_{t-1}) \leqslant 0$

$B_t \longleftarrow I_M$

$i \longleftarrow t$

sinon

$s_t \longleftarrow W_t - W_{t-1}$

$d_t \longleftarrow g_t - g_{t-1}$

$$B_t \longleftarrow B_{t-1} + \left(1 + \frac{d'_t B_{t-1} d_t}{d'_t s_t}\right) \frac{s_t s'_t}{s'_t d_t} - \frac{s_t d'_t B_{t-1} + B_{t-1} d_t s'_t}{d'_t s_t}$$

Terminaison

Si $\frac{E_t}{E_{t-1}} \approx 1$ alors l'apprentissage a convergé sinon retour au calcul du gradient.

Lorsque la matrice B_t est égale à l'identité, le gradient conjugué est égal au gradient. Au fur et à mesure des itérations, cette matrice toujours symétrique évolue en améliorant la convergence de l'optimisation. Néanmoins, la matrice B_t doit être ńănettoyéeă" (égale à l'identité) fréquemment afin d'éviter qu'elle n'agrège un passé trop lointain. Elle est aussi nettoyée lorsque le gradient conjugué semble trop s'éloigner du véritable gradient et devient plus proche d'une direction perpendiculaire.

La convergence de cet algorithme dans le cas des réseaux de neurones est plus rapide qu'un algorithme du premier ordre, une preuve en est donnée dans *[Driancourt1996]* (page **??**).

En pratique, la recherche de ϵ^* est réduite car le calcul de l'erreur est souvent coûteux, il peut être effectué sur un grand nombre d'exemples. C'est pourquoi on remplace l'étape de mise à jour de l'algorithme *BFGS* (page 48) par celle-ci :

Algorithme A3 : BFGS¿

Le nombre de paramètre de la fonction f est M.

Initialisation, calcul du gradient

Voir *BFGS* (page 48).

Recherche de :math :'epsilon^*'

$\epsilon^* \longleftarrow \epsilon_0$

while $E_{t+1} \geqslant E_t$ et $\epsilon^* \gg 0$

$\epsilon^* \longleftarrow \frac{\epsilon^*}{2}$

$W_{t+1} \longleftarrow W_t - \epsilon^* c_t$

$E_{t+1} \longleftarrow \sum_{i=1}^{N} e(Y_i - f(W_{t+1}, X_i))$

if $\epsilon_* \approx 0$ et $B_t \neq I_M$

$B_t \longleftarrow I_M$

$i \longleftarrow t$

Retour au calcul du gradient.

Mise à jour des coefficients

$$\begin{array}{lll} W_{t+1} & \longleftarrow & W_t - \epsilon^* c_t \\ E_{t+1} & \longleftarrow & \sum_{i=1}^{N} e\left(Y_i - f\left(W_{t+1}, X_i\right)\right) \\ t & \longleftarrow & t+1 \end{array}$$

Mise à jour de la matrice :math :'B_t', temrinaison

Voir *BFGS* (page 48).

L'algorithme DFP est aussi un algorithme de gradient conjugué qui propose une approximation différente de l'inverse de la dérivée seconde.

Algorithme A4 : DFP

Le nombre de paramètre de la fonction f est M.

Initialisation

Le premier jeu de coefficients W_0 du réseau de neurones est choisi aléatoirement.

$$\begin{array}{lll} t & \longleftarrow & 0 \\ E_0 & \longleftarrow & \sum_{i=1}^{N} e\left(Y_i - f\left(W_0, X_i\right)\right) \\ B_0 & \longleftarrow & I_M \\ i & \longleftarrow & 0 \end{array}$$

Calcul du gradient

$$\begin{array}{lll} g_t & \longleftarrow & \frac{\partial E_t}{\partial W}\left(W_t\right) = \sum_{i=1}^{N} e'\left(Y_i - f\left(W_t, X_i\right)\right) \\ c_t & \longleftarrow & B_t g_t \end{array}$$

Mise à jour des coefficients

$$\begin{array}{lll} \epsilon^* & \longleftarrow & \underset{\epsilon}{\arg\inf} \sum_{i=1}^{N} e\left(Y_i - f\left(W_t - \epsilon c_t, X_i\right)\right) \\ W_{t+1} & \longleftarrow & W_t - \epsilon^* c_t \\ E_{t+1} & \longleftarrow & \sum_{i=1}^{N} e\left(Y_i - f\left(W_{t+1}, X_i\right)\right) \\ t & \longleftarrow & t+1 \end{array}$$

Mise à jour de la matrice :math :'B_t'

si $t - i \geqslant M$ ou $g'_{t-1} B_{t-1} g_{t-1} \leqslant 0$ ou $g'_{t-1} B_{t-1}\left(g_t - g_{t-1}\right) \leqslant 0$

$B_t \longleftarrow I_M$

$i \longleftarrow t$

sinon

$d_t \longleftarrow W_t - W_{t-1}$

$s_t \longleftarrow g_t - g_{t-1}$

$B_t \longleftarrow$ B_{t-1} + dfrac{d_t d'_t} {d'_t s_t} - dfrac{B_{t-1} s_t s'_t B_{t-1} } { s'_t B_{t-1} s_t }'

Terminaison

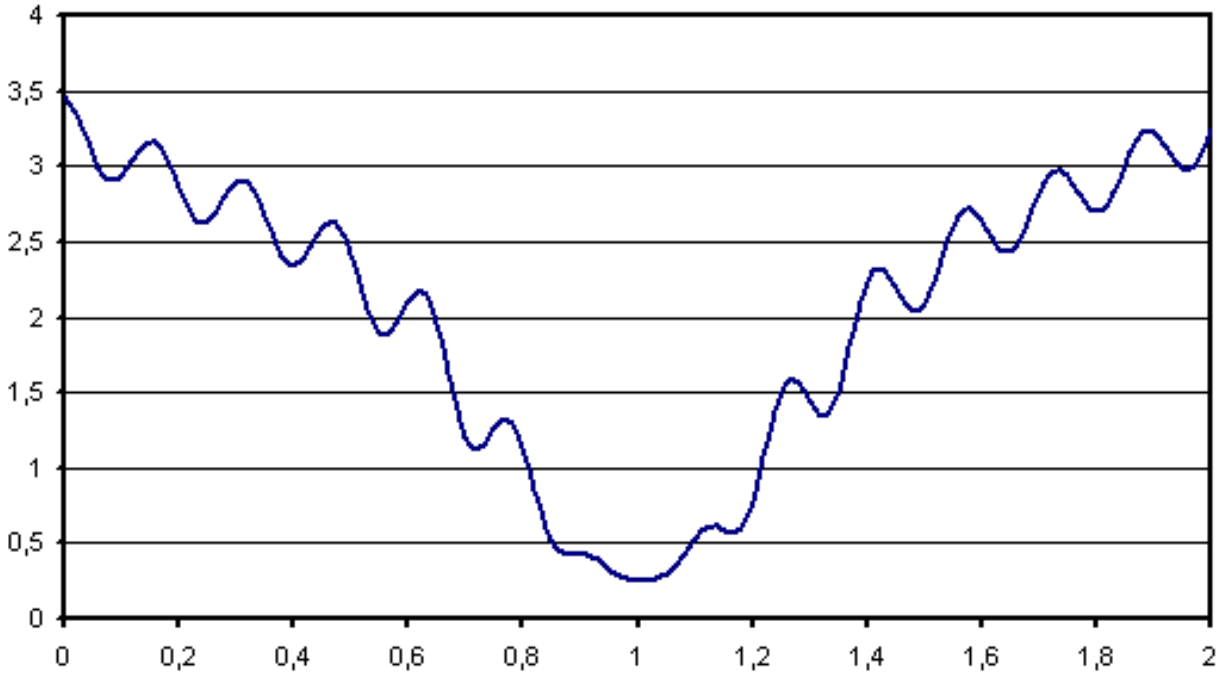
Si $\frac{E_t}{E_{t-1}} \approx 1$ alors l'apprentissage a convergé sinon retour à du calcul du gradient.

Seule l'étape de mise à jour B_t diffère dans les algorithmes *BFGS* (page 48) et *DFP* (page 49). Comme l'algorithme *BFGS* (page 48), on peut construire une version *DFP* (page 49)¿inspirée de l'algorithme *BFGS¿* (page 48).

Apprentissage avec gradient stochastique

Compte tenu des courbes d'erreurs très *accidentées* (page 50) dessinées par les réseaux de neurones, il existe une multitude de minima locaux. De ce fait, l'apprentissage global converge rarement vers le minimum global de la fonction d'erreur lorsqu'on applique les algorithmes basés sur le gradient global. L'apprentissage avec gradient stochastique est une solution permettant de mieux explorer ces courbes d'erreurs. De plus, les méthodes de gradient conjugué nécessite le stockage d'une matrice trop grande parfois pour des fonctions ayant quelques milliers de paramètres. C'est pourquoi l'apprentissage avec gradient stochastique est souvent préféré à l'apprentissage global pour de grands réseaux de neurones alors que les méthodes du second ordre trop coûteuses en calcul sont cantonnées à de petits réseaux. En contrepartie, la convergence est plus lente. La démonstration de cette convergence nécessite l'utilisation de quasi-martingales et est une convergence presque sûre *[Bottou1991]* (page **??**).

Figure F2 : Exemple de minimal locaux



Algprithme A1 : apprentissage stochastique

Initialisation

Le premier jeu de coefficients W_0 du réseau de neurones est choisi aléatoirement.

$$\begin{array}{lcl} t & \longleftarrow & 0 \\ E_0 & \longleftarrow & \sum_{i=1}^{N} e\left(Y_i - f\left(W_0, X_i\right)\right) \end{array}$$

Récurrence

$W_{t,0} \longleftarrow W_0$
for t' in $0..N-1$
    $i \longleftarrow$ nombre aléatoire dans $\{1,\ldots,N\}$
    $g \longleftarrow \frac{\partial E}{\partial W}(W_{t,t'}) = e'(Y_i - f(W_{t,t'}, X_i))$
    $W_{t,t'+1} \longleftarrow W_{t,t'} - \epsilon_t g$
$W_{t+1} \longleftarrow W_{t,N}$
$E_{t+1} \longleftarrow \sum_{i=1}^{N} e(Y_i - f(W_{t+1}, X_i))$
$t \longleftarrow t+1$

Terminaison

Si $\frac{E_t}{E_{t-1}} \approx 1$ alors l'apprentissage a convergé sinon retour au calcul du gradient.

En pratique, il est utile de converser le meilleur jeu de coefficients : $W^* = \underset{u \geqslant 0}{\arg\min}\ E_u$ car la suite $(E_u)_{u \geqslant 0}$ n'est pas une suite décroissante.

3.1.7 Classification

Vraisemblance d'un échantillon de variable suivant une loi multinomiale

Soit $(Y_i)_{1 \leqslant i \leqslant N}$ un échantillon de variables aléatoires i.i.d. suivant la loi multinomiale $\mathcal{M}(p_1,\ldots,p_C)$. On définit $\forall k \in \{1,\cdots,C\},\ d_k = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\{Y_i=k\}}$. La vraisemblance de l'échantillon est :

$$\begin{aligned}
L(Y_1,\ldots,Y_N,p_1,\ldots,p_C) &= \prod_{i=1}^{n} p_{Y_i} \\
\ln L(Y_1,\ldots,Y_N,p_1,\ldots,p_C) &= \sum_{i=1}^{n} \ln p_{Y_i} \\
\ln L(Y_1,\ldots,Y_N,p_1,\ldots,p_C) &= \sum_{k=1}^{C} \left[(\ln p_k) \sum_{i=1}^{N} \mathbb{1}_{\{Y_i=k\}} \right] \\
\ln L(Y_1,\ldots,Y_N,p_1,\ldots,p_C) &= N \sum_{k=1}^{C} d_k \ln p_k
\end{aligned}$$

Cette fonction est aussi appelée distance de Kullback-Leiber [32] (*[Kullback1951]* (page **??**)), elle mesure la distance entre deux distributions de variables aléatoires discrètes. L¿estimateur de maximum de vraisemblance [33] (emv) est la solution du problème suivant :

Problème P1 : estimateur du maximum de vraisemblance

32. https://fr.wikipedia.org/wiki/Divergence_de_Kullback-Leibler
33. https://fr.wikipedia.org/wiki/Maximum_de_vraisemblance

Soit un vecteur $(d_1, \ldots, d_N)$ tel que :

$$\left\{ \begin{array}{l} \sum_{k=1}^{N} d_k = 1 \\ \forall k \in \{1, \ldots, N\}, \; d_k \geqslant 0 \end{array} \right.$$

On cherche le vecteur $(p_1^*, \ldots, p_N^*)$ vérifiant :

$$(p_1^*, \ldots, p_N^*) = \underset{(p_1,\ldots,p_C)\in\mathbb{R}^C}{\arg\max} \sum_{k=1}^{C} d_k \ln p_k$$

$$\text{avec} \left\{ \begin{array}{l} \forall k \in \{1, \cdots, C\}, \; p_k \geqslant 0 \\ \text{et } \sum_{k=1}^{C} p_k = 1 \end{array} \right.$$

Théorème T1 : résolution du problème du maximum de vraisemblance

La solution du problème du *maximum de vraisemblance* (page 52) est le vecteur :

$$(p_1^*, \ldots, p_N^*) = (d_1, \ldots, d_N)$$

Démonstration

Soit un vecteur $(p_1, \ldots, p_N)$ vérifiant les conditions :

$$\left\{ \begin{array}{l} \sum_{k=1}^{N} p_k = 1 \\ \forall k \in \{1, \ldots, N\}, \; p_k \geqslant 0 \end{array} \right.$$

La fonction $x \longrightarrow \ln x$ est concave, d'où :

$$\begin{aligned} \Delta &= \sum_{k=1}^{C} d_k \ln p_k - \sum_{k=1}^{C} d_k \ln d_k \\ &= \sum_{k=1}^{C} d_k (\ln p_k - \ln d_k) = \sum_{k=1}^{C} d_k \ln \frac{p_k}{d_k} \\ &\leqslant \ln \left(\sum_{k=1}^{C} d_k \frac{p_k}{d_k} \right) = \ln \left(\sum_{k=1}^{C} p_k \right) = \ln 1 = 0 \\ &\leqslant 0 \end{aligned}$$

La distance de KullBack-Leiber compare deux distributions de probabilités entre elles. C'est elle qui va faire le lien entre le problème de *classification discret* (page 32) et les réseaux de neurones pour lesquels il faut impérativement une fonction d'erreur dérivable.

Problème de classification pour les réseaux de neurones

Le problème de *classification* (page 32) est un cas particulier de celui qui suit pour lequel il n'est pas nécessaire de connaître la classe d'appartenance de chaque exemple mais seulement les probabilités d'appartenance de cet exemple à chacune des classes.

Soient une variable aléatoire continue $X \in \mathbb{R}^p$ et une variable aléatoire discrète multinomiale $Y \in \{1, \cdots, C\}$, on veut estimer la loi de :

$$Y|X \sim \mathcal{M}\left(p_1\left(W, X\right), \ldots, p_C\left(W, X\right)\right) \text{ avec } W \in \mathbb{R}^M$$

Le vecteur $(p_1(W,X),\dots,p_C(W,X))$ est une fonction f de (W,X) où W est l'ensemble des M paramètres du modèle. Cette fonction possède p entrées et C sorties. Comme pour le problème de la régression, on cherche les poids W qui correspondent le mieux à l'échantillon :

$$A = \left\{ \left(X_i, y_i = \left(\eta_i^k\right)_{1 \leqslant k \leqslant C} \right) \in \mathbb{R}^p \times [0,1]^C \text{ tel que } \sum_{k=1}^{c} y_i^k = 1 \middle| 1 \leqslant i \leqslant N \right\}$$

On suppose que les variables $(Y_i|X_i)_{1 \leqslant i \leqslant N}$ suivent les lois respectives $(\mathcal{M}(y_i))_{1 \leqslant i \leqslant N}$ et sont indépendantes entre elles, la vraisemblance du modèle vérifie d'après l'équation *(1)* (page **??**) :

$$\begin{aligned} L_W &\propto \prod_{i=1}^{N} \prod_{k=1}^{C} \left[p_k(W, X_i) \right]^{\mathbb{P}(Y_i = k)} \\ \ln L_W &\propto \sum_{i=1}^{N} \sum_{k=1}^{C} \eta_i^k \ln \left[p_k(W, X_i) \right] \end{aligned}$$

La solution du problème $\overset{*}{W} = \underset{W \in \mathbb{R}^l}{\arg\max}\, L_W$ est celle d'un problème d'optimisation sous contrainte. Afin de contourner ce problème, on définit la fonction f :

$$\begin{array}{l} f : \mathbb{R}^M \times \mathbb{R}^p \longrightarrow \mathbb{R}^C \\ \forall (W,x) \in \mathbb{R}^M \times \mathbb{R}^p,\ f(W,x) = \left(f_1(W,x) \right), \dots, f_C(W,x) \\ \text{et } \forall i \in \{1,\cdots,N\},\ \forall k \in \{1,\cdots,C\},\ p^k(W,X_i) = \dfrac{e^{f_k(W,X_i)}}{\sum_{l=1}^{C} e^{f_l(W,X_i)}} \end{array}$$

Les contraintes sur $\left(p^k(W,X_i)\right)$ sont bien vérifiées :

$$\begin{array}{l} \forall i \in \{1,\cdots,N\},\ \forall k \in \{1,\cdots,C\},\ p^k(W,X_i) \geqslant 0 \\ \forall i \in \{1,\cdots,N\},\ \sum_{k=1}^{C} p^k(W,X_i) = 1 \end{array}$$

On en déduit que :

$$\begin{aligned} \ln L_W &\propto \sum_{i=1}^{N} \sum_{k=1}^{C} \eta_i^k \left[f_k(W,X_i) - \ln \left[\sum_{l=1}^{C} e^{f_l(W,X_i)} \right] \right] \\ \ln L_W &\propto \sum_{i=1}^{N} \sum_{k=1}^{C} \eta_i^k f_k(W,X_i) - \sum_{i=1}^{N} \ln \left[\sum_{l=1}^{C} e^{f_l(W,X_i)} \right] \underbrace{\sum_{k=1}^{C} \eta_i^k}_{=1} \end{aligned}$$

D'où :

$$\ln L_W \propto \sum_{i=1}^{N} \sum_{k=1}^{C} \eta_i^k f_k(W,X_i) - \sum_{i=1}^{N} \ln \left[\sum_{l=1}^{C} e^{f_l(W,X_i)} \right]$$

Ceci mène à la définition du problème de classification suivant :

Problème P2 : classification

Soit A l'échantillon suivant :

$$A = \left\{ \left(X_i, y_i = \left(\eta_i^k\right)_{1 \leqslant k \leqslant C} \right) \in \mathbb{R}^p \times \mathbb{R}^C \text{ tel que } \sum_{k=1}^{c} \eta_i^k = 1 \middle| 1 \leqslant i \leqslant N \right\}$$

y_i^k représente la probabilité que l'élément X_i appartiennent à la classe k : $\eta_i^k = \mathbb{P}(Y_i = k|X_i)$

Le classifieur cherché est une fonction f définie par :

$$\begin{array}{rcl} f : \mathbb{R}^M \times \mathbb{R}^p & \longrightarrow & \mathbb{R}^C \\ (W, X) & \longrightarrow & (f_1(W, X), \ldots, f_p(W, X)) \end{array}$$

Dont le vecteur de poids W^* est égal à :

$$W^* = \arg\max_W \sum_{i=1}^{N} \sum_{k=1}^{C} \eta_i^k f_k(W, X_i) - \sum_{i=1}^{N} \ln \left[\sum_{l=1}^{C} e^{f_l(W, X_i)} \right]$$

Réseau de neurones adéquat

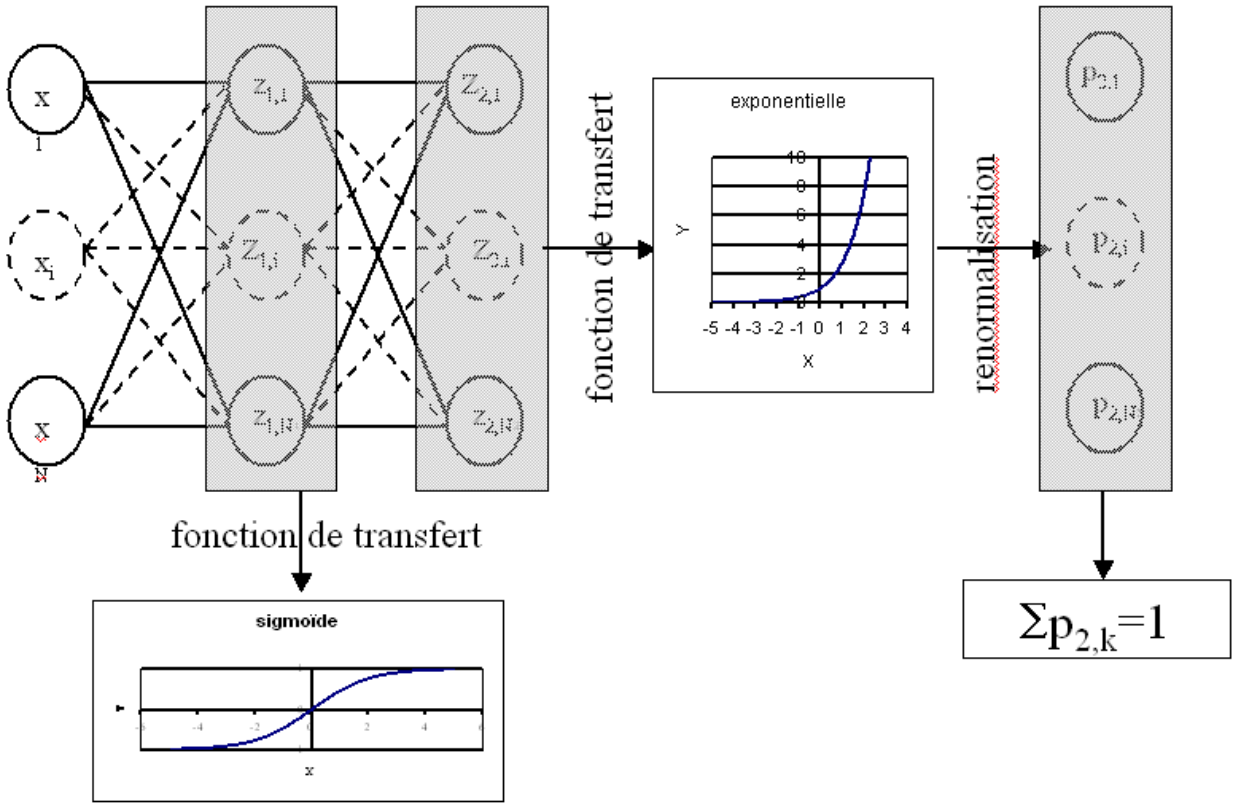
Dans le problème précédent, la maximisation de $\overset{*}{W} = \arg\max_{W \in \mathbb{R}^M} L_W$ aboutit au choix d'une fonction :

$$X \in \mathbb{R}^p \longrightarrow f(\overset{*}{W}, X) \in \mathbb{R}^C$$

Le réseau de neurones *suivant* (page 54) $g : (W, X) \in \mathbb{R}^M \times \mathbb{R}^p \longrightarrow \mathbb{R}^C$ choisi pour modéliser f aura pour sorties :

$$\begin{array}{l} X \in \mathbb{R}^p \longrightarrow g(\overset{*}{W}, X) \in \mathbb{R}^C \\ \forall k \in \{1, \cdots, C\},\ g_k(W, X) = e^{f_k(W, X)} \end{array}$$

Figure F1 : Réseau de neurones adéquat pour la classification



On en déduit que la fonction de transert des neurones de la couche de sortie est : $x \longrightarrow e^x$. La probabilité pour le vecteur $x \in \mathbb{R}^p$ d'appartenir à la classe $k \in \{1, \cdots, C\}$ est $p_k(\overset{*}{W}, x) = \mathbb{P}(Y = k|x) = \dfrac{g_k(\overset{*}{W}, x)}{\sum_{l=1}^{C} g_l(\overset{*}{W}, x)}$. La fonction d'erreur à minimiser est l'opposé de la log-vraisemblance du modèle :

$$\begin{aligned}
\overset{*}{W} &= \underset{W \in \mathbb{R}^M}{\arg\min} \left[\sum_{i=1}^{N} \left(- \sum_{k=1}^{C} \eta_i^k \ln \left(g_k \left(W, X_i \right) \right) + \ln \left[\sum_{l=1}^{C} g_l \left(W, X_i \right) \right] \right) \right] \\
&= \underset{W \in \mathbb{R}^M}{\arg\min} \left[\sum_{i=1}^{N} h \left(W, X_i, \eta_i^k \right) \right]
\end{aligned}$$

On note C_{rn} le nombre de couches du réseau de neurones, $z^k_{C_{rn}}$ est la sortie k avec $k \in \{1, \cdots, C\}$, $g_k(W, x) = z^k_{C_{rn}} = e^{y^k_{C_{rn}}}$ où $y^k_{C_{rn}}$ est le potentiel du neurone k de la couche de sortie.
On calcule :

$$\begin{aligned}
\frac{\partial h \left(W, X_i, y_i^k \right)}{\partial y^k_{C_{rn}}} &= -\eta_i^k + \frac{z^i_{Crn}}{\sum_{m=1}^{C} z^m_{Crn}} \\
&= p_k(\overset{*}{W}, x) - \eta_i^k
\end{aligned}$$

Cette équation permet d'adapter l'algorithme de la *rétropropagation* (page 44) décrivant rétropropagation pour le problème de la classification et pour un exemple $\left(X, y = \left(\eta^k \right)_{1 \leqslant k \leqslant C} \right)$. Seule la couche de sortie change.

Algorithme A1 : rétropropagation

Cet algorithme de rétropropagation est l'adaptation de *rétropropagation* (page 44) pour le problème de la classification. Il suppose que l'algorithme de *propagation* (page 27) a été préalablement exécuté. On note $y'_{c,i} = \frac{\partial e}{\partial y_{c,i}}$, $w'_{c,i,j} = \frac{\partial e}{\partial w_{c,i,j}}$ et $b'_{c,i} = \frac{\partial e}{\partial b_{c,i}}$.

Initialiasation

for i in $1..C_C$

$$y'_{C,i} \longleftarrow \frac{z_{C,i}}{\sum_{l=1}^{C} z_{C,l}} - \eta_i$$

Récurrence, Terminaison

Voir *rétropropagation* (page 44).

On vérifie que le gradient s'annule lorsque le réseau de neurones retourne pour l'exemple (X_i, y_i) la distribution de $Y|X_i \sim \mathcal{M}(y_i)$. Cet algorithme de rétropropagation utilise un vecteur désiré de probabilités $(\eta_1, \ldots, \eta_{C_C})$ vérifiant $\sum_{i=1}^{C_C} \eta_i = 1$. L'expérience montre qu'il est préférable d'utiliser un vecteur vérifiant la contrainte :

$$\begin{aligned}
&\forall i \in \{1, \ldots, C_C\}, \ \min \{\eta_i, 1 - \eta_i\} > \alpha \\
&\text{avec } \alpha > 0
\end{aligned}$$

Généralement, α est de l'ordre de $0,1$ ou $0,01$. Cette contrainte facilite le calcul de la vraisemblance et évite l'obtention de gradients quasi-nuls qui freinent l'apprentissage lorsque les fonctions exponnetielles sont saturées (voir *[Bishop1995]* (page **??**)).

3.1.8 Prolongements

Base d'apprentissage et base de test

Les deux exemples de régression et de classification *La régression* (page 28) et *Problème de classification pour les réseaux de neurones* (page 53) ont montré que la structure du réseau de neurones la mieux adaptée a une grande importance. Dans ces deux cas, une rapide vérification visuelle permet de juger de la qualité du modèle obtenu après apprentissage, mais bien souvent, cette ńăvisionă" est inaccessible pour des dimensions supérieures à deux. Le meilleur moyen de jauger le modèle appris est de vérifier si l'erreur obtenue sur une base ayant servi à l'apprentissage (ou *base d'apprentissage*) est conservée sur une autre base (ou *base de test*) que le modèle découvre pour la première fois.

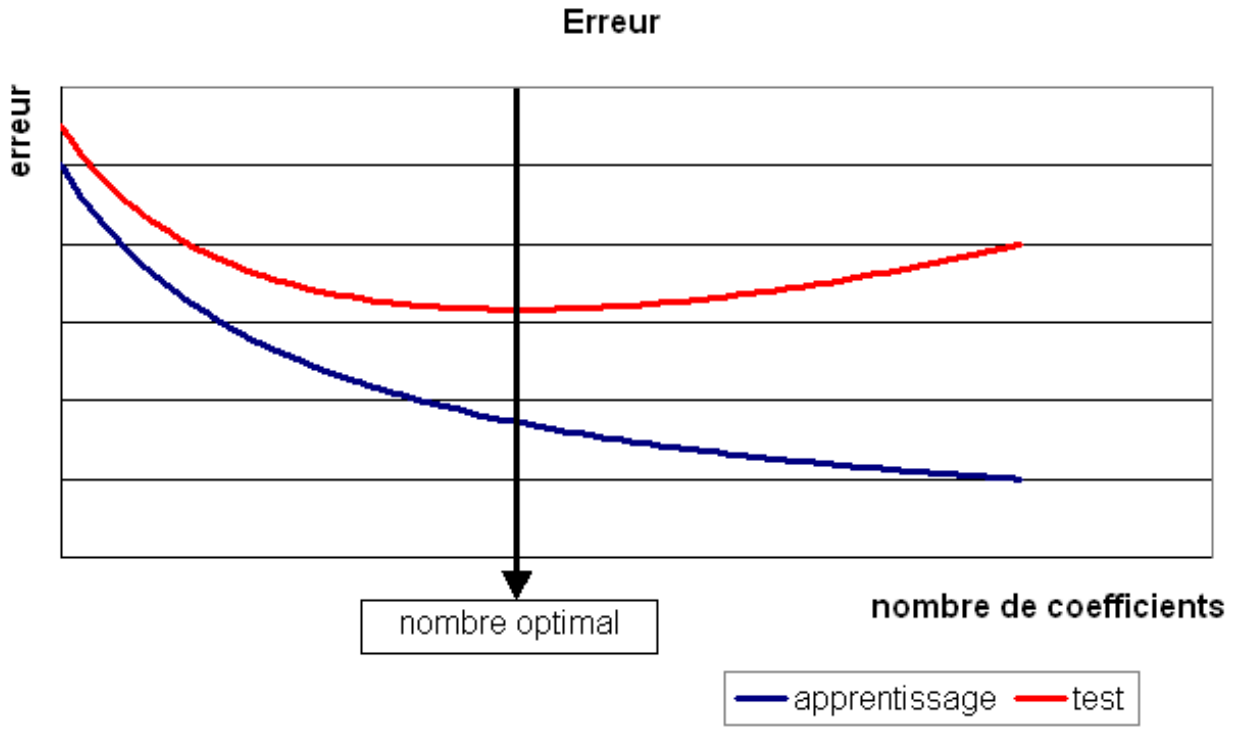
Soit $B = \{(X_i, Y_i) \,|1 \leqslant i \leqslant N\}$ l'ensemble des observations disponibles. Cet ensemble est aléatoirement scindé en deux sous-ensembles B_a et B_t de telle sorte que :

$$
\begin{array}{l}
B_a \neq \emptyset \text{ et } B_t \neq \emptyset \\
B_a \cup B_t = B \text{ et } B_a \cap B_t = \emptyset \\
\frac{\#B_a}{\#B_a \cup B_t} = p \in]0,1[, \text{ en règle générale, } p \in \left[\frac{1}{2}, \frac{3}{4}\right]
\end{array}
$$

Ce découpage est valide si tous les exemples de la base B obéissent à la même loi, les deux bases B_a et B_t sont dites *homogènes*. Le réseau de neurones sera donc appris sur la base d'apprentissage B_a et ńătestéă" sur la base de test B_t. Le test consiste à vérifier que l'erreur sur B_t est sensiblement égale à celle sur B_a, auquel cas on dit que le modèle (ou réseau de neurones) généralise bien. Le modèle trouvé n'est pas pour autant le bon modèle mais il est robuste. La courbe figure suivante illustre une définition du modèle optimal comme étant celui qui minimise l'erreur sur la base de test. Lorsque le modèle choisi n'est pas celui-là, deux cas sont possibles :

— Le nombre de coefficients est trop petit : le modèle généralise bien mais il existe d'autres modèles meilleurs pour lesquels l'erreur d'apprentissage et de test est moindre.
— Le nombre de coefficients est trop grand : le modèle généralise mal, l'erreur d'apprentissage est faible et l'erreur de test élevée, le réseau a appris la base d'apprentissage par coeur.

Figure F1 : Modèle optimal pour la base de test



Ce découpage des données en deux bases d'apprentissage et de test est fréquemment utilisé pour toute estimation de modèles résultant d'une optimisation réalisée au moyen d'un algorithme itératif. C'est le cas par exemple des modèles de Markov cachés. Elle permet de s'assurer qu'un modèle s'adapte bien à de nouvelles données.

Fonction de transfert à base radiale

La fonction de transfert est dans ce cas à base radiale (souvent abrégée par RBF pour radial basis function [34]. Elle ne s'applique pas au produit scalaire entre le vecteur des poids et celui des entrées mais à la distance euclidienne entre ces vecteurs.

Définition D1 : neurone distance

Un neurone distance à p entrées est une fonction $f : \mathbb{R}^{p+1} \times \mathbb{R}^p \longrightarrow \mathbb{R}$ définie par :

- $g : \mathbb{R} \rightarrow \mathbb{R}$
- $W \in \mathbb{R}^{p+1}, W = (w_1, \ldots, w_{p+1}) = (W', w_{p+1})$
- $\forall x \in \mathbb{R}^p,\ f(W, x) = e^{-\|W'-x\|^2 + w_{p+1}}$ avec $x = (x_1, \ldots, x_p)$

Ce neurone est un cas particulier du suivant qui pondère chaque dimension par un coefficient. Toutefois, ce neurone possède $2p + 1$ coefficients où p est le nombre d'entrée.

Définition D2 : neurone distance pondérée

Pour un vecteur donné $W \in \mathbb{R}^p = (w_1, \ldots, w_p)$, on note $W_i^j = (w_i, \ldots, w_j)$. Un neurone distance pondérée à p entrées est une fonction $f : \mathbb{R}^{2p+1} \times \mathbb{R}^p \longrightarrow \mathbb{R}$ définie par :

- $g : \mathbb{R} \rightarrow \mathbb{R}$
- $W \in \mathbb{R}^{2p+1}, W = (w_1, \ldots, w_{2p+1}) = (w_1, w_{2p+1})$

34. https://en.wikipedia.org/wiki/Radial_basis_function

— $\forall x \in \mathbb{R}^p,\ f(W,x) = \exp\left[-\left[\sum_{i=1}^{p} w_{p+i}(w_i - x_i)^2\right] + w_{p+1}\right]$ avec $x = (x_1, \ldots, x_p)$

La fonction de transfert est $x \longrightarrow e^x$ est le potentiel de ce neurone donc : $y = -\left[\sum_{i=1}^{p} w_{p+i}(w_i - x_i)^2\right] + w_{p+1}$.

L'algorithme de *rétropropagation* (page 44) est modifié par l'insertion d'un tel neurone dans un réseau ainsi que la rétropropagation. Le plus simple tout d'abord :

$$\begin{aligned} 1 \leqslant i \leqslant p, \quad \frac{\partial y}{\partial w_i} &= -2w_{p+i}(w_i - x_i) \\ p+1 \leqslant i \leqslant 2p, \quad \frac{\partial y}{\partial w_i} &= -(w_i - x_i)^2 \\ i = 2p+1, \quad \frac{\partial y}{\partial w_i} &= -1 \end{aligned}$$

Pour le neurone distance simple, la ligne *(1)* (page **??**) est superflue, tous les coefficients $(w_i)_{p+1 \leqslant i \leqslant 2p}$ sont égaux à 1. La relation *(6)* (page **??**) reste vraie mais n'aboutit plus à :eq :*algo_retro_5*, celle-ci devient en supposant que la couche d'indice $c+1$ ne contient que des neurones définie par la définition précédente.

$$\begin{aligned} \frac{\partial e}{\partial y_{c,i}} &= \sum_{l=1}^{C_{c+1}} \frac{\partial e}{\partial y_{c+1,l}} \frac{\partial y_{c+1,l}}{\partial z_{c,i}} \frac{\partial z_{c,i}}{\partial y_{c,i}} \\ &= \left[\sum_{l=1}^{C_{c+1}} \frac{\partial e}{\partial y_{c+1,l}} \left(2w_{c+1,l,p+i}(w_{c+1,l,i} - z_{c,i})\right)\right] \frac{\partial z_{c,i}}{\partial y_{c,i}} \end{aligned}$$

Poids partagés

Les poids partagés sont simplement un ensemble de poids qui sont contraints à conserver la même valeur. Soit G un groupe de poids partagés dont la valeur est w_G. Soit X_k et Y_k un exemple de la base d'apprentissage (entrées et sorties désirées), l'erreur commise par le réseau de neurones est $e(W, X_k, Y_k)$.

$$\frac{\partial e(W, X_k, Y_k)}{\partial w_G} = \sum_{w \in G} \frac{\partial e(W, X_k, Y_k)}{\partial w_G} \frac{\partial w_G}{\partial w} = \sum_{w \in G} \sum \frac{\partial e(W, X_k, Y_k)}{\partial w_G}$$

Par conséquent, si un poids w appartient à un groupe G de poids partagés, sa valeur à l'itération suivante sera :

$$w_{t+1} = w_t - \varepsilon_t \left(\sum_{w \in G} \frac{\partial e(W, X_k, Y_k)}{\partial w}\right)$$

Cette idée est utilisée dans les réseaux neuronaux convolutifs [35] (deep learning [36], CS231n Convolutional Neural Networks for Visual Recognition [37]).

Dérivée par rapport aux entrées

On note (X_k, Y_k) un exemple de la base d'apprentissage. Le réseau de neurones est composé de C couches, C_i est le nombre de neurones sur la ième couche, C_0 est le nombre d'entrées. Les entrées sont appelées $(z_{0,i})_{1 \leqslant i \leqslant C_0}$, $(y_{1,i})_{1 \leqslant i \leqslant C_1}$ sont les potentiels des neurones de la première couche, on en déduit que, dans le cas d'un neurone classique (non distance) :

$$\frac{\partial e(W, X_k, Y_k)}{\partial z_{0,i}} = \sum_{j=1}^{C_1} \frac{\partial e(W, X_k, Y_k)}{\partial y_{1,j}} \frac{\partial y_{1,j}}{\partial z_{0,i}} = \sum_{j=1}^{C_1} \frac{\partial e(W, X_k, Y_k)}{\partial y_{1,j}} w_{1,j,i}$$

35. https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif
36. https://fr.wikipedia.org/wiki/Apprentissage_profond
37. http://cs231n.github.io/neural-networks-1/#layers

Comme le potentiel d'un neurone distance n'est pas linéaire par rapport aux entrées $\left(y = \sum_{i=1}^{N} (w_i - z_{0,i})^2 + b\right)$, la formule devient dans ce cas :

$$\frac{\partial e(W, X_k, Y_k)}{\partial z_{0,i}} = \sum_{j=1}^{C_1} \frac{\partial e(W, X_k, Y_k)}{\partial y_{1,j}} \frac{\partial y_{1,j}}{\partial z_{0,i}} = -2 \sum_{j=1}^{C_1} \frac{\partial e(W, X_k, Y_k)}{\partial y_{1,j}} (w_{1,j,i} - z_{0,i})$$

Régularisation ou Decay

Lors de l'apprentissage, comme les fonctions de seuil du réseau de neurones sont bornées, pour une grande variation des coefficients, la sortie varie peu. De plus, pour ces grandes valeurs, la dérivée est quasi nulle et l'apprentissage s'en trouve ralenti. Par conséquent, il est préférable d'éviter ce cas et c'est pourquoi un terme de régularisation est ajouté lors de la mise à jour des coefficients (voir *[Bishop1995]* (page **??**)). L'idée consiste à ajouter à l'erreur une pénalité fonction des coefficients du réseau de neurones : $E_{reg} = E + \lambda \sum_i w_i^2$.

Et lors de la mise à jour du poids w_i^t à l'itération $t+1$: $w_i^{t+1} = w_i^t - \epsilon_t \left[\frac{\partial E}{\partial w_i} - 2\lambda w_i^t\right]$.

Le coefficient λ peut décroître avec le nombre d'itérations et est en général de l'ordre de $0,01$ pour un apprentissage avec gradient global, plus faible pour un apprentissage avec gradient stochastique.

Problèmes de gradients

La descente du gradient repose sur l'algorithme de *rétropropagation* (page 44) qui propoge l'erreur depuis la dernière couche jusqu'à la première. Pour peu qu'une fonction de seuil soit saturée. Hors la zone rouge, le gradient est très atténué.

Après deux couches de fonctions de transferts, le gradient est souvent diminué. On appelle ce phénomène le Vanishing gradient problem [38]. C'est d'autant plus probable que le réseau est gros. Quelques pistes pour y remédier : Recurrent Neural Networks Tutorial, Part 3 – Backpropagation Through Time and Vanishing Gradients [39], Why are deep neural networks hard to train ? [40]. L'article Deep Residual Learning for Image Recognition [41] présente une structure de réseau qui va dnas le même sens. De la même manière, la norme du gradient peut exploser plus particulièrement dans le cas des réseaux de neurones récurrents [42] : Understanding the exploding gradient problem [43].

Sélection de connexions

Ce paragraphe présente un algorithme de sélection de l'architecture d'un réseau de neurones proposé par Cottrel et Al. dans *[Cottrel1995]* (page **??**). La méthode est applicable à tout réseau de neurones mais n'a été démontrée que pour la classe de réseau de neurones utilisée pour la *régression* (page 28). Les propriétés qui suivent ne sont vraies que des réseaux à une couche cachée et dont les sorties sont linéaires. Soit (X_k, Y_k) un exemple de la base d'apprentissage, les résidus de la régression sont supposés normaux et i.i.d. L'erreur est donc (voir *Formulation du problème de la régression* (page 34)) : $e(W, X_k, Y_k) = (f(W, X_k) - Y_k)^2$.

On peut estimer la loi asymptotique des coefficients du réseau de neurones. Des connexions ayant un rôle peu important peuvent alors être supprimées sans nuire à l'apprentissage en testant la nullité du coefficient associé. On note $\widehat{W}$ les poids trouvés par apprentissage et $\overset{*}{W}$ les poids optimaux. On définit :

$$\begin{aligned} \text{la suite } \widehat{\varepsilon_k} &= f\left(\widehat{W}, X_k\right) - Y_k,\ \widehat{\sigma}_N^2 = \frac{1}{N} \sum_{k=1}^{N} \widehat{\varepsilon_k}^2 \\ \text{la matrice } \widehat{\Sigma_N} &= \frac{1}{N} \left[\nabla_{\widehat{W}} e(W, X_k, Y_k)\right] \left[\nabla_{\widehat{W}} e(W, X_k, Y_k)\right]' \end{aligned}$$

38. https://en.wikipedia.org/wiki/Vanishing_gradient_problem
39. http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/
40. http://neuralnetworksanddeeplearning.com/chap5.html
41. http://arxiv.org/pdf/1512.03385v1.pdf
42. https://en.wikipedia.org/wiki/Recurrent_neural_network
43. http://arxiv.org/pdf/1211.5063v1.pdf

Théorème T1 : loi asymptotique des coefficients

Soit f un réseau de neurone défini par *perceptron* (page 26) composé de :

— une couche d'entrées
— une couche cachée dont les fonctions de transfert sont sigmoïdes
— une couche de sortie dont les fonctions de transfert sont linéaires

Ce réseau sert de modèle pour la fonction f dans le problème de *régression* (page 28) avec un échantillon $((X_1, Y_1), \ldots, (X_N, Y_N))$, les résidus sont supposés normaux. La suite $(\widehat{\epsilon_k})$ définie par *(2)* (page **??**) vérifie :

$$\frac{1}{N}\sum_{i=1}^{N}\widehat{\epsilon_k} = 0 = \mathbb{E}\left[f\left(\widehat{W}, X\right) - Y\right]$$

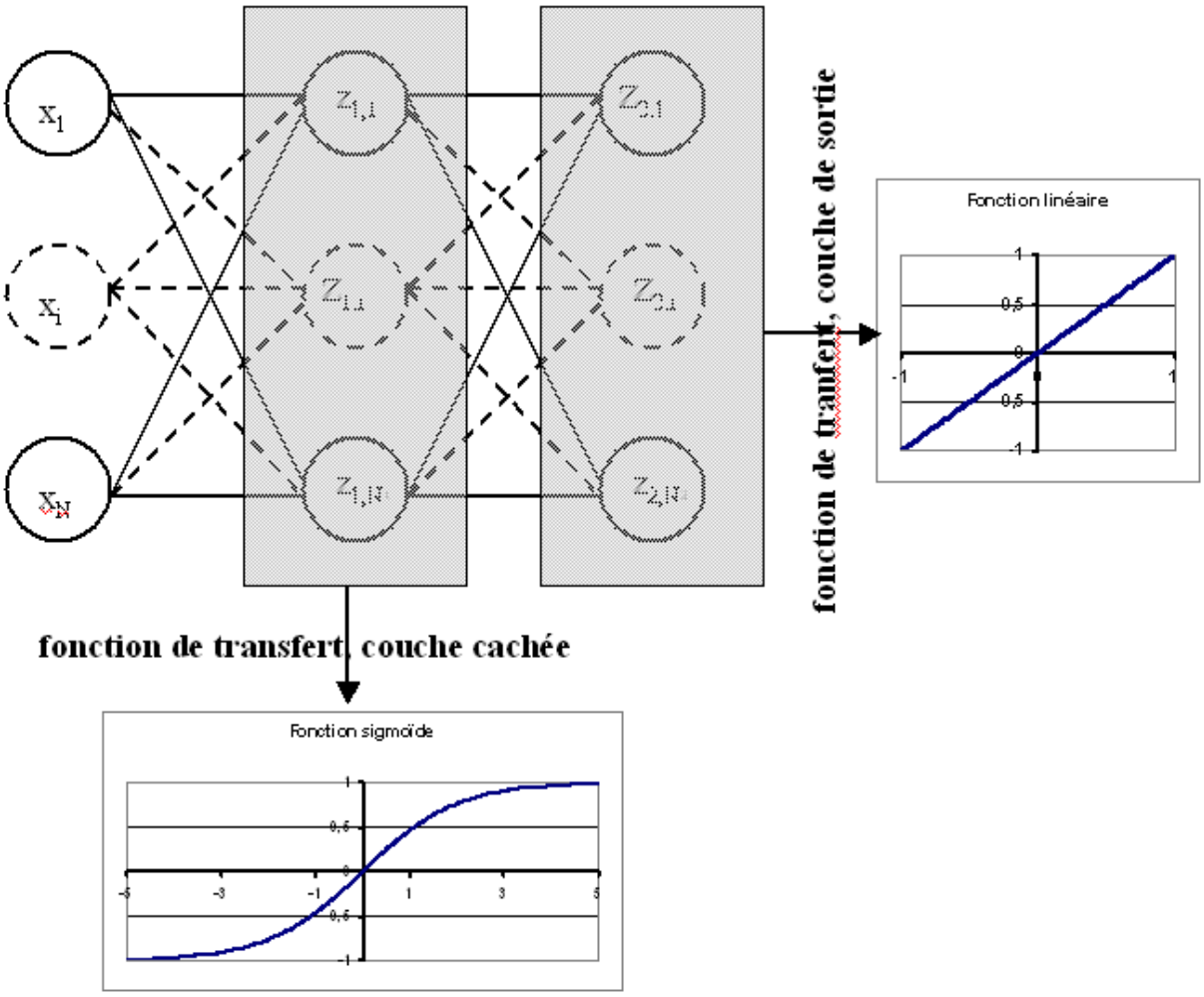
Et le vecteur aléatoire $\widehat{W} - W^*$ vérifie :

$$\sqrt{N}\left[\widehat{W} - W^*\right] \overset{T\to+\infty}{\longrightarrow} \mathcal{N}\left(0, \widehat{\sigma_N}^2\widehat{\Sigma_N}\right)$$

Où la matrice $\widehat{\Sigma_N}$ est définie par *(2)* (page **??**).

end{xtheorem}

Figure F2 : Réseau de neurones pour lequel la sélection de connexions s'applique



La démonstration de ce théorème est donnée par l'article *[Cottrel1995]* (page **??**). Ce théorème mène au corollaire suivant :

Corollaire C1 : nullité d'un coefficient

Les notations utilisées sont celles du théorème sur *loi asymptotique des coefficients* (page 60). Soit w_k un poids du réseau de neurones d'indice quelconque k. Sa valeur estimée est $\widehat{w_k}$, sa valeur optimale w_k^*. D'après le théorème :

$$N \frac{\left(\widehat{w_k} - w_k^*\right)^2}{\widehat{\sigma_N}^2 \left(\widehat{\Sigma_N}^{-1}\right)_{kk}} \overset{T \to +\infty}{\longrightarrow} \chi_1^2$$

Ce résultat permet, à partir d'un réseau de neurones, de supprimer les connexions pour lesquelles l'hypothèse de nullité n'est pas réfutée. Afin d'aboutir à l'architecture minimale adaptée au problème, Cottrel et Al. proposent dans *[Cottrel1995]* (page **??**) l'algorithme suivant :

Théorème T2 : sélection d'architecture

Les notations utilisées sont celles du théorème *loi asymptotique des coefficients* (page 60). f est un réseau de neurones de paramètres W. On définit la constante τ, en général $\tau = 3,84$ puisque $\mathbb{P}(X < \tau) = 0,95$ si $X \sim \chi_1^2$.

Initialisation

Une architecture est choisie pour le réseau de neurones f incluant un nombre *M* de paramètres.

Apprentissage

Le réseau de neurones f est appris. On calcule les nombre et matrice $\widehat{\sigma_N}^2$ et $\widehat{\Sigma_N}$. La base d'apprentissage contient N exemples.

Test

for k in $1..M$

$$t_k \longleftarrow N \frac{\widehat{w_k}^2}{\widehat{\sigma_N}^2 \left(\widehat{\Sigma_N}^{-1}\right)_{kk}}$$

Sélection

$k' \longleftarrow \arg\min_k t_k$

si $t_{k'} < \tau$

Le modèle obtenu est supposé être le modèle optimal. L'algorithme s'arrête.

sinon

La connexion k' est supprimée ou le poids $w_{k'}$ est maintenue à zéro.

$M \longleftarrow M - 1$

Retour à l'apprentissage.

Cet algorithme est sensible au minimum local trouvé lors de l'apprentissage, il est préférable d'utiliser des méthodes du second ordre afin d'assurer une meilleure convergence du réseau de neurones.

L'étape de sélection ne supprime qu'une seule connexion. Comme l'apprentissage est coûteux en calcul, il peut être intéressant de supprimer toutes les connexions k qui vérifient $t_k < \tau$. Il est toutefois conseillé de ne pas enlever trop de connexions simultanément puisque la suppression d'une connexion nulle peut réhausser le test d'une autre connexion, nulle à cette même itération, mais non nulle à l'itération suivante. Dans l'article *[Cottrel1995]* (page **??**), les auteurs valident leur algorithme dans le cas d'une régression grâce à l'algorithme suivant.

Algorithme A1 : validation de l'algorithme de sélection des coefficients

Choix aléatoire d'un modèle

Un réseau de neurones est choisi aléatoirement, soit $f : \mathbb{R}^p \to \mathbb{R}$ la fonction qu'il représente. Une base d'apprentissage A (ou échantillon) de N observations est générée aléatoirement à partir de ce modèle :

$$\begin{array}{l} \text{soit } (\epsilon_i)_{1 \leqslant i \leqslant N} \text{ un bruit blanc} \\ A = \left\{ (X_i, Y_i)_{1 \leqslant i \leqslant N} \middle| \forall i \in \{1, \cdots, N\},\ Y_i = f(X_i) + \epsilon_i \right\} \end{array}$$

Choix aléatoire d'un modèle

L'algorithme de *sélection* (page 61) à un réseau de neurones plus riche que le modèle choisi dans l'étape d'initilisation. Le modèle sélectionné est noté g.

Validation

Si $\|f - g\| \approx 0$,

l'algorithme de *sélection* (page 61) est validé.

3.1.9 Analyse en composantes principales (ACP) et Auto Encoders

Cet algorithme est proposé dans *[Song1997]* (page **??**). Autrefois réseau diabolo, le terme auto-encoder [44] est plus utilisé depuis l'avénement du deep learning. Il s'agit de compresser avec perte un ensemble de points. L¿ACP [45] est une forme de compression linéaire puisqu'on cherche à préserver l'information en projetant un nuage de points de façon à maximiser l'inertie du nuage. Les auto-encoders fonctionnent sur le même principe avec des modèles non linéaires.

subsection{Principe}

L'algorithme implémentant l'analyse en composantes principales est basé sur un réseau linéaire dit *ńădiaboloă"*, ce réseau possède une couche d'entrées à N entrées, une couche cachée et une couche de sortie à N sorties. L'objectif est d'apprendre la fonction identité sur l'espace $\mathbb{R}^N$. Ce ne sont plus les sorties qui nous intéressent mais la couche cachée intermédiaire qui effectue une compression ou projection des vecteurs d'entrées puisque les entrées et les sorties du réseau auront pour but d'être identiques.

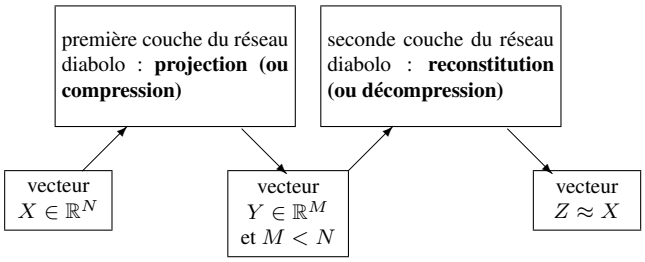


Figure F1 : Principe de la compression par un réseau diabolo

La figure suivante illustre un exemple de compression de vecteur de $\mathbb{R}^3$ dans $\mathbb{R}^2$.

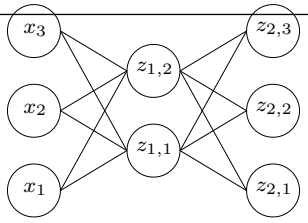


Figure F2 : Réseau diabolo : réduction d'une dimension Ce réseau possède 3 entrées et 3 sorties Minimiser l'erreur $\sum_{k=1}^{N} E(X_k, X_k)$ revient à compresser un vecteur de dimension 3 en un vecteur de dimension 2. Les coefficients de la première couche du réseau de neurones permettent de compresser les données. Les coefficients de la seconde couche permettent de les décompresser.

La compression et décompression ne sont pas inverses l'une de l'autre, à moins que l'erreur *(1)* (page **??**) soit nulle. La décompression s'effectue donc avec des pertes d'information. L'enjeu de l'ACP est de trouver un bon compromis entre le nombre de coefficients et la perte d'information tôlérée. Dans le cas de l'ACP, la compression est *ńălinéaireă"*, c'est une projection.

Problème de l'analyse en composantes principales

L'analyse en composantes principales ou ACP est définie de la manière suivante :

44. https://en.wikipedia.org/wiki/Autoencoder
45. https://fr.wikipedia.org/wiki/Analyse_en_composantes_principales

Problème P1 : analyse en composantes principales (ACP)

Soit $(X_i)_{1 \leqslant i \leqslant N}$ avec $\forall i \in \{1, \ldots, N\}$, $X_i \in \mathbb{R}^p$. Soit $W \in M_{p,d}(\mathbb{R})$, $W = (C_1, \ldots, C_d)$ où les vecteurs (C_i) sont les colonnes de W et $d < p$. On suppose également que les (C_i) forment une base othonormée. Par conséquent :

$$W'W = I_d$$

$(W'X_i)_{1 \leqslant i \leqslant N}$ est l'ensemble des vecteurs (X_i) projetés sur le sous-espace vectoriel engendré par les vecteurs (C_i). Réaliser une analyse en composantes principales, c'est trouver le meilleur plan de projection pour les vecteurs (X_i), celui qui maximise l'inertie de ce nuage de points, c'est donc trouver W^* tel que :

$$W^* = \underset{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}}{\arg\max} E(W) = \underset{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}}{\arg\max} \left[\sum_{i=1}^{N} \|W'X_i\|^2 \right]$$

Le terme $E(W)$ est l'inertie du nuage de points (X_i) projeté sur le sous-espace vectoriel défini par les vecteurs colonnes de la matrice W.

Résolution d'une ACP avec un réseau de neurones diabolo

Un théorème est nécessaire avant de construire le réseau de neurones menant à la résolution du problème de l¿*ACP* (page 64) afin de passer d'une optimisation sous contrainte à une optimisation sans contrainte.

Théorème T1 : résolution de l'ACP

Les notations utilisées sont celles du problème de l¿*ACP* (page 64). Dans ce cas :

$$S = \underset{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}}{\arg\max} \left[\sum_{i=1}^{N} \|W'X_i\|^2 \right] = \underset{W \in M_{p,d}(\mathbb{R})}{\arg\min} \left[\sum_{i=1}^{N} \|WW'X_i - X_i\|^2 \right]$$

De plus S est l'espace vectoriel engendré par les d vecteurs propres de la matrice $XX' = \sum_{i=1}^{N} X_i X_i'$ associées aux d valeurs propres de plus grand module.

Démonstration

Partie 1

L'objectif de cette partie est de chercher la valeur de :

$$\max_{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}} E(W)$$

Soit $X = (X_1, \ldots, X_N) \in (\mathbb{R}^p)^N$, alors :

$$E(W) = \sum_{i=1}^{N} \|W'X_i\|^2 = tr(X'WW'X) = tr(XX'WW')$$

La matrice XX' est symétrique, elle est donc diagonalisable et il existe une matrice $P \in M_p(\mathbb{R})$: $math$: telle qu :

$$\begin{array}{l} P'XX'P = D_X \text{ avec } D_X \text{ diagonale} \\ P'P = I_p \end{array} \tag{3.11}$$

Soit $P = (P_1, \ldots, P_p)$ les vecteurs propres de la matrice XX' associés aux valeurs propres $(\lambda_1, \ldots, \lambda_p)$ telles que $|\lambda_1| \geqslant ... \geqslant |\lambda_p|$. Pour mémoire, $W = (C_1, \ldots, C_d)$, et on a :

$$\forall i \in \{1, \ldots, p\}, \; XX'P_i = \lambda_i P_i$$
$$\forall i \in \{1, \ldots, d\}, \; C_i = P_i \Longrightarrow XX'WW' = D_{X,d} = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \ldots & 0 \\ 0 & 0 & \lambda_d \end{pmatrix}$$

D'où :

$$E(W) = tr(XX'WW') = tr(PD_XP'WW') = tr(D_XP'WW'P)$$

Donc :

$$\max_{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}} E(W) = \max_{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}} tr(D_XP'WW'P) = \max_{\substack{Y \in M_{p,d}(\mathbb{R}) \\ Y'Y = I_d}} tr(D_XYY') = \sum_{i=1} d\lambda_i$$

Partie 2

Soit $Y \in \max_{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}} tr(X'WW'X)$, $Y = (Y_1, \ldots, Y_d) = \left(y_i^k\right)_{\substack{1 \leqslant i \leqslant d \\ 1 \leqslant k \leqslant p}}$.

Chaque vecteur Y_i est écrit dans la base $(P_1, \ldots, P_p)$ définie en *(3)* (page **??**) :

$$\forall i \in \{1, \ldots, d\}, \; Y_i = \sum_{k=1}^{p} y_i^k P_p$$

Comme $Y'Y = I_d$, les vecteurs $(Y_1, \ldots, Y_d)$ sont orthogonaux deux à deux et normés, ils vérifient donc :

$$\begin{cases} \forall i \in \{1, \ldots, d\}, & \sum_{k=1}^{p} \left(y_i^k\right)^2 = 1 \\ \forall (i,j) \in \{1, \ldots, d\}^2, & \sum_{k=1}^{p} y_i^k y_j^k = 0 \end{cases}$$

De plus :

$$XX'YY' = XX' \left(\sum_{i=1}^{d} Y_i Y_i' \right) = \sum_{i=1}^{d} XX'Y_iY_i'$$

On en déduit que :

$$\begin{aligned} \forall i \in \{1, \ldots, d\}, \; XX'Y_iY_i' &= XX' \left(\sum_{k=1}^{p} y_i^k P_k \right) \left(\sum_{k=1}^{p} y_i^k P_k \right)' \\ &= \left(\sum_{k=1}^{p} \lambda_k y_i^k P_k \right) \left(\sum_{k=1}^{p} y_i^k P_k \right)' \end{aligned}$$

D'où :

$$\forall i \in \{1, \ldots, d\}, \; tr(XX'Y_iY_i') = \sum_{k=1}^{p} \lambda_k \left(y_i^k\right)^2$$

Et :

$$\begin{aligned} tr(XX'YY') &= \sum_{i=1}^{d} \sum_{k=1}^{p} \lambda_k \left(y_i^k\right)^2 \\ tr(XX'YY') &= \sum_{k=1}^{p} \lambda_k \left(\sum_{i=1}^{d} \left(y_i^k\right)^2 \right) = \sum_{k=1}^{p} \lambda_k \end{aligned}$$

Ceci permet d'affirmer que :

$$Y \in \max_{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}} tr\left(X'WW'X\right) \Longrightarrow vect\left(Y_1, \ldots, Y_d\right) = vect\left(P_1, \ldots, P_d\right)$$

Les équations *(4)* (page **??**) et *(5)* (page **??**) démontrent la seconde partie du théorème.

Partie 3

$$\begin{aligned}
\sum_{i=1}^{n} \left\|WW'X_i - X_i\right\|^2 &= \sum_{i=1}^{n} \left\|\left(WW' - I_N\right)X_i\right\|^2 \\
&= tr\left(X'\left(WW' - I_p\right)^2 X\right) \\
&= tr\left(XX'\left(\left(WW'\right)^2 - 2WW' + I_p\right)\right) \\
&= tr\left(XX'\left(WW'WW' - 2WW' + I_p\right)\right) \\
&= tr\left(XX'\left(-WW' + I_p\right)\right) \\
&= -tr\left(XX'WW'\right) + tr\left(XX'\right)
\end{aligned}$$

D'où :

$$\max_{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}} \left(\sum_{i=1}^{N} \left\|W'X_i\right\|^2\right) = \min_{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}} \left(\sum_{i=1}^{N} \left\|WW'X_i - X_i\right\|^2\right)$$

Partie 4

XX' est une matrice symétrique, elle est donc diagonalisable :

$$\exists P \in GL_N\left(\mathbb{R}\right) \text{ telle que } P'XX'P = D_p \text{ où } D_p \text{ est diagonale}$$

On en déduit que :

$$\begin{aligned}
\sum_{i=1}^{N} \left\|WW'X_i - X_i\right\|^2 &= tr\left(XX'\left(WW' - I_p\right)^2\right) \\
&= tr\left(PP'XX'PP'\left(WW' - I_p\right)^2\right) \\
&= tr\left(PD_pP'\left(WW' - I_p\right)^2\right) \\
&= tr\left(D_p\left(P'WW'P - I_p\right)^2\right) \\
&= tr\left(D_p\left(YY' - I_p\right)^2\right) \text{ avec } Y = P'W
\end{aligned}$$

D'où :

$$\arg\min_{Y} \left\{tr\left(D_p\left(YY' - I_p\right)^2\right)\right\} = \left\{Y \in M_{Nd}\left(\mathbb{R}\right) | YY' = I_d\right\}$$

Finalement, l'équation *(7)* (page **??**) permet de démontrer la première partie du théorème, à savoir *(2)* (page **??**) :

$$S = \arg\max_{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}} \left[\sum_{i=1}^{N} \left\|W'X_i\right\|^2\right] = \arg\min_{W \in M_{p,d}(\mathbb{R})} \left[\sum_{i=1}^{N} \left\|WW'X_i - X_i\right\|^2\right]$$

Calcul de valeurs propres et de vecteurs propres

Le calcul des valeurs propres et des vecteurs propres d'une matrice fait intervenir un réseau diabolo composé d'une seule couche cachée et d'une couche de sortie avec des fonctions de transfert linéaires. On note sous forme de matrice

(W) les coefficients de la seconde couche du réseau dont les biais sont nuls. On note d le nombre de neurones sur la couche cachée, et p le nombre d'entrées.

$$\forall i \in \{1,...,d\}\,,\; y_{1,i} = \sum_{j=1}^{p} w_{ji} x_j$$

Soit $X \in \mathbb{R}^p$ les entrées, $Y = (y_{1,1},...,y_{1,d}) \in \mathbb{R}^d$, on obtient que : $Y = W'X$.

Les poids de la seconde couche sont définis comme suit :

$$\forall\,(i,j) \in \{1,...,p\} \times \{1,...,d\}\; w_{2,j,i} = w_{1,i,j}$$

Par conséquent, le vecteur des sorties $Z \in \mathbb{R}^p$ du réseau ainsi construit est $Z = WW'X$. On veut minimiser l'erreur pour $(X_i)_{1 \leqslant i \leqslant N}$:

$$E = \sum_{i=1}^{N} \|WW'X_i - X_i\|^2$$

Il suffit d'apprendre le réseau de neurones pour obtenir :

$$W_d^* = \underset{W \in M_{pd}(\mathbb{R})}{\arg\max} \sum_{i=1}^{N} \|WW'X_i - X_i\|^2$$

D'après ce qui précède, l'espace engendré par les vecteurs colonnes de W est l'espace engendré par les k premiers vecteurs propres de la matrice $XX' = (X_1,...,X_P)\,(X_1,...,X_P)'$ associés aux k premières valeurs propres classées par ordre décroissant de module.

On en déduit que W_1^* est le vecteur propre de la matrice M associée à la valeur propre de plus grand module. W_2^* est l'espace engendré par les deux premiers vecteurs. Grâce à une orthonormalisation de Schmidt[46]. On en déduit à partir de W_1^* et W_2^*, les deux premiers vecteurs propres. Par récurrence, on trouve l'ensemble des vecteurs propres de la matrice XX'.

Définition D1 : orthonormalisation de Schmidt

L'orthonormalisation de Shmidt :

Soit $(e_i)_{1 \leqslant i \leqslant N}$ une base de $\mathbb{R}^p$

On définit la famille $(\varepsilon_i)_{1 \leqslant i \leqslant p}$ par :

$$\begin{aligned} \varepsilon_1 &= \frac{e_1}{\|e_1\|} \\ \forall i \in \{1,\cdots,p\}\,,\; \varepsilon_i &= \frac{e_i - \sum_{j=1}^{i-1} < e_i, \varepsilon_j > \varepsilon_j}{\left\| e_i - \sum_{j=1}^{i-1} < e_i, \varepsilon_j > \varepsilon_j \right\|} \end{aligned}$$

On vérifie que le dénominateur n'est jamais nul. $e_i - \sum_{j=1}^{i-1} < e_i, \varepsilon_j > \varepsilon_j \neq 0$ car $\forall k \in \{1,...,N\}\,,\; vect\,(e_1,...,e_k) = vect\,(\varepsilon_1,...,\varepsilon_k)$

Propriété P1 : base orthonormée

46. https://fr.wikipedia.org/wiki/Algorithme_de_Gram-Schmidt

La famille $(\varepsilon_i)_{1 \leqslant i \leqslant p}$ est une base orthonormée de $\mathbb{R}^p$.

L'algorithme qui permet de déterminer les vecteurs propres de la matrice XX' définie par le théorème de l¿*ACP* (page 64) est le suivant :

Algorithme A1 : vecteurs propres

Les notations utilisées sont celles du théorème de l¿*ACP* (page 64). On note V_d^* la matrice des d vecteurs propres de la matrice XX' associés aux d valeurs propres de plus grands module.

for d, p

Un réseau diabolo est construit avec les poids $W_d \in M_{p,d}(\mathbb{R})$ puis appris.

Le résultat de cet apprentissage sont les poids W_d^*.

if $d > 1$

L'orthonormalisation de Schmit permet de déduire V_d^* de V_{d-1}^* et W_d^*.

else

$V_d^* = W_d^*$

Analyse en Composantes Principales (ACP)

L'analyse en composantes principales permet d'analyser une liste d'individus décrits par des variables. Comme exemple, il suffit de prendre les informations extraites du recensement de la population française qui permet de décrire chaque habitant par des variables telles que la catégorie socio-professionnelle, la salaire ou le niveau d'étude. Soit $(X_1, ..., X_N)$ un ensemble de N individus décrits par p variables :

$$\forall i \in \{1, ..., N\}, \; X_i \in \mathbb{R}^p$$

L'ACP consiste à projeter ce nuage de point sur un plan qui conserve le maximum d'information. Par conséquent, il s'agit de résoudre le problème :

$$W^* = \underset{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}}{\arg\min} \left(\sum_{i=1}^{N} \left\| W' X_i \right\|^2 \right) \text{ avec } d < N$$

Ce problème a été résolu dans les paragraphes *Problème de l'analyse en composantes principales* (page 63) et *Calcul de valeurs propres et de vecteurs propres* (page 66), il suffit d'appliquer l'algorithme *vecteurs propres* (page 68).

Soit $(X_i)_{1 \leqslant i \leqslant N}$ avec $\forall i \in \{1, ..., N\}, \; X_i \in \mathbb{R}^p$. Soit $(P_1, \ldots, P_p)$ l'ensemble des vecteurs propres normés de la matrice XX' associés aux valeurs propres $(\lambda_1, \ldots, \lambda_p)$ classées par ordre décroissant de modules. On définit $\forall d \in \{1, \cdots, p\}, \; W_d = (P_1, \ldots, P_d) \in M_{p,d}$. On définit alors l'inertie I_d du nuage de points projeté sur l'espace vectoriel défini par P_d. On suppose que le nuage de points est centré, alors :

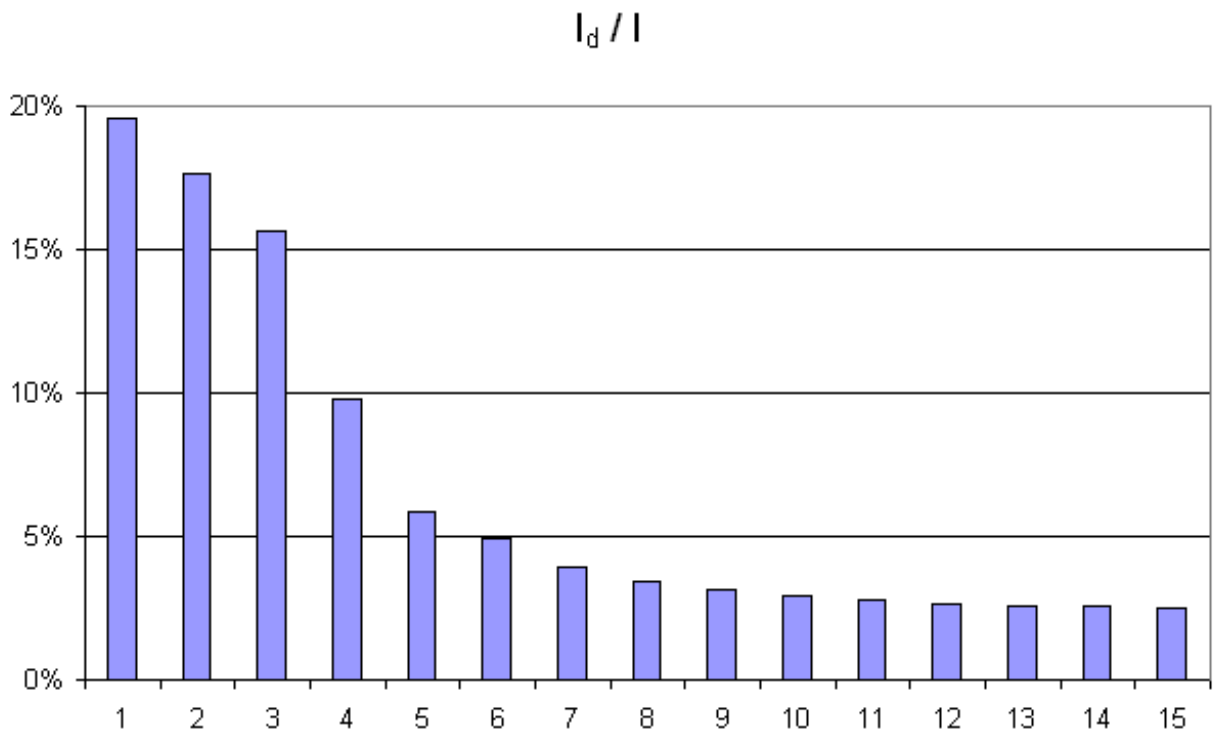
$$\forall d \in \{1, \cdots, p\}, \; I_d = \sum_{k=1}^{N} \left(P_d' X_k \right)^2 = tr\left(X' P_d P_d' X \right) = tr\left(XX' P_d P_d' \right) = \lambda_d$$

Comme $(P_1, \ldots, P_p)$ est une base orthonormée de $\mathbb{R}^p$, on en déduit que :

$$I = \sum_{k=1}^{P} X_k' X_k = \sum_{d=1}^{N} I_d = \sum_{d=1}^{p} \lambda_d$$

De manière empirique, on observe fréquemment que la courbe $(d, I_d)_{1 \leqslant d \leqslant p}$ montre un point d'inflexion (voir figure ci-dessous). Dans cet exemple, le point d'inflexion correspond à $d = 4$. En analyse des données, on considère empiriquement que seuls les quatres premières dimensions contiennent de l'information.

Figure F3 : Courbe d'inertie pour l'ACP

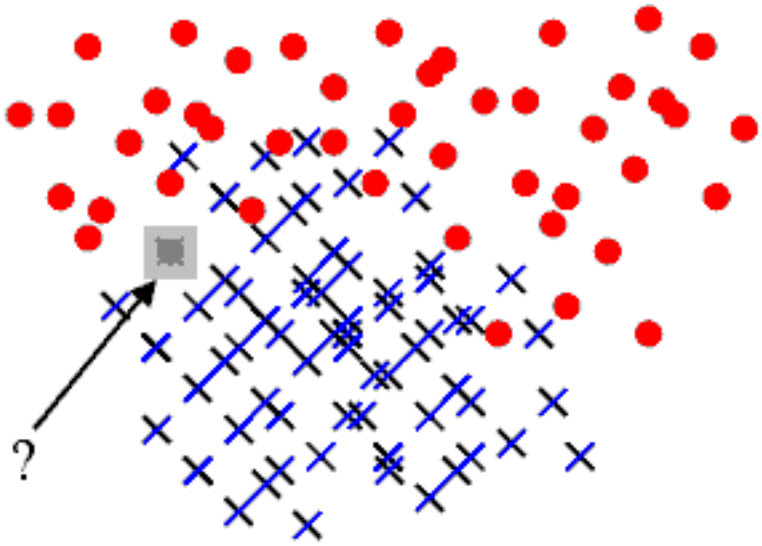


Courbe d'inertie : point d'inflexion pour $d = 4$, l'expérience montre que généralement, seules les projections sur un ou plusieurs des quatre premiers vecteurs propres reflètera l'information contenue par le nuage de points.

3.1.10 Bibliographie

3.2 Classification à l'aide des plus proches voisins

La figure suivante représente un problème de classification classique. On dispose d'un nuage de points réparti en deux classes. Un nouveau point semblable aux précédents se présente, sa classe est inconnue. L'objectif est de lui attribuer une classe en utilisant le fait qu'on connaît la classe d'appartenance des autres points.



A partir d'un nuage de points pour lesquels la classe d'appartenance est connue, comment classer un nouveau point pour lequel cette classe est inconnue ? Une méthode simple consiste à attribuer à ce nouveau point la même classe que le plus proche des points appartenant au nuage initial. C'est la méthode des plus proches voisins (ou nearest neighbours [47]) Elle est facile à implémenter mais peu utilisée car souvent très gourmande en temps de calcul lorsque le nuage de points est conséquent. Le premier paragraphe décrit cette méthode, les suivants cherchent à accélérer l'algorithme selon que le nuage de points appartient à un espace vectoriel ou non. La dernière partie présente l'algorithme *LAESA* pour le cas où le nuage de points appartient à un espace métrique quelconque.

3.2.1 Principe

Cette méthode est la plus simple puisqu'elle consiste à associer à x, l'élément à classer, le label $c\left(x_{i^*}\right)$ de l'élément le plus proche x_{i^*} dans l'ensemble $(x_1, \ldots, x_N)$. Ceci mène à l'algorithme de classification suivant :

Algorithme A1 : 1-PPV ou plus proche voisin

Soit $X = (x_1, \ldots, x_N) \subset E$ un ensemble d'éléments d'un espace métrique quelconque, soit $\left(c\left(x_1\right), \ldots, c\left(x_N\right)\right)$ les classes associées à chacun des éléments de X. On note d la distance définie sur l'espace métrique E. Soit x un élément à classer, on cherche à déterminer la classe $\hat{c}(x)$ associée à x. On définit x_{i^*} comme étant :

$$x_{i^*} = \underset{i \in \{1, \cdots, N\}}{\arg\min} \; d\left(x_i, x\right)$$

Alors $\hat{c}(x) = c\left(x_i^*\right)$.

Cet algorithme est souvent appelé *1-PPV* (ou *1-NN* pour Nearest Neighbors). Il existe une version améliorée *k-PPV* qui consiste à attribuer à x la classe la plus représentée parmi ses k plus proches voisins.

Algorithme A2 : k-PPV ou k-plus proches voisins

Soit $X = (x_1, \ldots, x_N) \subset E$ un ensemble d'éléments d'un espace métrique quelconque, soit $\left(c\left(x_1\right), \ldots, c\left(x_N\right)\right)$ les classes associées à chacun des éléments de X. On note d la distance définie sur l'espace métrique E. $\omega(x, y)$ est une fonction strictement positive mesurant la ressemblance entre x et y. Soit x un élément à classer, on cherche à déterminer la classe $c(x)$ associée à x. On définit l'ensemble S_k^* incluant les k-plus proches voisins de x, cet ensemble vérifie :

$$card S_k^* = 0 \text{ et } \max_{y \in S_k^*} d\left(y, x\right) \leqslant \min_{y \in X - S_k^*} d\left(y, x\right)$$

47. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

On calcule les occurrences $f(i)$ de chaque classe i dans l'ensemble S_k^* :

$$f(i) = \sum_{y \in S_k^*} \omega(x, y) \mathbb{1}_{\{c(y)=i\}} \tag{3.12}$$

On assigne alors à x la classe :math :'c(x)$ choisie dans l'ensemble :

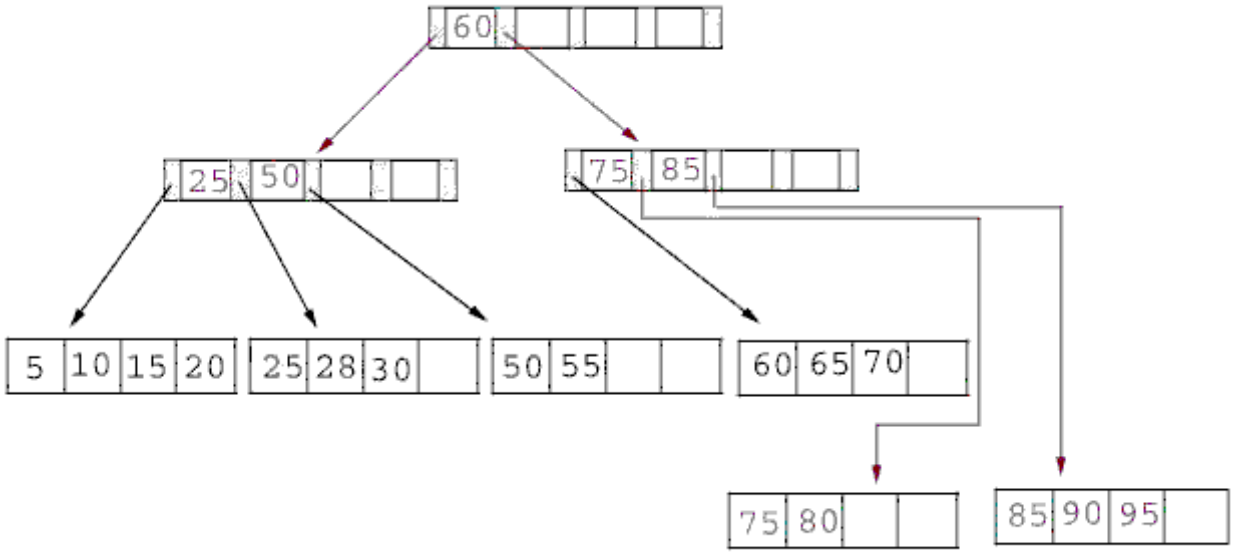
$$\hat{c}(x) \in \arg\max_{i \in \mathbb{N}} f(i)$$

Dans sa version la plus simple, la fonction $\omega(x, y)$ utilisée lors du calcul de la contribution *f (1)* (page **??**) est constante. Mais il est possible de lui affecter une valeur tenant compte de la proximité entre x et y. La table suivante donne quelques exemples de contributions possibles.

Exemple de contribution $w(x, y)$ pour l'algorithme des *k-ppv* (page 70). Ces fonctions sont toutes décroissantes (strictement ou non) par rapport à la distance d. L'inconvénient majeur de la méthode des plus proches voisins est sa longueur puisqu'elle implique le calcul des distances entre x et chacun des éléments de l'ensemble $(x_1, \ldots, x_N)$. C'est pourquoi de nombreuses méthodes d'optimisation ont été développées afin d'accélérer ce processus. Les deux premiers paragraphes traitent le cas où les points x_i appartiennent à un espace vectoriel et ont donc des coordonnées. Les suivant traitent le cas où les points x_i n'ont pas de coordonnées et appartiennent à un espace métrique quelconque.

3.2.2 B+ tree

Ce premier algorithme B+ tree [48] s'applique dans le cas réel afin d'ordonner des nombres dans un arbre de sorte que chaque noeud ait un père et pas plus de n fils.



Définition D1 : B+ tree

Soit B_n un B+ tree [49], soit N un noeud de B_n, il contient un vecteur $V(N) = (x_1, \ldots, x_t)$ avec $0 \leqslant t \leqslant n$ et $x_1 < \ldots < x_t$. Ce noeud contient aussi exactement $t - 1$ noeuds fils notés $(N_1, \ldots, N_{t-1})$. On désigne par $D(N_t)$

48. https://en.wikipedia.org/wiki/B%2B_tree
49. https://en.wikipedia.org/wiki/B%2B_tree

l'ensemble des descendants du noeud N_t et $G(N_t) = \{V(M) | M \in D(N_t)\}$. Le noeud N vérifie :

$$\forall x \in G(N_t),\ x_t \leqslant x < x_{t+1}$$
avec par convention $x_0 = -\infty$ et $x_{t+1} = +\infty$

Cet arbre permet de trier une liste de nombres, c'est une généralisation du tri quicksort [50] pour lequel $n = 2$. Comme pour le tri *quicksort*, l'arbre est construit à partir d'une série d'insertions et de cet ordre dépend la rapidité du tri. L'espérance du coût (moyenne sur tous les permutations possibles de k éléments), le coût de l'algorithme est en $O(k \log_n k)$.

3.2.3 R-tree ou Rectangular Tree

L'arbre R-tree [51] est l'adaptation du mécanisme du B+ tree [52] au cas multidimensionnel (voir *[Guttman1984]* (page **??**)). La construction de cet arbre peut se faire de manière globale - construction de l'arbre sachant l'ensemble de points à classer - ou de manière progressive - insertion des points dans l'arbre les uns à la suite des autres -. Toutefois, ces méthodes sont resteintes à des espaces vectoriels.

Il n'existe pas une seule manière de construire un R-tree [53], les noeuds de ces arbres suivent toujours la contrainte des B+ tree [54] qui est d'avoir un père et au plus n fils. Les R-tree [55] ont la même structure que les B+ tree [56] ôtée de leurs contraintes d'ordonnancement des fils. De plus, ces arbres organisent spatialement des rectangles ou boîtes en plusieurs dimensions comme le suggère la figure précédente. Les boîtes à organiser seront nommés les objets, ces objets sont ensuite regroupés dans des boîtes englobantes. Un noeud n d'un R-tree [57] est donc soit une feuille, auquel cas la boîte qu'il désigne est un objet, dans ce cas, il n'a aucun fils, soit le noeud désigne une boîte englobante $B(n)$. On désigne par $\mathcal{B}$ l'ensemble des boîtes d'un espace vectoriel quelconque et $v(b)$ désigne son volume. Pour un noeud n non feuille, $A(n)$ désigne l'ensemble des descendants de ce noeud. $B(n)$ est défini par :

$$B(n) = \arg\min \{v(b) | b \in \mathcal{B} \text{ et } \forall n' \in A(n'),\ B(n') \subset B(n)\}$$

La recherche dans un R-tree [58] consiste à trouver tous les objets ayant une intersection avec une autre boîte ou fenêtre W, soit l'ensemble L :

$$L = \{B(n) | B(n) \text{ est un objet et } B(n) \cap W \neq \emptyset\}$$

Cet ensemble est construit grâce à l'algorithme suivant :

Algorithme A3 : recherche dans un R-tree

Les notations sont celles utilisées dans ce paragraphe. On désigne par r le noeud racine d'un R-tree [59]. Soit n un noeud, on désigne par $F(n)$ l'ensemble des fils de ce noeud.

initialisation

$L \longleftarrow 0$ | $N \longleftarrow \{r\}$

itération

50. https://fr.wikipedia.org/wiki/Tri_rapide
51. https://en.wikipedia.org/wiki/R-tree
52. https://en.wikipedia.org/wiki/B%2B_tree
53. https://en.wikipedia.org/wiki/R-tree
54. https://en.wikipedia.org/wiki/B%2B_tree
55. https://en.wikipedia.org/wiki/R-tree
56. https://en.wikipedia.org/wiki/B%2B_tree
57. https://en.wikipedia.org/wiki/R-tree
58. https://en.wikipedia.org/wiki/R-tree
59. https://en.wikipedia.org/wiki/R-tree

while $N \neq \emptyset$

for n in $1..N$
if $W \cap B(n) \neq \emptyset$
$N \longleftarrow N \cup F(n)$
if $B(n)$ est un objet
$L \longleftarrow B(n)$

L est l'ensemble cherché.

Il reste à construire le R-tree [60], opération effectuée par la répétition successive de l'algorithme suivant permettant d'insérer un objet dans un R-tree [61].

Algorithme A4 : insertion d'un objet dans un R-tree

Les notations utilisées sont les mêmes que celles de l'algorithme de *recherche* (page 72). On cherche à insérer l'object E désigné par son noeud feuille e. On suppose que l'arbre contient au moins un noeud, sa racine r. On désigne également par $p(n)$ le père du noeud n. Chaque noeud ne peut contenir plus de s fils. On désigne par $v^*(G) = \min\{P | P \in \mathcal{B} \text{ et } \cup_{g \in G} B(g) \subset P\}$.

sélection du noeud d'insertion

$n^* \longleftarrow r$
tant que n^* n'est pas un noeud feuille
On choisit le fils f de n^*
qui minimise l'accroissement $v_f - v(B(f))$
du volume avec v_f défini par :
$v_f = \min\{v(P) | P \in \mathcal{B} \text{ et } B(f) \cup B(e) \subset P\}$
$n^* \longleftarrow f$

ajout du noeud

Si $p(n^*)$ a moins de s fils, alors le noeud e devient le fils de $p(n^*)$ et $B(p(n^*))$ est mis à jour d'après l'étape précédente. L'insertion est terminée. Dans le cas contraire, on sépare découpe le noeud $p(n^*)$ en deux grâce à l'étape suivante.

découpage des noeuds

L'objectif est de diviser le groupe G composé de $s+1$ noeuds en deux groupes G_1 et G_1. Tout d'abord, on cherche le couple (n_1, n_2) qui minimise le critère $d = v^*(\{n_1, n_2\}) - v(B(n_1)) - v(B(n_2))$ alors : $G_1 \longleftarrow n_1$, $G_2 \longleftarrow n_2$ et $G \longleftarrow G - G_1 \cup G_2$

tant que $G \neq \emptyset$
On choisit un noeud $n \in G$, on détermine i^*
tel que :math :‘vpa{acc{n} cup G_i} - vpa{G_i}$ soit minimal.
$G \longleftarrow G - \{n\}$
$G_{i^*} \longleftarrow G_{i^*} \cup \{n\}$

60. https://en.wikipedia.org/wiki/R-tree
61. https://en.wikipedia.org/wiki/R-tree

Si la recherche est identique quel que soit l'arbre construit, chaque variante de la construction de l'arbre tente de minimiser les intersections des boîtes et leur couverture. Plus précisément, l'étape qui permet de découper les noeuds est conçue de manière à obtenir des boîtes englobantes de volume minimale et/ou d'intersection minimale avec d'autres boîtes englobantes. L'algorithme R+~Tree (voir *[Sellis1987]* (page **??**)) essaye de minimiser les intersections entre boîtes et les objets à organiser sont supposés n'avoir aucune intersection commune. La variante R* tree [62] (voir *[Beckmann1990]* (page **??**)) effectue un compromis entre l'intersection et la couverture des boîtes englobantes. L'algorithme X-tree [63] (voir *[Berchtold1996]* (page **??**)) conserve l'historique de la construction de l'arbre ce qui lui permet de mieux éviter les intersections communes entre boîtes. Ces techniques appartiennent à une classe plus larges d'algorithmes de type Branch and Bound [64].

3.2.4 LAESA

Cet algorithme permet de chercher les plus proches voisins dans un ensemble inclus dans un espace métrique quelconque. Il s'appuie sur l'inégalité triangulaire. L'algorithme LAESA [65] ou *Linear Approximating Eliminating Search Algorithm*, (voir *[Rico-Juan2003]* (page **??**)) consiste à éviter un trop grand nombre de calculs de distances en se servant de distances déjà calculées entre les éléments de E et un sous-ensemble B inclus dans E contenant des *pivots*. La sélection des pivots peut être aléatoire ou plus élaborée comme celle effectuée par l'algorithme qui suit, décrit dans l'article *[Moreno2003]* (page **??**).

Algorithme A5 : LAESA : sélection des pivots

Soit $E = \{y_1, \ldots, y_N\}$ un ensemble de points, on cherche à déterminer un sous-ensemble de pivots $B = \{p_1, \ldots, p_P\} \subset E$.

initialisation

$B \longleftarrow y \in E$ choisi arbitrairement.

calcul de la fonction g

for y in $E - B$
    $g(y) \longleftarrow 0$
    for p in B
        $g(y) \longleftarrow g(y) + d(y, p)$

mise à jour de B

Trouver $p^* \in \arg\max \{g(p) | p \in E - B\}$
$B \longleftarrow B \cup \{p^*\}$
Si $cardB < P$, retour à l'étape précédente sinon fin.

L'algorithme LAESA [66] utilise les pivots pour diminuer le nombre de calculs en utilisant l'inégalité triangulaire. Par exemple, soit x un élément à classer, p_j un pivot, y_i un point du nuage. On suppose qu'on connaît $d(x, p_j)$, $d(p_j, y_i)$

62. https://en.wikipedia.org/wiki/R*_tree
63. https://en.wikipedia.org/wiki/X-tree
64. https://en.wikipedia.org/wiki/Branch_and_bound
65. https://tavianator.com/aesa/
66. https://tavianator.com/aesa/

et d^* la distance du point x à un autre point du nuage. L'inégalité triangulaire permet d'affirmer que si : $d(x, y_i) \geqslant |d(x, p_j) - d(p_j, y_i)| > d^*$, alors il n'est pas nécessaire de calculer la distance $d(x, y_i)$ pour affirmer que $d(x, y_i) > d^*$. L'élément y_i ne peut être l'élément le plus proche.

Algorithme A6 : LAESA

Soit $E = \{y_1, \ldots, y_N\}$ un ensemble de points, $B = \{p_1, \ldots, p_P\} \subset E$ un ensemble de pivots inclus dans E. On cherche à déterminer le voisinage $V(x)$ de x inclus dans E vérifiant :

$$\forall y \in V(x),\ d(x, y) \leqslant \rho$$

On suppose que la matrice $M = (m_{ij})_{\substack{\leqslant i \leqslant P \\ 1 \leqslant j \leqslant N}}$ a été calculée préalablement comme suit :

$$\forall (i, j),\ m_{ij} = d(p_i, y_j)$$

initialisation

for i in $1..P$
    $d_i \longleftarrow d(x, p_i)$
$d^* \longleftarrow \min\{d_i | 1 \leqslant i \leqslant P\}$
d^* est la distance du point x au pivot le plus proche.

recherche du plus proche élément

$S \longleftarrow \emptyset$
for i in $1..N$
    $d' \longleftarrow \max\{|d_j - m_{ji}|\}$
    if $d' < d^*$
        $d \longleftarrow d(x, y_i)$
        if $d' \leqslant d^*$
            $d^* \longleftarrow d'$
            $S \longleftarrow \{y_i\}$

3.2.5 Résultats théoriques

L'article *[Farago1993]* (page **??**) démontre également qu'il existe une majoration du nombre moyen de calcul de distances pour peu que la mesure de l'espace contenant l'ensemble E et l'élément x soit connue et que l'ensemble $B = \{p_1, \ldots, p_P\}$ des pivots vérifie :

$$\begin{array}{rl} \exists (\alpha, \beta) \in \mathbb{R}_*^+ \text{ tels que} & \\ \forall (x, y) \in E^2,\ \forall i & \alpha\, d(x, y) \geqslant |d(x, p_i) - d(p_i, y)| \\ \forall (x, y) \in E^2, & \max_i |d(x, p_i) - d(p_i, y)| \geqslant \beta\, d(x, y) \end{array}$$

L'algorithme développé dans *[Farago1993]* (page **??**) permet de trouver le point de plus proche d'un élément x dans un ensemble :math :'E = ensemble{x_1}{x_N}'selon l'algorithme suivant :

Algorithme A7 : plus proche voisin d'après [Farago1993]_

Soit $E = \{x_1, \ldots, x_N\}$ et $B = \{p_1, \ldots, p_P\} \subset E \subset X$. Soit $x \in X$, un élément quelconque. On suppose que les valeurs $m_{ij} = d(x_i, p_j)$ ont été préalablement calculées.

initialisation

On calcule préalablement les coefficients $\gamma(x_i)$:

$$\forall i \in \{1, \ldots, N\},\ \gamma(x_i) \longleftarrow \max_{j \in \{1,\ldots,P\}} |m_{ij} - d(x, p_j)|$$

élaguage

On définit $t_0 \longleftarrow \min_i \gamma(x_i)$. Puis on construit l'ensemble $F(x) = \{x_i \in E | \gamma(x_i)\} \leqslant \frac{\alpha}{\beta} t_0$.

plus proche voisin

Le plus proche x^* voisin est défini par : $x^* \in \arg\min \{d(x, y) | y \in F(x)\}$.

Et un petit théorème.

Théorème T1 : [Farago1993]_ 1

Les notations sont celles de l'algorithme précédent. Il retourne le plus proche voisin x^* de x inclus dans E. Autrement dit, $\forall x \in X,\ x^* \in F(x)$.

Théorème T2 : [Farago1993]_ 2

Les notations sont celles du même algorithme. On définit une mesure sur l'ensemble X, $B(x, r)$ désigne la boule de centre x et de rayon r, $Z \in X$ une variable aléatoire, de plus :

$$p(x, r) = P_X(B(x, r)) = \mathbb{P}(Z \in B(x, r))$$

On suppose qu'il existe $d > 0$ et une fonction $f : X \longrightarrow \mathbb{R}$ tels que :

$$\lim_{r \to 0} \frac{p(x, r)}{r^d} = f(x) > 0$$

La convergence doit être uniforme et presque sûre. On note également F_N le nombre de calculs de dissimilarité effectués par l'algorithme où N est le nombre d'élément de E, P désigne toujours le nombre de pivots, alors :

$$\limsup_{n \to \infty} \mathbb{E} F_N \leqslant k + \left(\frac{\alpha}{\beta}\right)^{2d}$$

3.2.6 Implémentation

La classe `NuagePoints` implémente les nuages de points sans optimisation. Il utilise la même interface que sklearn.neighbors.NearestNeighbors [67]. La second classe `NuagePointsLeasa`.

<<<

```
import numpy
from mlstatpy.ml.kppv_laesa import NuagePointsLaesa

X = numpy.array([[0, 0], [3, 3], [1, 1]])
nuage = NuagePointsLaesa(2)
nuage.fit(X)
dist, indices = nuage.kneighbors(X)
print("distance", dist)
print("indices", indices)
```

67. http://scikit-learn.org/stable/modules/generated/neighbors.NearestNeighbors.html

>>>

```
    distance [0. 0. 0.]
    indices [0 1 2]
```

3.2.7 Bilbiographie

3.3 Liens entre factorisation de matrices, ACP, k-means

La factorisation de matrice non négative [68]. Cette méthode est utilisée dans le cadre de la recommandation de produits à des utilisateurs. Lire également *[Acara2011]* (page **??**), *[Gupta2010]* (page **??**).

3.3.1 Factorisation de matrices et rang

La factorisation d'une matrice [69] est un problème d'optimisation qui consiste à trouver pour une matrice $M \in \mathcal{M}_{pq}$ à coefficients positifs ou nuls :

$$M = WH$$

Où W et H sont de rang k et de dimension $W \in \mathcal{M}_{pk}$ et $H \in \mathcal{M}_{kq}$. *On s'intéresse ici au cas où les coefficients ne sont pas nécessairement positifs.* Si $k < rang(M)$, le produit WH ne peut être égal à M. Dans ce cas, on cherchera les matrices qui minimise :

Problème P1 : Factorisation de matrices positifs

Soit $M \in \mathcal{M}_{pq}$, on cherche les matrices à coefficients positifs $W \in \mathcal{M}_{pk}$ et $H \in \mathcal{M}_{kq}$ qui sont solution du problème d'optimisation :

$$\min_{W,h} \left\{ \|M - WH\|^2 \right\} = \min_{W,H} \sum_{ij} (m_{ij} - \sum_{k} w_{ik} h_{kj})^2$$

---

68. https://en.wikipedia.org/wiki/Non-negative_matrix_factorization
69. https://en.wikipedia.org/wiki/Non-negative_matrix_factorization

3.3.2 Quelques cas simples

Le notebook *Valeurs manquantes et factorisation de matrices* (page 320) montre la décroissante de l'erreur en fonction du rang et l'impact de la corrélation sur cette même erreur. Le dernier paragraphe montre qu'il n'existe pas de solution unique à un problème donné. L'exemple suivant s'intéresse à une matrice 3x3. Les trois points forment un triangle dans un plan.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from numpy import identity, array

M = identity(3)
W = array([[0.5, 0.5, 0], [0, 0, 1]]).T
H = array([[1, 1, 0], [0.0, 0.0, 1.0]])
wh = W @ H

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(M[:,0], M[:,1], M[:,2], c='b', marker='o', s=600, alpha=0.5)
ax.scatter(wh[:,0], wh[:,1], wh[:,2], c='r', marker='^')
plt.show()
```

On peut voir la matrice M comme un ensemble de $n = 3$ points dans un espace vectoriel. La matrice W est un ensemble de $k < n$ points dans le même espace. La matrice WH, de rang k est une approximation de cet ensemble dans le même espace, c'est aussi n combinaisons linéaires de k points de façon à former n points les plus proches proches de n points de la matrice M.

3.3.3 Intuition géométrique

L'exemple précédente suggère une interprétation géométrique d'une factorisation de matrice. Sans valeur manquante, ce problème est équivalent à une Analyse en Composantes Principales (ACP) [70] (voir aussi *[Boutsidis2008]* (page **??**) (décomposition en valeurs singulières comme algorithme d'initialisation). Nous allons le montrer grâce à quelques lemmes et théorèmes.

Lemme L1 : Rang k

On note $M = (m_{ij})$, $W^k = (w^k_{il})$, $H^k = (h^k_{lj})$ avec $1 \leqslant i \leqslant p$, $1 \leqslant j \leqslant q$, et $1 \leqslant l \leqslant k$ avec $k < \min(p, q)$. On suppose que les matrices sont solution du problème d'optimisation $\min_{W,H} \|M - WH\|^2$. On suppose que $rang(M) \geqslant k$. Alors les les matrices W^k et H^k sont de rang k.

On procède par récurrence. Ce lemme est nécessairement vrai pour $k = 1$ car la matrice M n'est pas nulle. De manière évidente, $\left\|M - W^{k-1}H^{k-1}\right\|^2 \geqslant \left\|M - W^kH^k\right\|^2$. Comme $rang(M) \geqslant k$, il existe un vecteur colonne V de la matrice M qui ne fait pas partie de l'espace vectoriel engendré par les $k-1$ vecteurs de la matrice W^{k-1}. On construit la matrice $Y^k = [W^{k-1}, V]$. Par construction, $rang(Y) = k$. De même, on construit G^k à partir de H^{k-1} en remplaçant la dernière colonne et en ajoutant une ligne :

$$G^k = \begin{bmatrix} H^{k-1}[1..p-1] & 0 \\ 0 & 1 \end{bmatrix}$$

Par construction, le dernier vecteur est de la matrice produit est identique à celui de la matrice M.

$$\left\|M - Y^{k-1}G^{k-1}\right\|^2 = \left\|M - W^{k-1}H^{k-1}\right\|^2 - \sum_i (m_{iq} - w^{k-1}_{ik}h^{k-1}_{kq})^2$$

70. https://fr.wikipedia.org/wiki/Analyse_en_composantes_principales

Nous avons fabriqué une matrice de rang k qui fait décroître l'erreur du problème d'optimisation. On procède par l'absurde pour dire que si $rang(W) = k-1$, on peut construire une matrice de rang k qui fait décroître l'erreur ce qui est impossible. Le lemme est donc vrai.

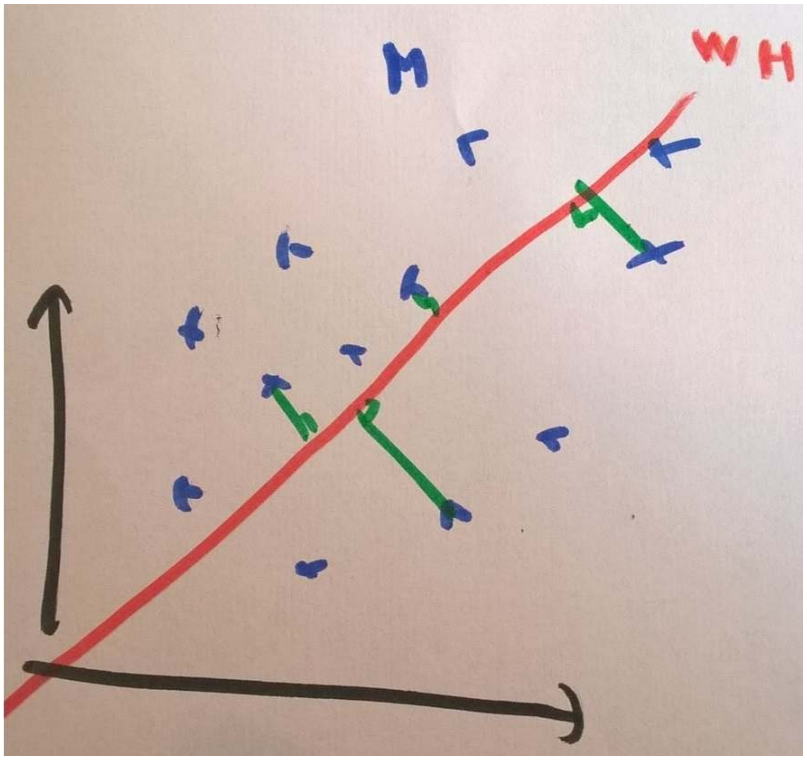
Ce lemme fait également apparaître la construction de q points dans un espace vectoriel engendré par les k vecteurs colonnes de la matrice W_k. Il est donc possible de choisir n'importe quel base W'_k de cet espace et d'exprimer les q points de $W_k H_k$ avec cette nouvelle base. Cela signifie qu'on peut écrire la matrice W_k dans une base B_k comme $W_k = B_k C_k$ et $W_k H_k = B_k C_k C_k^{-1} G_k$.

Lemme L2 : Projection

On note $M = (m_{ij})$, $W^k = (w^k_{il})$, $H^k = (h^k_{lj})$ avec $1 \leqslant i \leqslant p$, $1 \leqslant j \leqslant q$, et $1 \leqslant l \leqslant k$ avec $k < \min(p,q)$. On suppose que les matrices sont solution du problème d'optimisation $\min_{W,H} \|M - WH\|^2$. On considère que la matrice M est un ensemble de q points dans dans un espace vectoriel de dimension p. La matrice WH représente des projections de ces points dans l'espace vectoriel engendré par les k vecteurs colonnes de la matrice W.

La figure suivante illustre ce lemme. $\|M - WH\|^2$ s'écrit comme la somme des distances entre q points :

$$\|M - WH\|^2 = \sum_{j=1}^{q} \|M[j] - W_k H_k[j]\|^2$$



Or on sait que si W_k est fixé, les q points de la matrice $W_k H_k$ évolue sur un hyperplan de dimension k. Le point de ce plan le plus du vecteur $M[j]$ est sa projection sur ce plan.

Théorème T1 : La factorisation de matrice est équivalente à une analyse en composantes principales

On note $M = (m_{ij})$, $W^k = (w^k_{il})$, $H^k = (h^k_{lj})$ avec $1 \leqslant i \leqslant p$, $1 \leqslant j \leqslant q$, et $1 \leqslant l \leqslant k$ avec $k < \min(p,q)$. On suppose que les matrices sont solution du problème d'optimisation $\min_{W,H} \|M - WH\|^2$. On considère que la matrice M est un ensemble de q points dans dans un espace vectoriel de dimension p. On suppose $p < q$. La matrice

W_k définit un hyperplan identique à celui défini par les k vecteurs propres associés aux k plus grande valeurs propres de la matrice MM' où M' est la transposée de M.

Une analyse en composante principale consiste à trouver l'hyperplan qui maximise l'inertie de la projection d'un nuage sur ce plan. Le théorème *résolution de l'ACP* (page 64) a montré que :

$$S = \underset{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}}{\arg\max} \left[\sum_{i=1}^{N} \|W'X_i\|^2 \right] = \underset{W \in M_{p,d}(\mathbb{R})}{\arg\min} \left[\sum_{i=1}^{N} \|WW'X_i - X_i\|^2 \right]$$

Dans notre cas, chaque ligne de la matrice M est un vecteur X_i. La matrice W_k est identique à celle cherchée lors du problème de factorisation de matrices. Les colonnes de la matrice H_k sont égales à $W'X_i$. Il reste à montrer que le minimum trouvé dans les deux problèmes est le même. Le notebook *Factorisation et matrice et ACP* (page 314) montre que cela fonctionne sur un exemple. La démonstration du théorème montre également que $W'W = I_d$ et dans ce cas précis, $WW'X_i$ représente les coordonnées de la projection du point X_i sur le plan défini par les vecteurs W. C'est aussi ce que montre *second lemmme* (page 79). S'il s'agit du même plan, cela signifie que les deux formulations, ACP et factorisation de matrices, aboutissent au même minimum. Comme l'algorithme de l'ACP détermine le meilleur plan projecteur, nécessairement, il correspond à celui trouvé par la factorisation de matrice.

3.3.4 k-means

On peut construire deux matrices W et H à partir des résultats d'un *k-means* (page 3). Celui-ci détermine k centres auxquels on effecte les points du nuage de départ. Dans ce cas-ci, la matrice W est constituée des coordonnées de ces centres. On note C_l le cluster l, la matrice $H^k = (h^k_{lj})$ est définie comme suit :

$$h^k_{lj} = \mathbb{1}_{\{X_j \in C_l\}}$$

Les coefficients sont soit 0 ou 1. On peut alors essayer de forcer la factorisation de matrice vers une matrice H avec pas de un 1 sur chaque colonne et des zéros partout ailleurs. Le résultat sera assez proche d'un clustering.

3.3.5 Quelques résultats

Le notebook *Factorisation et matrice et ACP* (page 314) illustre le lien entre ACP et factorisation de matrice en deux dimensions.

3.3.6 Prolongements

Tous les résultats montrés ici ne sont valables que si la norme L_2 est utilisée. Cela permet de mieux comprendre les références proposées dans la documentation de Non-negative matrix factorization (NMF or NNMF) [71]. Si l'ACP et la factorisation de matrices sont équivalentes, les algorithmes pour trouver le minimum diffèrent et sont plus ou moins appropriés dans certaines configurations. Lire *[Gilles2014]* (page **??**).

Prédiction

Prédire revient à supposer que la matrice M est composée de vecteurs colonnes $X_1, ..., X_q$. La matrice W reste inchangée et la prédiction revient à déterminer les coordonnées de la projection d'un nouveau point X_{q+1} dans le plan définit par W.

71. http://scikit-learn.org/stable/modules/decomposition.html#nmf

Factorisation non-négative

Le problème le plus souvent évoqué est celui de la factorisation non-négative : NMF [72]. Ce problème est une optimisation avec contrainte : les coefficients doivent tous être positifs ou nuls. Il n'est bien sûr plus équivalent à une ACP. En revanche, la factorisation de matrice est un problème équivalent à celui résolu par la Décomposition en Valeur Singulière (SVD) [73] qui cherche à décomposer une matrice $M = U\Sigma V^*$. La matrice Σ est une matrice diagonale.

Norme

L'ACP avec une norme L_1 revient à trouver le plan qui minimise la somme des distances à la projection et non la somme des distances au carrés. Cela réduit l'impact des points aberrants mais le problème n'est plus équivalent à la factorisation de matrices avec une norme L_1.

Sparsité

Une ACP suppose que le calcul de valeurs propres d'une matrice et c'est fastidieux lorsque la dimension du problème est très grande. On lui préfère alors un algorithme tel que Sparse PCA [74]. La factorisation de matrice est plus efficace qu'une ACP sur les problèmes sparses et de grande dimension. Lire Non-negative Matrix Factorization with Sparseness Constraints [75].

Valeurs manquantes

Contourner le problème des valeurs manquantes veut souvent dire, soit supprimer les enregistrements contenant des valeurs manquantes, soit choisir un modèle capable de faire avec ou soit trouver un moyen de les remplacer. On peut gérer plus facilement le problème des valeurs manquantes avec une factorisation de matrices. On peut également se server de la méthode pour calculer une ACP avec des valeurs manquantes.

— Imputation de données manquantes [76]
— Principal component analysis with missing values : a comparative survey of methods [77]

Interprétation

La factorisation de matrice peut être utilisée comme outil de segmentation et d'interprétation pour des images, des vidéos. Lire A tutorial on Non-Negative Matrix Factorisation with Applications to Audiovisual Content Analysis [78].

— Gesture recognition using a NMF-based representation of motion-traces extracted from depth silhouettes [79]

NTF

Le problème de Non-Negative Matrix Factorisation (NMF) [80] est un cas particulier de Non-Negative Tensor Factorisation (NTF) [81]. Lire aussi PARAFAC. Tutorial and applications [82].

72. https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-explo-nmf.pdf
73. https://fr.wikipedia.org/wiki/D%C3%A9composition_en_valeurs_singuli%C3%A8res
74. http://scikit-learn.org/stable/modules/decomposition.html#sparsepca
75. http://www.jmlr.org/papers/volume5/hoyer04a/hoyer04a.pdf
76. https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-idm.pdf
77. http://pbil.univ-lyon1.fr/members/dray/files/articles/dray2015a.pdf
78. http://perso.telecom-paristech.fr/~essid/teach/NMF_tutorial_ICME-2014.pdf
79. https://hal.archives-ouvertes.fr/hal-00990252/document
80. https://en.wikipedia.org/wiki/Non-negative_matrix_factorization
81. http://www.cs.huji.ac.il/~shashua/papers/NTF-icml.pdf
82. https://www.cs.cmu.edu/~pmuthuku/mlsp_page/lectures/Parafac.pdf

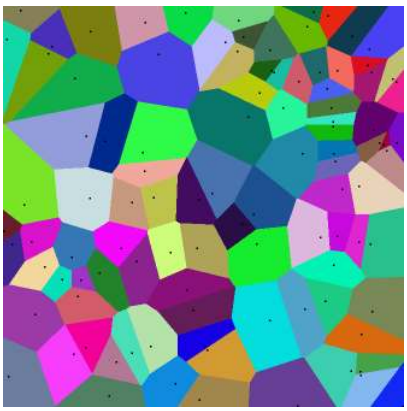
3.3.7 Bibliographie

3.4 Régression logistique, diagramme de Voronoï, k-Means

Ce qui suit explore les liens entre une régression logistique, les diagrammes de Voronoï pour construire un classifieur qui allie la régression logistique et les clustering type *k-means* (page 3). Le point de départ est une conjecture : les régions créées par une régression logistique sont convexes.

3.4.1 Diagramme de Voronoï

Un diagramme de Voronoï [86] est le diagramme issu des intersections des médiatrices entre n points.



On définit un ensemble de points $(X_1, ..., X_n)$. La zone d'influence de chaque point est défini par $V(X_i) = \{x | d(x, X_i) \leqslant d(x, X_j) \forall j\}$. Si d est la distance euclidienne, la frontière entre deux points X_i, X_j est un segment sur la droite d'équation $d(x, X_i) = d(x, X_j)$:

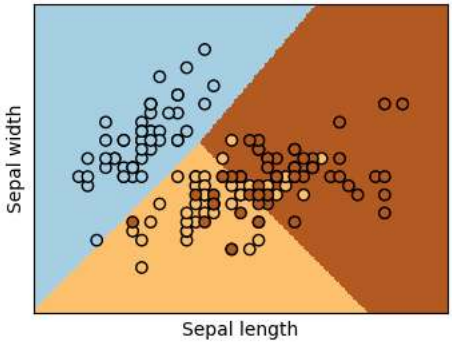
$$
\begin{array}{rl}
& \|x - X_i\|^2 - \|x - X_j\|^2 = 0 \\
\Longrightarrow & \|x\|^2 - 2\langle x, X_i\rangle + \|X_i\|^2 - (\|x\|^2 - 2\langle x, X_j\rangle + \|X_j\|^2) = 0 \\
\Longrightarrow & 2\langle x, X_j - X_i\rangle + \|X_i\|^2 - \|X_j\|^2 = 0 \\
\Longrightarrow & 2\langle x, X_j - X_i\rangle + \frac{1}{2}\langle X_i + X_j, X_i - X_j\rangle = 0 \\
\Longrightarrow & \left\langle x - \frac{X_i + X_j}{2}, X_i - X_j \right\rangle = 0
\end{array}
$$

Ce système constitue $\frac{n(n-1)}{2}$ droites ou hyperplans si l'espace vectoriel est en dimension plus que deux. Le diagramme de Voronoï est formé par des segments de chacune de ces droites. On peut retourner le problème. On suppose qu'il existe $\frac{n(n-1)}{2}$ hyperplans, existe-t-il *n* points de telle sorte que les hyperplans initiaux sont les frontières du diagramme de Voronoï formé par ces *n* points ? Les paragraphes qui suivent expliquent explorent cette hypothèse.

86. https://fr.wikipedia.org/wiki/Diagramme_de_Vorono%C3%AF

3.4.2 Régression logistique

scikit-learn [87] a rendu populaire le jeu de données Iris [88] qui consiste à classer des fleurs en trois classes en fonction des dimensions de leurs pétales.



<<<

```
from sklearn.datasets import load_iris
data = load_iris()
X, y = data.data[:, :2], data.target

from sklearn.linear_model import LogisticRegression
clr = LogisticRegression()
clr.fit(X, y)

print("coef_", clr.coef_)
print("intercept_", clr.intercept_)
```

>>>

```
    /usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:␣
↪FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver␣
↪to silence this warning.
      FutureWarning)
    /usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:459:␣
↪FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify the␣
↪multi_class option to silence this warning.
      "this warning.", FutureWarning)
    coef_ [[-2.48966566  3.99890697]
     [ 0.49528644 -1.63324843]
     [ 1.15716697 -1.77500049]]
    intercept_ [ 0.80059233  1.23763338 -2.21755542]
```

La fonction de prédiction est assez simple : $f(x) = Ax + B$. La classe d'appartenance du point x est déterminé par $\max_i f(x)_i$. La frontière entre deux classes *i, j* est définie par les deux conditions : $\max_k f(x)_k = f(x)_i = f(x)_j$. On retrouve bien $\frac{n(n-1)}{2}$ hyperplans. On définit la matrice A comme une matrice ligne $(L_1, ..., L_n)$ où n est le nombre de

87. http://scikit-learn.org/
88. http://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html#sphx-glr-auto-examples-linear-model-plot-iris-logistic-py

classes. L'équation de l'hyperplan entre deux classes devient :

$$\begin{array}{rl} & L_iX + B_i = L_jX + B_j \\ \iff & \langle L_i - L_j, X\rangle + B_i - B_j = 0 \\ \iff & \langle L_i - L_j, X\rangle + B_i - B_j = 0 \\ \iff & \left\langle L_i - L_j, X - \frac{L_i+L_j}{2}\right\rangle + \left\langle L_i - L_j, \frac{L_i+L_j}{2}\right\rangle + B_i - B_j = 0 \\ \iff & \left\langle L_i - L_j, X - \frac{L_i+L_j}{2}\right\rangle + \frac{1}{2}\left\|L_i\right\|^2 - \frac{1}{2}\left\|L_j\right\|^2 + B_i - B_j = 0 \\ \iff & \left\langle L_i - L_j, X - \frac{L_i+L_j}{2}\right\rangle + \frac{1}{2}\left\|L_i\right\|^2 + B_i - (\frac{1}{2}\left\|L_j\right\|^2 + B_j) = 0 \end{array}$$

Il y a peu de chance que cela fonctionne comme cela. Avant de continuer, assurons-nous que les régions associées aux classes sont convexes. C'est une condition nécessaire mais pas suffisante pour avoir un diagramme de Voronoï.

Soit X_1 et X_2 appartenant à la classe *i*. On sait que $\forall k, L_iX_1 + B_i \geqslant L_kX_1 + B_k$ et $\forall k, L_iX_2 + B_i \geqslant L_kX_2 + B_k$. On considère un point X sur le segment $[X_1, X_2]$, donc il existe $\alpha, \beta \geqslant 0$ tel que $X = \alpha X_1 + \beta X_2$ et $\alpha + \beta = 1$. On vérifie que :

$$\begin{array}{rl} & L_iX + B_i = L_i(\alpha X_1 + \beta X_2) + B_i = \alpha(L_iX_1 + B_i) + \beta(L_iX_2 + B_i) \\ \geqslant & \alpha(L_kX_1 + B_k) + \beta(L_kX_2 + B_k) = L_k(\alpha X_1 + \beta X_2) + B_k \forall k \end{array}$$

Donc le point *X* appartient bien à classe *i* et celle-ci est convexe. La régression logistique forme une partition convexe de l'espace des features.

Théorème T1 : convexité des classes formées par une régression logistique

On définit l'application $\mathbb{R}^d \to \mathbb{N}$ qui associe la plus grande coordonnée $f(X) = \arg\max_k(AX + B)_k$. *A* est une matrice $\mathcal{M}_{dc}$, *B* est un vecteur de $\mathbb{R}^d$, *c* est le nombre de parties. L'application f définit une partition convexe de l'espace vectoriel $\mathbb{R}^d$.

Revenons au cas de Voronoï. La classe prédite dépend de $\max_k(Ax + B)_k$. On veut trouver *n* points $(P_1, ..., P_n)$ tels que chaque couple (P_i, P_j) soit équidistant de la frontière qui sépare leurs classes. Il faut également les projections des deux points sur la frontière se confondent et donc que les vecteurs $P_i - P_j$ et $L_i - L_j$ sont colinéaires.

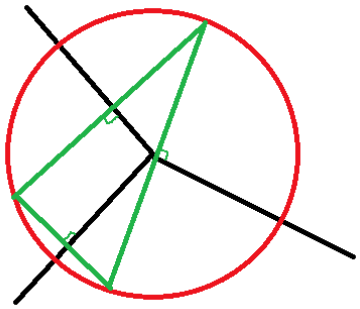
$$\iff \begin{cases} \langle L_i - L_j, P_i\rangle + B_i - B_j = -\left(\langle L_i - L_j, P_j\rangle + B_i - B_j\right) \\ P_i - P_j - \left\langle P_i - P_j, \frac{L_i-L_j}{\|L_i-L_j\|}\right\rangle \frac{L_i-L_j}{\|L_i-L_j\|} = 0 \end{cases} \iff \begin{cases} \langle L_i - L_j, P_i + P_j\rangle + 2(B_i - B_j) = 0 \\ P_i - P_j - \left\langle P_i - P_j, \frac{L_i-L_j}{\|L_i-L_j\|}\right\rangle \frac{L_i-L_j}{\|L_i-L_j\|} = 0 \end{cases}$$

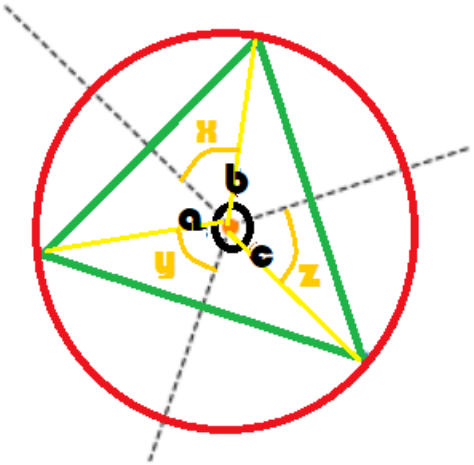
La seconde équation en cache en fait plusieurs puisqu'elle est valable sur plusieurs dimensions mais elles sont redondantes. Il suffit de choisir un vecteur u_{ij} non perpendiculaire à $L_i - L_j$ de sorte que qu'il n'est pas perpendiculaire au vecteur $L_i - L_j$ et de considérer la projection de cette équation sur ce vecteur. C'est pourquoi on réduit le système au suivant qui est équivalent au précédent si le vecteur u_{ij} est bien choisi.

$$\implies \begin{cases} \langle L_i - L_j, P_i + P_j\rangle + 2(B_i - B_j) = 0 \\ \langle P_i - P_j, u_{ij}\rangle - \left\langle P_i - P_j, \frac{L_i-L_j}{\|L_i-L_j\|}\right\rangle \left\langle \frac{L_i-L_j}{\|L_i-L_j\|}, u_{ij}\right\rangle = 0 \end{cases}$$

3.4.3 Diagramme de Voronoï et partition convexe

Faisons un peu de géométrie avant de résoudre ce problème car celui-ci a dans la plupart des cas plus d'équations que d'inconnues. Chaque frontière entre deux classes est la médiatrice d'un segment $[P_i, P_j]$. Le dessin suivant trace un diagramme de Voronoï à trois points. L'intersection est le centre des médiatrices du triangle formé par les points de Voronoï. Pour les trouver, on trace un cercle, n'importe lequel, puis une droite perpendiculaire à l'une des médiatrice. On obtient deux points. Le troisième est obtenu en traçant une seconde perpendiculaire et par construsction, la troisième droite est perpendiculaire à la troisième médiatrice. Et on nomme les angles.

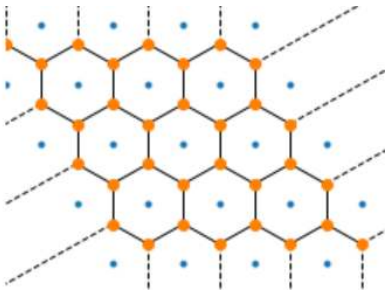


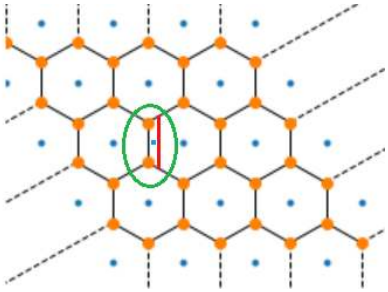


Les triangles formés par les côtés jaunes sont isocèles. On en déduit que $a+b+c = 2\pi = 2(x+y+z)$. On en déduit aussi que :

$$\begin{aligned} x+y &= a \\ y+z &= c \\ x+z &= b \end{aligned}$$

On en conclut que $a+b+c = 2\pi = 2(x+y+z) = 2(x+c)$ et $x = \pi - c$. Il existe une infinité de triplets de 3 points qui aboutissent à ce diagramme de Voronoï. Il suffit de changer la taille du cercle. On montre aussi qu'en dimension 2 et 3 classes, il existe toujours une solution au problème posé. Maintenant, si on considère la configuration suivante avec des points disposés de telle sorte que le diagramme de Voronoï est un maillage hexagonal. $a = b = c = \frac{2\pi}{3}$ et $x = y = z = \frac{\pi}{3}$. Il n'existe qu'un ensemble de points qui peut produire ce maillage comme diagramme de Voronoï. Mais si on ajoute une petite zone (dans le cercle vert ci-dessous), il est impossible que ce diagramme soit un diagramme de Voronoï bien que cela soit une partition convexe.



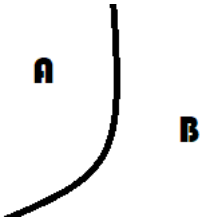


On revient à la détermination du diagramme de Voronoï associé à une régression logistique. On a montré qu'il n'existe pas tout le temps pour n'importe quelle partition convexe. Mais cela ne veut pas dire que tel est le cas pour une régression logistique.

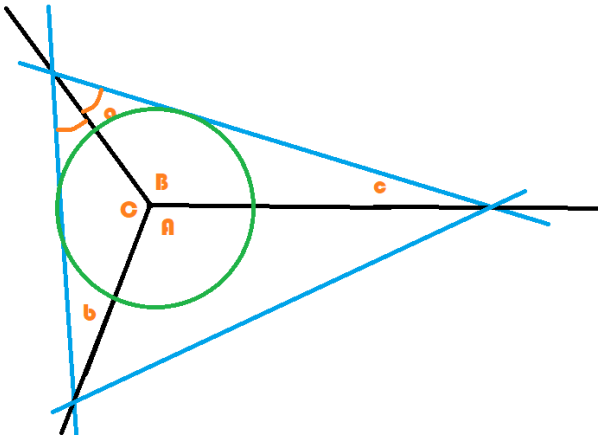
3.4.4 Régression logistique et partition convexe

On a montré que la régression logistique réalise une partition convexe de l'espace vectoriel des variables. On note *L(n)* l'ensemble des partitions à *n* classes. Le diagramme de Voronoï correspondent également à un sous-ensemble *V(n)*. $L(n) = V(n)$, that is the question.

On peut se poser la question de savoir si *L(n)* un sous-ensemble ou tout l'ensemble auquel cas la réponse à la question précédente est triviale. Considérons d'abord deux parties voisines d'une partition convexe formée par une fonction telle que celle décrite par le théorème sur la *convexité des classes formées par une régression logistique* (page 84).



L'image qui précède montre une partition qui n'est pas convexe. La partie *A* l'est mais pas la partie *B*. En fait, il est facile de montrer que la seule frontière admissible entre deux parties convexe est un hyperplan. Si la partition contient *n* partie, il y a au pire $\frac{n(n-1)}{2}$ frontières, ce qui correspond également au nombre d'hyperplans définis par la fonction de prédiction associée à la régression logistique.



L'image qui précède présente une classification en trois zones (droites noires). On a choisi une droite bleue au hasard. En prenant son symétrique par rapport à une des droites noires (*D*), on a deux droites D_1, D_2. L'ensemble des points $\{x|d(x, D_1) = d(x, D_2)\}$ correspond à la droite noire. Il doit être de même pour les trois droites bleues, autrement dit, l'intersection des droites est le centre du cercle inscrit dans le triangle bleu ce qui n'est visiblement pas le cas sur l'image. Il paraît vraisemblable de dire que les régressions logisitiques ne permettent pas de former toutes les partitions convexes. On pourrait le montrer mais cela ne permettrait pas de répondre à la question initiale $L(n) = V(n)$?

3.4.5 Voronoï et régression logistique

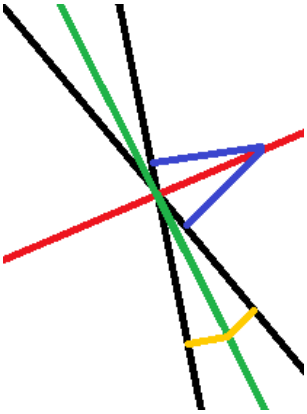
On sait que $L(2) = V(2)$ quelque soit la dimension, que $L(3) = V(3)$ en dimension 2. La matrice L définit une régression logistique. Le diagramme de Voronoï qui lui correspond est solution du système d'équations qui suit :

$$\Longrightarrow \left\{ \begin{array}{l} \left\langle \frac{L_i - L_j}{\|L_i - L_j\|}, P_i + P_j \right\rangle + 2\frac{B_i - B_j}{\|L_i - L_j\|} = 0 \\ \langle P_i - P_j, u_{ij} \rangle - \left\langle P_i - P_j, \frac{L_i - L_j}{\|L_i - L_j\|} \right\rangle \left\langle \frac{L_i - L_j}{\|L_i - L_j\|}, u_{ij} \right\rangle = 0 \end{array} \right. \tag{3.13}$$

Avec u_{ij} choisi de telle sorte que les vecteur $L_i - L_j$ et u_{ij} ne soit pas coliénaires. Ce système inclut des équations entre classes ou régions qui ne sont pas voisines. Il y a potentiellement $\frac{n(n-1)}{2}$ équations pour n inconnues. Il n'est pas évident de dire si ce système à une solution. Voyons plutôt l'ensemble des droites formées par un diagramme de Voronoï. Un point appartient à un segment s'il est à égale distance de deux points.

$$\begin{array}{ll} & \forall i < j,\ d(X, P_i) = d(X, P_j) \\ \Longleftrightarrow & \forall i < j,\ \|X - P_i\|^2 = \|X - P_j\|^2 \\ \Longleftrightarrow & \forall i < j,\ \langle (X - P_i) + (X - P_j), (X - P_i) - (X - P_j) \rangle \\ \Longleftrightarrow & \forall i < j,\ \left\langle X - \frac{P_i + P_j}{2}, P_j - P_i \right\rangle = 0 \\ \Longleftrightarrow & \forall i < j,\ \langle X, P_j - P_i \rangle + \frac{\|P_i\|^2}{2} - \frac{\|P_j\|^2}{2} = 0 \end{array}$$

Pour une partition convexe formée à partir de droite, comme c'est le cas d'une régression linéaire, un point appartient à un segment s'il est à égale distance de deux droites. L'ensemble de ces points correspond à deux droites, les deux bissectrices.



Seule l'une de ces droites est la bonne. L'équation d'une droite est donnée par $\langle X, L_i \rangle + B_i = 0$.

$$\left\langle X, \frac{L_i}{\|L_i\|} \right\rangle + \frac{B_i}{\|L_i\|} = \left\langle X, \frac{L_j}{\|L_j\|} \right\rangle + \frac{B_j}{\|L_j\|} \text{ ou } \left\langle X, \frac{L_i}{\|L_i\|} \right\rangle + \frac{B_i}{\|L_i\|} = -\left\langle X, \frac{L_j}{\|L_j\|} \right\rangle - \frac{B_j}{\|L_j\|}$$

On choisit l'une de ces droites.

$$\forall i < j,\ \left\langle X, \frac{L_j}{\|L_j\|} - \frac{L_i}{\|L_i\|} \right\rangle + \frac{B_j}{\|L_j\|} - \frac{B_i}{\|L_i\|} = 0$$

On peut voir que si tous les points sont situés sur la boule unité, à savoir $\|P_i\| = 1$, la régression logistique s'écrit simplement avec $L_i = P_i$ et $B_i = -\frac{1}{2}$. On revient au système d'équations linéaires *(1)* (page **??**) et on en cherche une solution approchée un peu à la façon RANSAC [89] avec une régression linéaire et la norme *L1*. Il n'existe pas toujours de diagramme de Voronoï équivalent à la partition convexe réalisée par une regréssion logistique. Il est facile de trouver un contre-exemple en essayant de résoudre le système précédent. C'est ce que fait la fonction `voronoi_estimation_from_lr`. La fonction essaye avec quelques approximations et heuristiques de déterminer les points du diagramme de Voronoï. Si elle réussit du premier coup, c'est qu'il y avait équivalence ce qui arrive peu souvent. Il faudrait refaire les calculs à la main et non de façon approchée pour valider un contre exemple. Une prochaine fois peut-être. Ce qu'il faut retenir est que la régression logistique réalise une partition convexe de l'espace des variables.

89. http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RANSACRegressor.html

3.4.6 Notebooks

Le notebook qui suit reprend les différents éléments théoriques présentés ci-dessus. Il continue l'étude d'une régression logistique et donne une intuition de ce qui marche ou pas avec un tel modèle. Notamment, le modèle est plus performant si les classes sont réparties sur la boule unité de l'espace des features.

Voronoï et régression logistique

Le notebook étudie la pertinence d'un modèle de régression logistique dans certaines configurations. Il regarde aussi le diagramme de Voronoï associé à une régression logistique à trois classes. Il donne quelques intuitions sur les modèles que la régression logistique peut résoudre.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

```
%matplotlib inline
```

Régression logistique

```
from sklearn.datasets import load_iris
data = load_iris()
X, y = data.data[:, :2], data.target
```

```
from sklearn.linear_model import LogisticRegression
clr = LogisticRegression()
clr.fit(X, y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
clr.coef_
```

```
array([[-2.49579289,  4.01011301],
       [ 0.49709451, -1.63380222],
       [ 1.15921404, -1.77736568]])
```

```
clr.intercept_
```

```
array([ 0.81713932,  1.22543562, -2.22516119])
```

```
import numpy
x = numpy.array([[1, 2]])
clr.decision_function(x)
```

```
array([[ 6.34157245, -1.54507432, -4.6206785 ]])
```

```
A = clr.coef_
B = clr.intercept_
```

On vérifie que la fonction de décision correspond à la formule suivant.

```
(A@x.T).T.ravel() + B
```

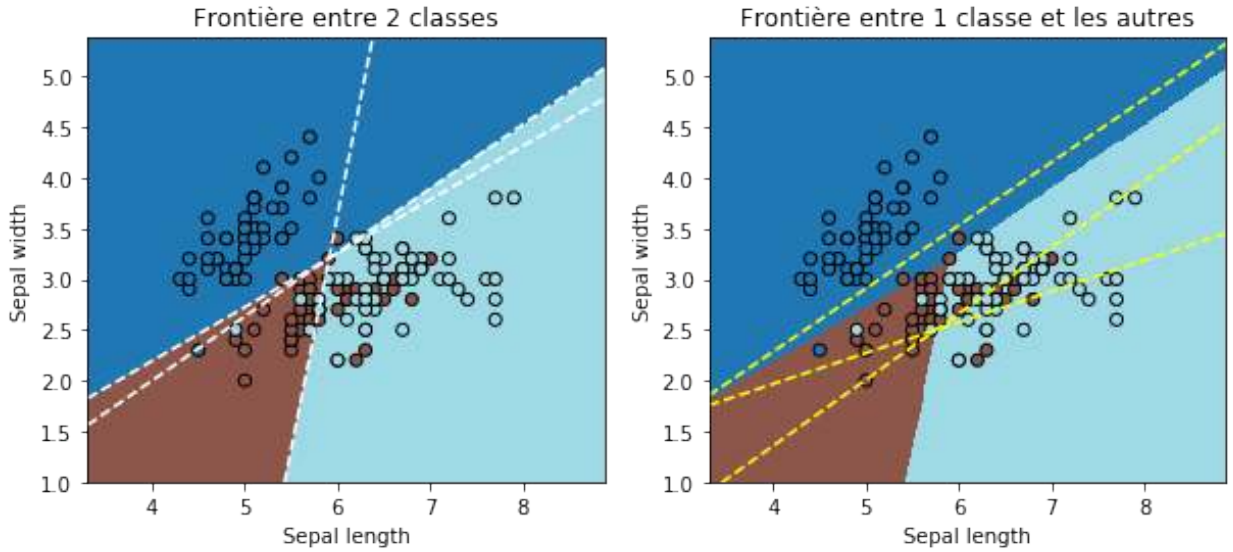
```
array([ 6.34157245, -1.54507432, -4.6206785 ])
```

```
import matplotlib.pyplot as plt

def draw_border(clr, X, y, fct=None, incx=1, incy=1, figsize=None, border=True,␣
↪ax=None):

    # voir https://sashat.me/2017/01/11/list-of-20-simple-distinct-colors/
    # https://matplotlib.org/examples/color/colormaps_reference.html
    _unused_ = ["Red", "Green", "Yellow", "Blue", "Orange", "Purple", "Cyan",
                "Magenta", "Lime", "Pink", "Teal", "Lavender", "Brown", "Beige",
                "Maroon", "Mint", "Olive", "Coral", "Navy", "Grey", "White", "Black"]

    h = .02  # step size in the mesh
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - incx, X[:, 0].max() + incx
    y_min, y_max = X[:, 1].min() - incy, X[:, 1].max() + incy
    xx, yy = numpy.meshgrid(numpy.arange(x_min, x_max, h), numpy.arange(y_min, y_max,␣
↪h))
    if fct is None:
        Z = clr.predict(numpy.c_[xx.ravel(), yy.ravel()])
    else:
        Z = fct(clr, numpy.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    cmap = plt.cm.tab20
    Z = Z.reshape(xx.shape)
    if ax is None:
        fig, ax = plt.subplots(1, 1, figsize=figsize or (4, 3))
    ax.pcolormesh(xx, yy, Z, cmap=cmap)

    # Plot also the training points
    ax.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=cmap)
    ax.set_xlabel('Sepal length')
    ax.set_ylabel('Sepal width')

    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())

    # Draw lines
    x1, x2 = xx.min(), xx.max()
    cl = 0
    if border:
        for i in range(0, clr.coef_.shape[0]):
            for j in range(i+1, clr.coef_.shape[0]):
                delta = clr.coef_[i] - clr.coef_[j]
                db = clr.intercept_[i] - clr.intercept_[j]
                y1 = (-db - delta[0] * x1) / delta[1]
                y2 = (-db - delta[0] * x2) / delta[1]
                ax.plot([x1, x2], [y1, y2], '--', color="white")
                cl += 1
    else:
        for i in range(0, clr.coef_.shape[0]):
            delta = clr.coef_[i]
```

(suite sur la page suivante)

(suite de la page précédente)

```
            db = clr.intercept_[i]
            y1 = (-db - delta[0] * x1) / delta[1]
            y2 = (-db - delta[0] * x2) / delta[1]
            ax.plot([x1, x2], [y1, y2], '--', color="yellow")
            cl += 1

    return ax

fig, ax = plt.subplots(1, 2, figsize=(10,4))
draw_border(clr, X, y, ax=ax[0])
draw_border(clr, X, y, border=False, ax=ax[1])
ax[0].set_title("Frontière entre 2 classes")
ax[1].set_title("Frontière entre 1 classe et les autres");
```

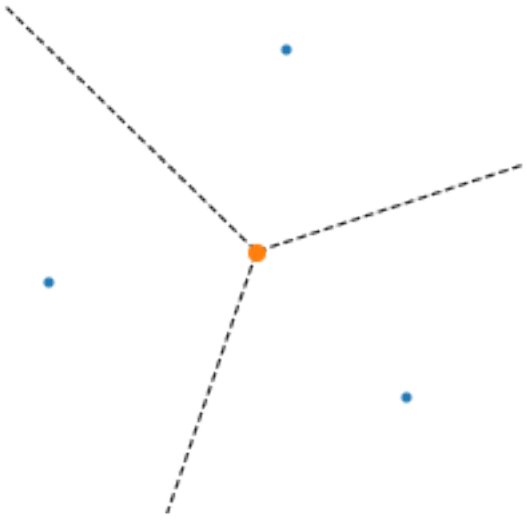


Quelques diagramme de Voronoï

```
points = numpy.array([[1, 2], [3, 4], [4, 1]])
```

```
from scipy.spatial import Voronoi, voronoi_plot_2d
vor = Voronoi(points)
```

```
fig, ax = plt.subplots(figsize=(4,4))
ax.ishold = lambda: True  # bug between scipy and matplotlib 3.0
voronoi_plot_2d(vor, ax=ax)
ax.set_xlim([0, 5])
ax.set_ylim([0, 5])
ax.axis('off');
```



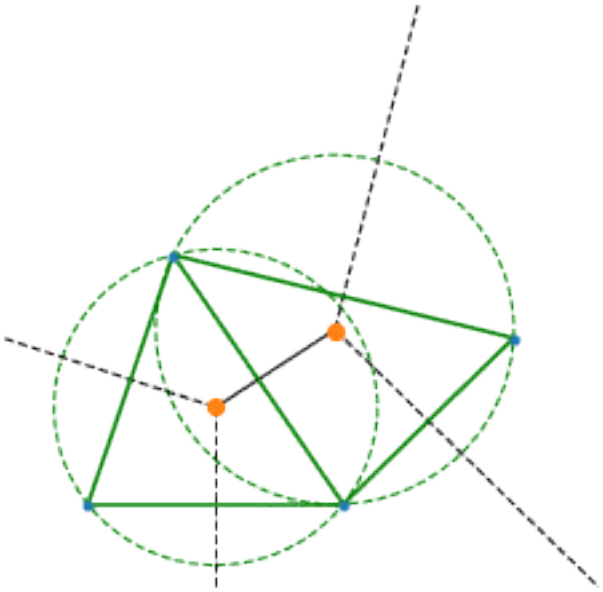
```
vor.point_region
```

```
array([3, 1, 2], dtype=int64)
```

```
vor.vertices
```

```
array([[2.75, 2.25]])
```

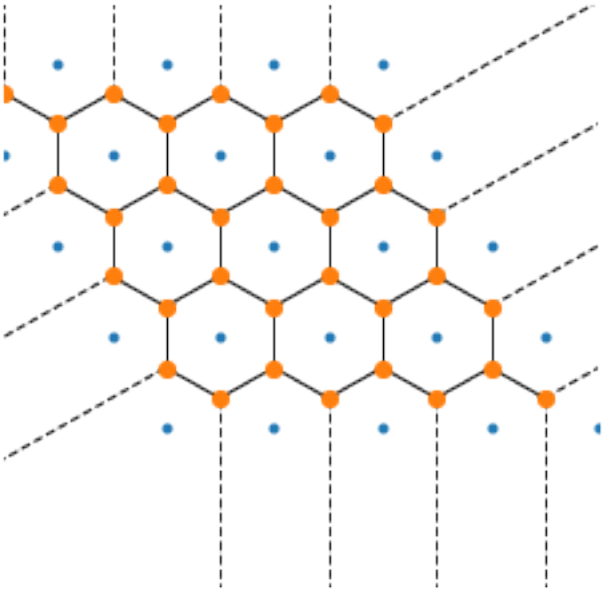
```
from matplotlib.patches import Circle
from matplotlib.collections import PatchCollection
points = numpy.array([[1, 1], [2, 4], [4, 1], [6,3]])
vor = Voronoi(points)
fig, ax = plt.subplots(figsize=(4,4))
cs = []
for i in range(vor.vertices.shape[0]):
    v = vor.vertices[i, :]
    d = (v - points[2, :])
    r = (d.dot(d) ** 0.5)
    circle = Circle((v[0], v[1]), r, fill=False, ls='--', edgecolor='g', visible=True)
    ax.add_artist(circle)
for i in range(points.shape[0]):
    for j in range(i+1, points.shape[0]):
        if i == 0 and j == 3:
            continue
        ax.plot(points[[i, j], 0], points[[i, j], 1], "g-")
ax.ishold = lambda: True  # bug between scipy and matplotlib 3.0
voronoi_plot_2d(vor, ax=ax)
ax.set_xlim([0, 7])
ax.set_ylim([0, 7])
ax.axis('off');
```



```
import math
n = 5
a = math.pi * 2 / 3
points = []
for i in range(n):
    for j in range(n):
        points.append([i + j * math.cos(a), j * math.sin(a)])
points = numpy.array(points)
```

```
vor = Voronoi(points)
```

```
fig, ax = plt.subplots(figsize=(4,4))
ax.ishold = lambda: True  # bug between scipy and matplotlib 3.0
voronoi_plot_2d(vor, ax=ax)
ax.set_xlim([-1.5, 4])
ax.set_ylim([-1.5, 4])
ax.axis('off');
```



Un diagramme de Voronoï proche

On applique la formule définie par Régression logistique, diagramme de Voronoï, k-Means [90] et on résoud le système linéaire défini par :

$$\begin{array}{rl} & \left\{\begin{array}{l} \langle L_i - L_j, P_i\rangle + B_i - B_j = -\{\langle L_i - L_j, P_j\rangle + B_i - B_j\} \\ P_i - P_j - \left\langle P_i - P_j, \frac{L_i-L_j}{\|L_i-L_j\|}\right\rangle \frac{L_i-L_j}{\|L_i-L_j\|} = 0 \end{array}\right. \\ \iff & \left\{\begin{array}{l} \langle L_i - L_j, P_i + P_j\rangle + 2(B_i - B_j) = 0 \\ P_i - P_j - \left\langle P_i - P_j, \frac{L_i-L_j}{\|L_i-L_j\|}\right\rangle \frac{L_i-L_j}{\|L_i-L_j\|} = 0 \end{array}\right. \\ \implies & \left\{\begin{array}{l} \langle L_i - L_j, P_i + P_j\rangle + 2(B_i - B_j) = 0 \\ \langle P_i - P_j, u\rangle - \left\langle P_i - P_j, \frac{L_i-L_j}{\|L_i-L_j\|}\right\rangle \left\langle \frac{L_i-L_j}{\|L_i-L_j\|}, u\right\rangle = 0 \end{array}\right. \end{array}$$

Où u est un vecteur unité quelconque. On cherche à résoudre sous la forme d'un système linéaire $LP = B$ où le vecteur P est l'ensemble des coordonnées de tous les points cherchés. D'après la page cité ci-dessus, dans le cas d'un diagramme à trois classes, ce système a une infinité de solutions.

```
import numpy
matL = []
matB = []
L = clr.coef_
B = clr.intercept_
for i in range(0, L.shape[0]):
    for j in range(i + 1, L.shape[0]):
        li = L[i, :]
        lj = L[j, :]
        c = (li - lj)
        nc = (c.T @ c) ** 0.5

        # condition 1
```

(suite sur la page suivante)

90. http://www.xavierdupre.fr/app/mlstatpy/helpsphinx/c_ml/lr_voronoi.html

(suite de la page précédente)

```
        mat = numpy.zeros((L.shape))
        mat[i,:] = c
        mat[j,:] = c
        d = -2*(B[i] - B[j])
        matB.append(d)
        matL.append(mat.ravel())

        # condition 2 - cache plusieurs équations
        # on ne prend que la première coordonnée
        c /= nc
        c2 = c * c[0]
        mat = numpy.zeros((L.shape))
        mat[i,:] = -c2
        mat[j,:] = c2

        mat[i,0] += 1
        mat[j,0] -= 1
        matB.append(0)
        matL.append(mat.ravel())

matL = numpy.array(matL)
matB = numpy.array(matB)
matL.shape, matB.shape, numpy.linalg.det(matL)
```

```
((6, 6), (6,), 2.0281820935727704e-16)
```

```
import pandas
pandas.DataFrame(matL)
```

Le déterminant est très faible suggérant que la matrice est non inversible et on sait qu'elle l'est dans ce cas. On remplace la dernière équation en forçant la coordonnée d'un point.

```
matL[-1,:] = 0
matL[-1,0] = 1
matB[-1] = 3
numpy.linalg.det(matL)
```

```
42.07770646874508
```

On vérifie que le système linéaire est celui attendu.

```
import pandas
df = pandas.DataFrame(matL)
df['B'] = matB
df
```

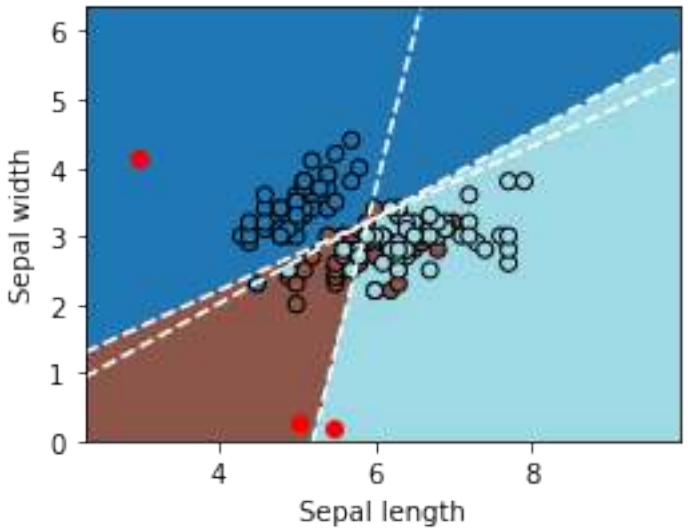
```
from numpy.linalg import inv
points = (inv(matL) @ matB).reshape((3,2))
points
```

```
array([[3.        , 4.12377262],
       [5.03684606, 0.2827372 ],
       [5.48745959, 0.18503334]])
```

```
x = points[0, :]
c1 = (L@x.T).T.ravel() + B
x = points[1, :]
c2 = (L@x.T).T.ravel() + B
x = points[2, :]
c3 = (L@x.T).T.ravel() + B
numpy.vstack([c1,c2,c3])
```

```
array([[  9.86655487,  -4.02070972,  -6.07697098],
       [-10.61997713,   3.26728747,   3.1110941 ],
       [-12.13641872,   3.65091377,   3.80710713]])
```

```
ax = draw_border(clr, X, y, incx=2, incy=2)
ax.plot(points[:, 0], points[:, 1], 'ro');
```



Régression logistique dans un quadrillage

On s'intéresse un problème de régression logistique où le problème est très facile mais pas forcément évident du point de vue d'une régression logistique.

```
Xs = []
Ys = []
n = 20
for i in range(0, 4):
    for j in range(0, 3):
        x1 = numpy.random.rand(n) + i*1.1
        x2 = numpy.random.rand(n) + j*1.1
        Xs.append(numpy.vstack([x1,x2]).T)
        Ys.extend([i*3+j] * n)
X = numpy.vstack(Xs)
Y = numpy.array(Ys)
X.shape, Y.shape
```

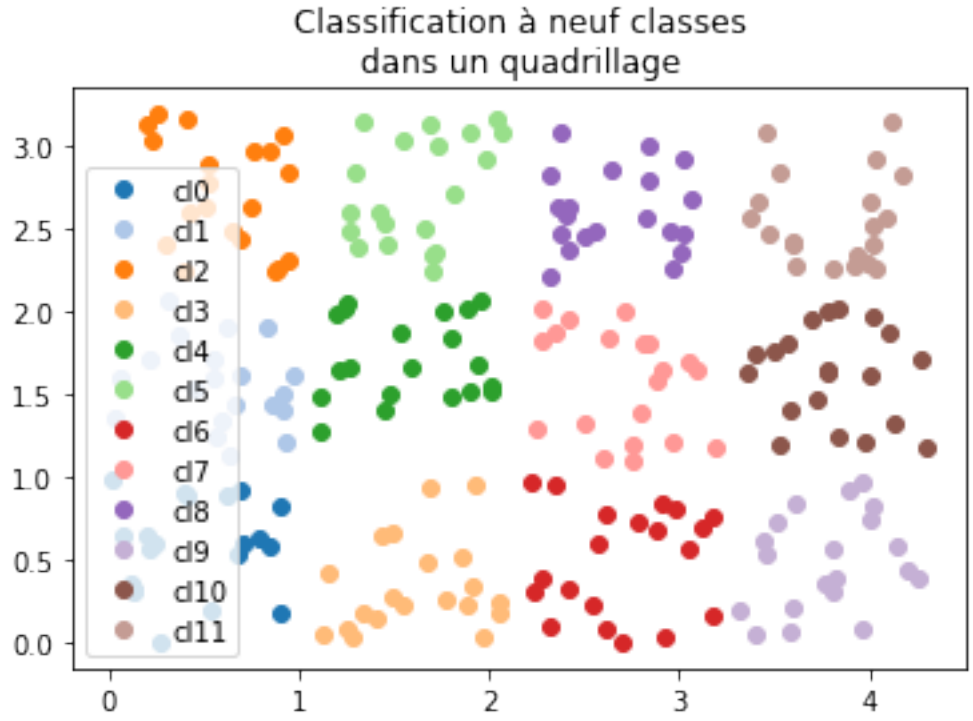
```
((240, 2), (240,))
```

```
set(Y)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

On vérifie que le nuage de points est tel qu'indiqué.

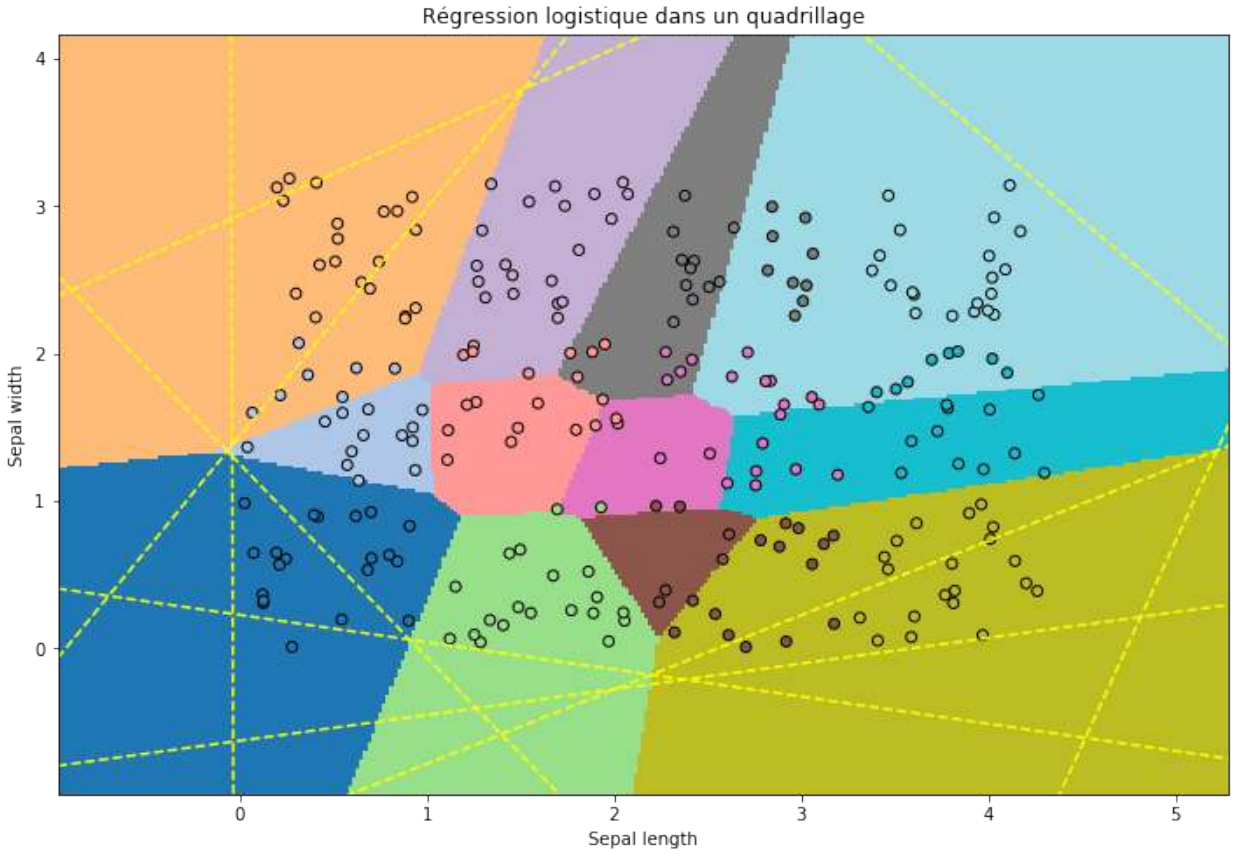
```
fig, ax = plt.subplots(1, 1, figsize=(6,4))
for i in range(0, 12):
    ax.plot(X[Y==i,0], X[Y==i,1], 'o', label="cl%d"%i, color=plt.cm.tab20.colors[i])
ax.legend()
ax.set_title("Classification à neuf classes\ndans un quadrillage");
```




```
from sklearn.linear_model import LogisticRegression
clr = LogisticRegression()
clr.fit(X, Y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
ax = draw_border(clr, X, Y, incx=1, incy=1, figsize=(12,8), border=False)
ax.set_title("Régression logistique dans un quadrillage");
```




```
clr.score(X, Y)
```

```
0.6958333333333333
```

On copie les features en les mettant au carré. Le problème est toujours aussi simple mais la régression logistique a plus de variables non corrélées sur lesquelles s'appuyer.

```
def create_feat(X):
    X2 = X.copy()
    X2[:, 0] = X2[:, 0] * X2[:, 0]
    X2[:, 1] = X2[:, 1] * X2[:, 1]
    XX2 = numpy.hstack([X, X2])
    return XX2

clr2 = LogisticRegression()
clr2.fit(create_feat(X), Y)
```

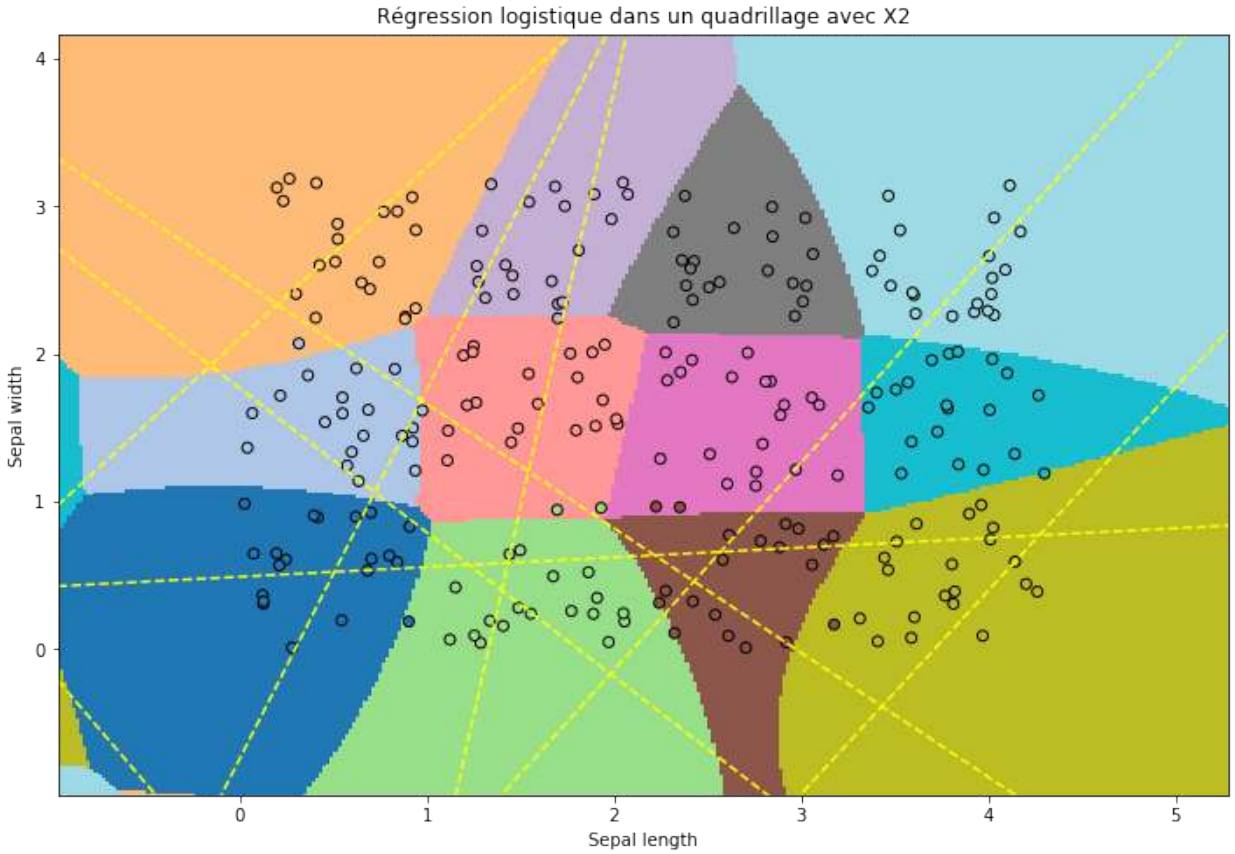
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
def fct_predict(clr, X):
    return clr.predict(create_feat(X))

ax = draw_border(clr2, X, Y, fct=fct_predict, incx=1, incy=1, figsize=(12,8),␣
↪border=False)
```

(suite sur la page suivante)

(suite de la page précédente)

```
ax.set_title("Régression logistique dans un quadrillage avec X2");
```




```
clr2.score(create_feat(X), Y)
```

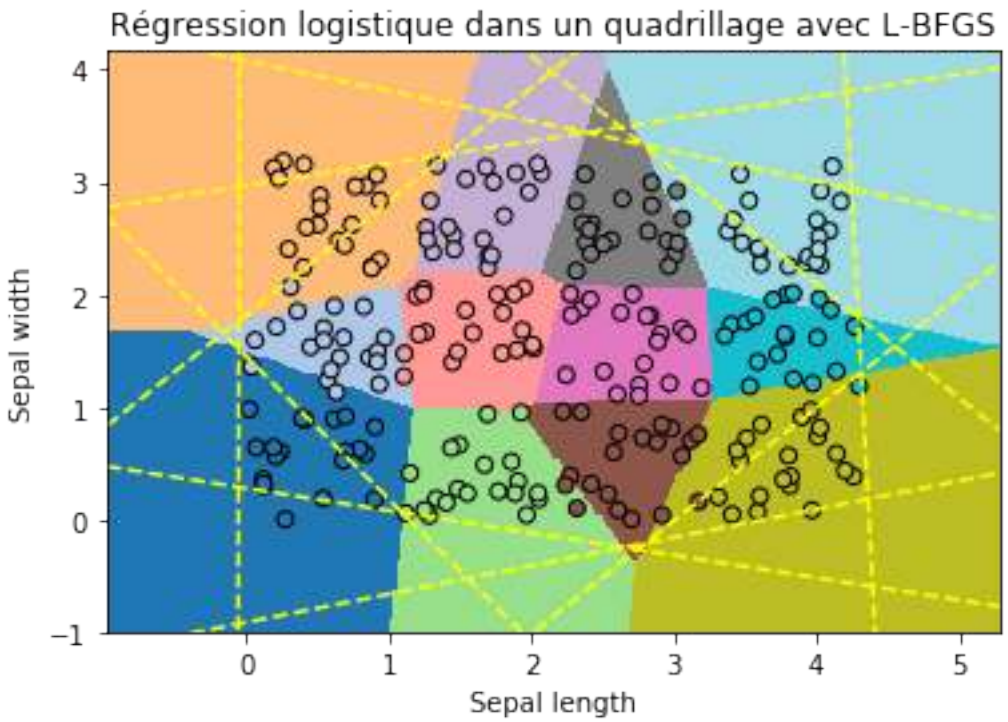
```
0.9583333333333334
```

Du fait que ce problème de classification est équivalent à un diagramme de Voronoï, il a été construit comme tel, le fait que la régression logistique semble être provenir d'un problème de convergence numérique plutôt que du modèle théorique. Pour vérfier on joue avec les paramètres d'apprentissage. Tout d'abord, l'algorithme de descente de gradient.

```
clr_t = LogisticRegression(solver='lbfgs')
clr_t.fit(X, Y)
clr_t.score(X, Y)
```

```
0.9
```

```
ax = draw_border(clr_t, X, Y, incx=1, incy=1, figsize=(6,4), border=False)
ax.set_title("Régression logistique dans un quadrillage avec L-BFGS");
```



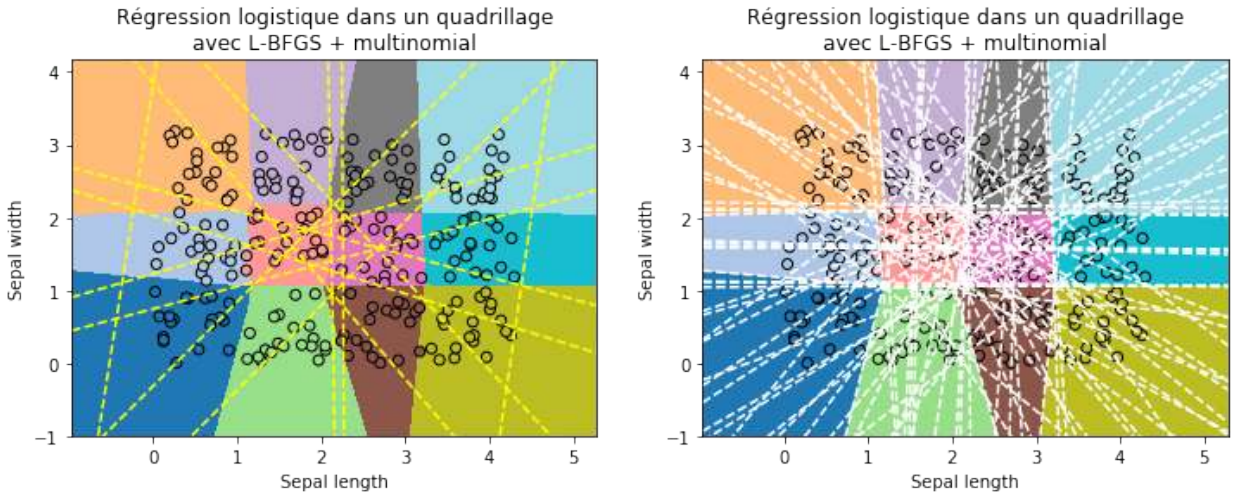
Ensuite, on change la façon de résoudre le problème. Plutôt que de résoudre n problèmes de classifications binaires, on résoud un seul problème avec une erreur de classification égale à la Multinomial logistic regression [91].

```
clr_t = LogisticRegression(solver='lbfgs', multi_class='multinomial')
clr_t.fit(X, Y)
clr_t.score(X, Y)
```

```
0.9875
```

```
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
draw_border(clr_t, X, Y, incx=1, incy=1, figsize=(6,4), border=False, ax=ax[0])
draw_border(clr_t, X, Y, incx=1, incy=1, figsize=(6,4), border=True, ax=ax[1])
ax[0].set_title("Régression logistique dans un quadrillage\navec L-BFGS + multinomial
↪")
ax[1].set_title("Régression logistique dans un quadrillage\navec L-BFGS + multinomial
↪");
```

91. https://en.wikipedia.org/wiki/Multinomial_logistic_regression



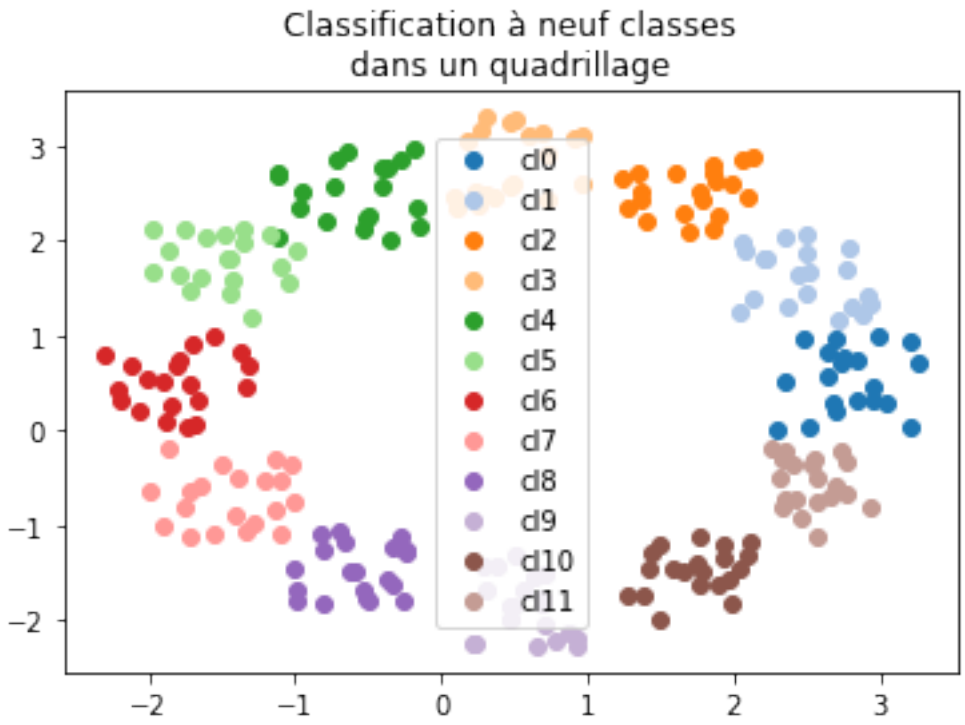
Les frontières entre une classes et les autres n'ont plus l'air d'avoir de signification géométrique. L'approche une classe contre toutes les autres marchent bien si celles-ci ont des frontières convexes sans angles aigus et si elles ne sont pas bornées. En gros, cette approche rapide fonctionne bien si toutes les classes sont disposées autour de la boule unité ou d'une boule unité composée sur un sous-ensemble des dimensions.

Régression logistique autour d'un cercle

```
from math import cos, sin, pi
Xs = []
Ys = []
n = 20
for i in range(0, 12):
    x1 = numpy.random.rand(n) + 2.3*cos(i/ 12. * 2 * pi)
    x2 = numpy.random.rand(n) + 2.3*sin(i/ 12. * 2 * pi)
    Xs.append(numpy.vstack([x1,x2]).T)
    Ys.extend([i] * n)
X = numpy.vstack(Xs)
Y = numpy.array(Ys)
X.shape, Y.shape
```

```
((240, 2), (240,))
```

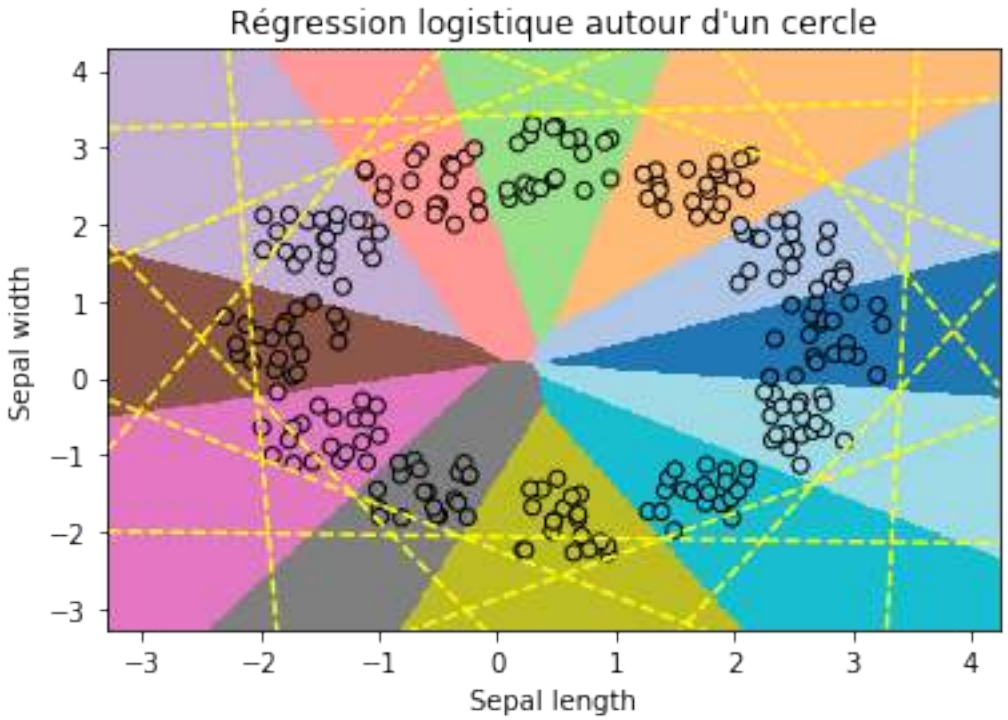
```
fig, ax = plt.subplots(1, 1, figsize=(6,4))
for i in range(0, 12):
    ax.plot(X[Y==i,0], X[Y==i,1], 'o', label="cl%d"%i, color=plt.cm.tab20.colors[i])
ax.legend()
ax.set_title("Classification à neuf classes\ndans un quadrillage");
```




```
clr_c = LogisticRegression()
clr_c.fit(X, Y)
clr_c.score(X, Y)
```

```
0.9833333333333333
```

```
ax = draw_border(clr_c, X, Y, incx=1, incy=1, figsize=(6,4), border=False)
ax.set_title("Régression logistique autour d'un cercle");
```



Rien n'est prouvé, ce ne sont que des observations. On peut se poser la question si le problème précédent n'était pas justement choisi pour montrer que dans un cas, l'approche une classe contre les autres dans le cas d'un quadrillage est particulièrement malvenue. On accroît l'espace entre les classes.

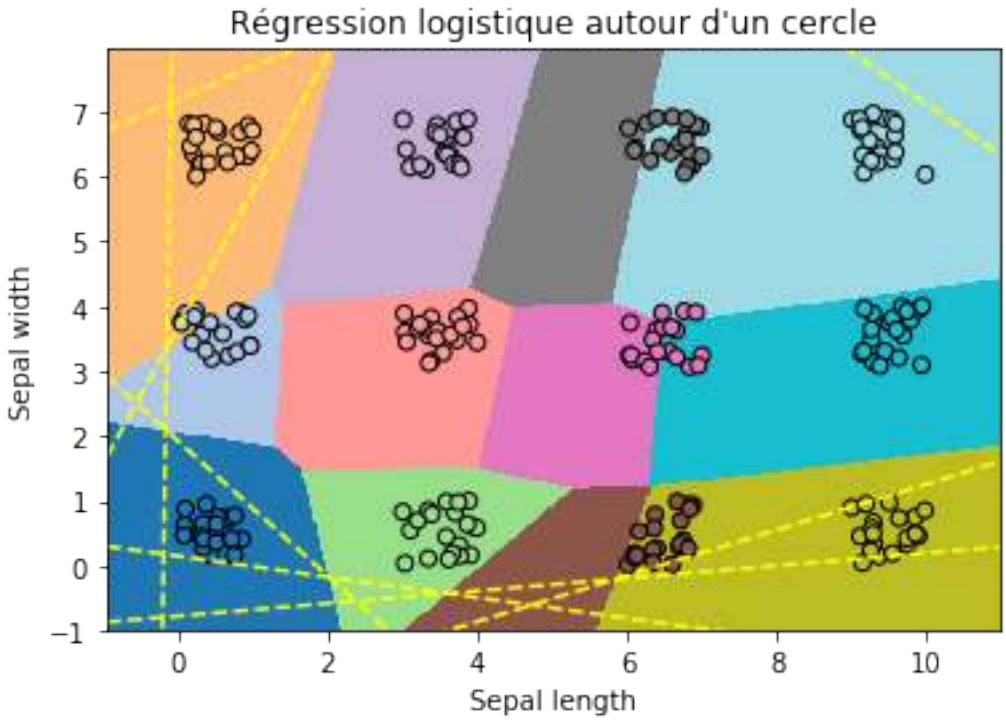
```
Xs = []
Ys = []
n = 20
for i in range(0, 4):
    for j in range(0, 3):
        x1 = numpy.random.rand(n) + i*3
        x2 = numpy.random.rand(n) + j*3
        Xs.append(numpy.vstack([x1,x2]).T)
        Ys.extend([i*3+j] * n)
X = numpy.vstack(Xs)
Y = numpy.array(Ys)
X.shape, Y.shape
```

```
((240, 2), (240,))
```

```
clr_q = LogisticRegression()
clr_q.fit(X, Y)
clr_q.score(X, Y)
```

```
0.7875
```

```
ax = draw_border(clr_q, X, Y, incx=1, incy=1, figsize=(6,4), border=False)
ax.set_title("Régression logistique autour d'un cercle");
```



A priori non mais on préfère l'approche une classe contre les autres car elle est beaucoup plus rapide. L'approche multinomiale requiert de changer d'algorithme de descente de gradient.

```
clr_q = LogisticRegression()
%timeit clr_q.fit(X, Y)
```

```
4.25 ms ± 148 ţs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
clr_qmn = LogisticRegression(multi_class='multinomial', solver='lbfgs')
%timeit clr_qmn.fit(X, Y)
```

```
55.4 ms ± 1.18 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Pousser les classes sur la boule unité

Puisque le modèle est plus facile à apprendre lorsque les classes sont réparties sur la boule unité, l'idéal serait d'avoir une transformation qui le fait, comme d'ajouter des dimensions. La régression logistique ne peut modéliser que des classes convexes. Cela veut dire que le barycentre, sous cette hypothèses, appartient à la zone que le modèle attribute à une classe donnée. On calcule ce barycentre pour toutes les classes et on ajoute comme variables la distance à chacun de ces centres. On reprend le problème du quadrillage.

```
Xs = []
Ys = []
n = 20
for i in range(0, 4):
    for j in range(0, 3):
        x1 = numpy.random.rand(n) + i*1.1
        x2 = numpy.random.rand(n) + j*1.1
        Xs.append(numpy.vstack([x1,x2]).T)
```

(suite sur la page suivante)

(suite de la page précédente)

```
        Ys.extend([i*3+j] * n)
X = numpy.vstack(Xs)
Y = numpy.array(Ys)
X.shape, Y.shape
```

```
((240, 2), (240,))
```

```
bary = []
for i in range(12):
    b = X[Y==i].mean(axis=0)
    bary.append(b)
barys = numpy.vstack(bary)
barys.shape
```

```
(12, 2)
```

```
from sklearn.metrics.pairwise import euclidean_distances
dist = euclidean_distances(X, barys)
dist.shape
```

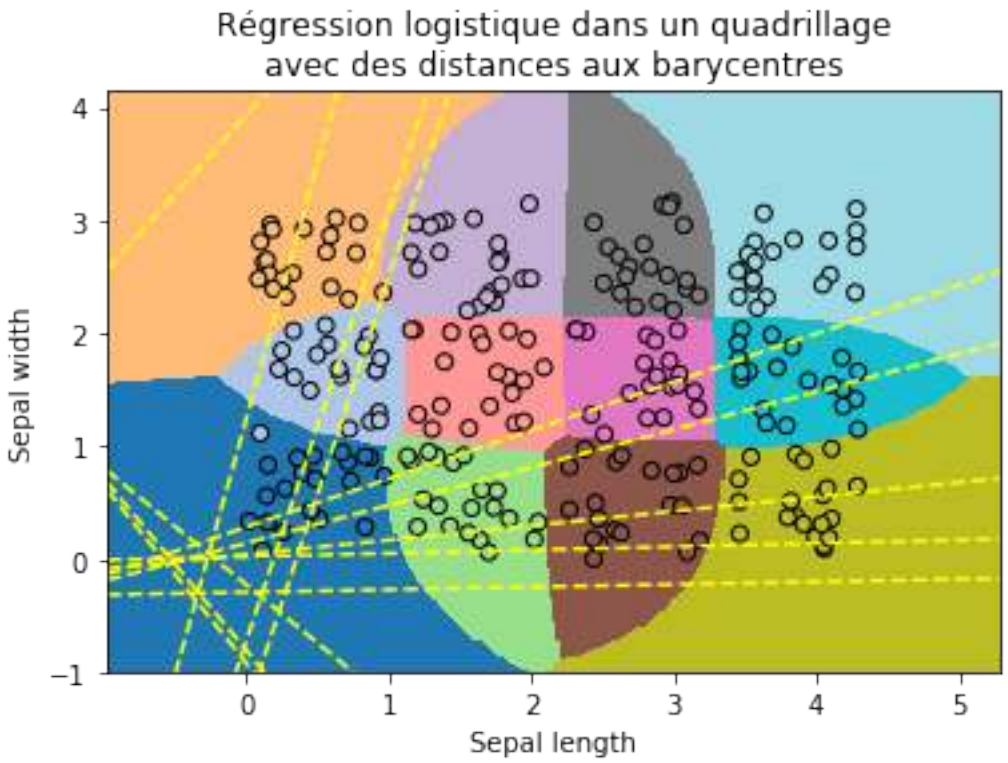
```
(240, 12)
```

```
Xext = numpy.hstack([X, dist])
```

```
clr_ext = LogisticRegression()
clr_ext.fit(Xext, Y)
clr_ext.score(Xext, Y)
```

```
0.9916666666666667
```

```
def fct_predict(clr, X):
    dist = euclidean_distances(X, barys)
    Xext = numpy.hstack([X, dist])
    return clr.predict(Xext)

ax = draw_border(clr_ext, X, Y, fct=fct_predict, incx=1, incy=1, figsize=(6,4),␣
↪border=False)
ax.set_title("Régression logistique dans un quadrillage\navec des distances aux␣
↪barycentres");
```



Cela répond également à une question : **Que faire lorsque les classes ne sont pas convexes ?** Une idée consiste à effectuer un k-means [92] par classe jusqu'à ce que chaque classe soit à peu près converte par un ensemble de cluster appris sur cette classe.

Cas presque hexagonal

Pour tester quelques idées et parce c'est joli. L'idéal serait de se rapprocher d'un pavage de Penrose [93].

```
import math
n = 4
a = math.pi * 2 / 3
points = []
Ys = []
for i in range(n):
    for j in range(n):
        dil = ((i+1)**2 + (j+1)**2) ** 0.6
        for k in range(0,20):
            x = i + j * math.cos(a)
            y = j * math.sin(a)
            points.append([x * dil, y * dil])
            Ys.append(i*n+j)
            mi = 0.5
            for r in [0.1, 0.3, mi]:
                nb = 6 if r == mi else 12
                for k in range(0, nb):
                    x = i + j * math.cos(a) + r * math.cos(math.pi*2/nb * k + math.pi/
↪6)
```

(suite sur la page suivante)

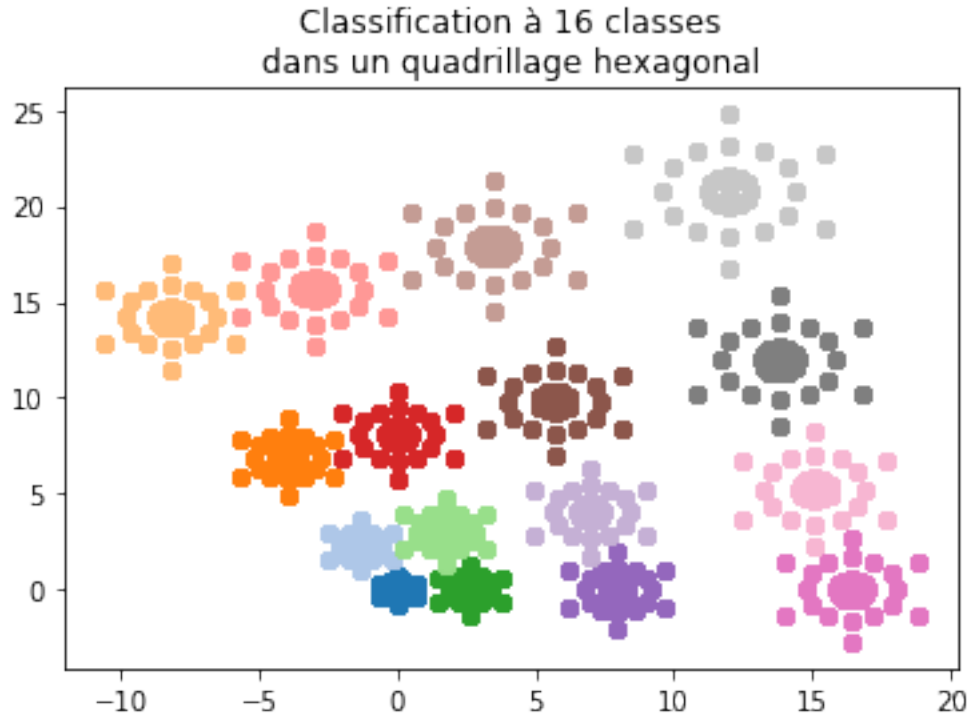
92. http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
93. https://fr.wikipedia.org/wiki/Roger_Penrose

(suite de la page précédente)

```
                    y = j * math.sin(a) + r * math.sin(math.pi*2/nb * k + math.pi/6)
                    points.append([x * dil, y * dil])
                    Ys.append(i*n+j)
X = numpy.array(points)
Y = numpy.array(Ys)
set(Y)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
```

```
fig, ax = plt.subplots(1, 1, figsize=(6,4))
for i in range(0, max(Y)+1):
    ax.plot(X[Y==i,0], X[Y==i,1], 'o', label="cl%d"%i, color=plt.cm.tab20.colors[i
↪%20])
ax.set_title("Classification à 16 classes\ndans un quadrillage hexagonal");
```




```
clr_hex = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=200)
clr_hex.fit(X, Y)
clr_hex.score(X, Y)
```

```
0.9919354838709677
```

```
ax = draw_border(clr_hex, X, Y, incx=1, incy=1, figsize=(6,4), border=False)
ax.set_title("Régression logistique dans\nun quadrillage hexagonal");
```

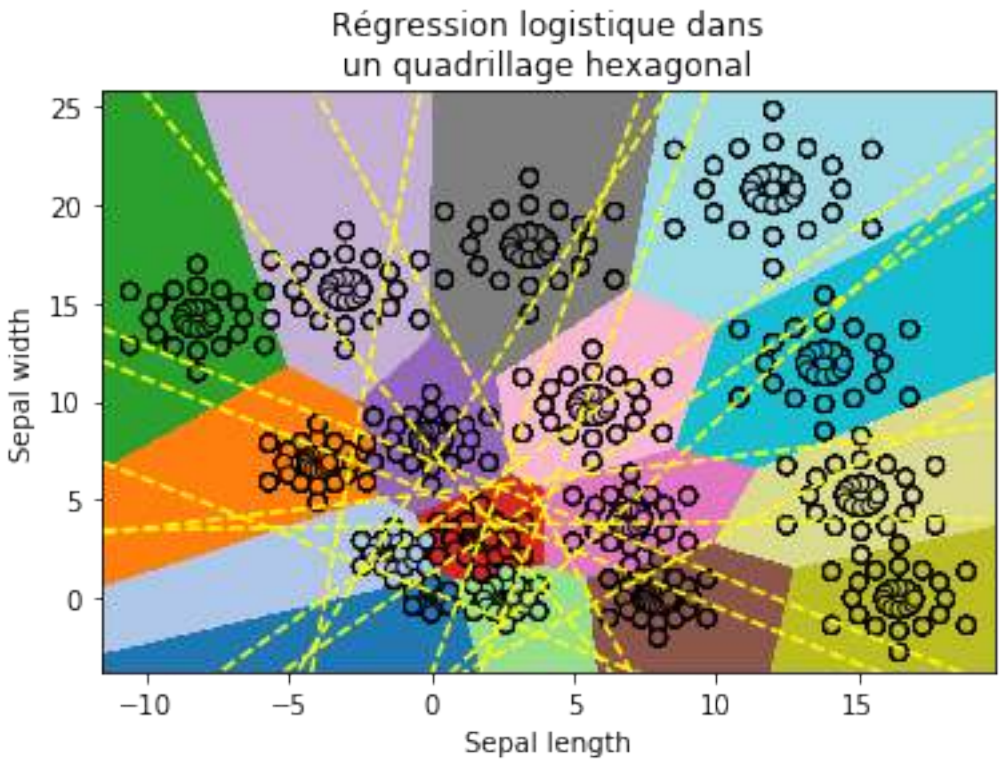


Diagramme de Voronoï approché

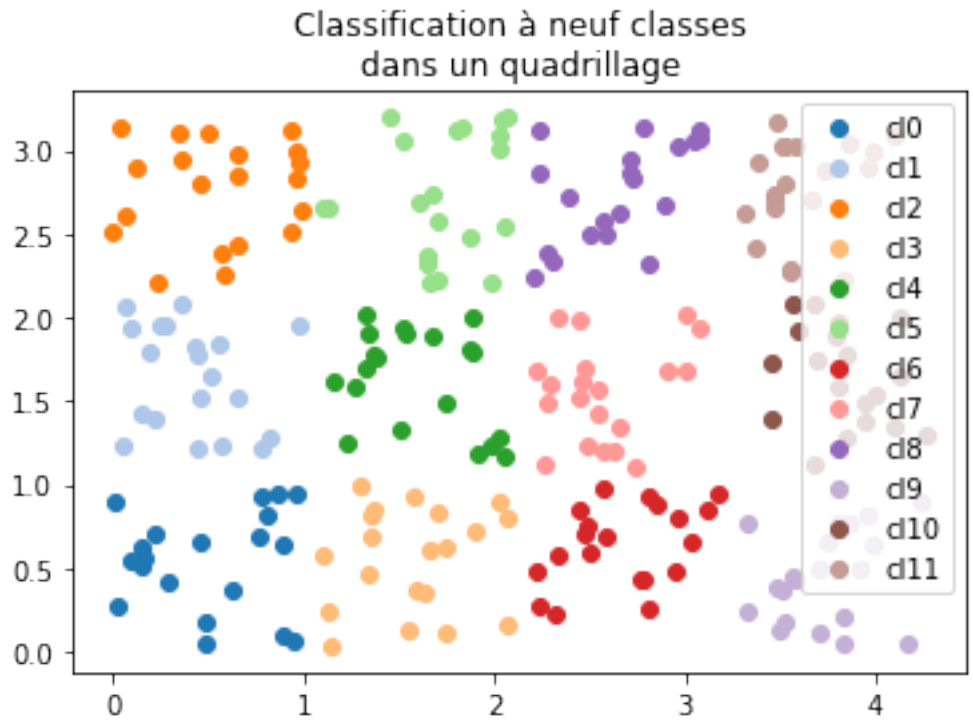
On pousse l'idée implémentée dans le cas de trois classes pour un nombre de classes quelconque. Il n'existe pas de façon générique de diagramme de Voronoï équivalent. On résoud le système linéaire avec une régression quantile et d'autres astuces de calculs à découvrir dans le code de la fonction voronoi_estimation_from_lr[94].

```
Xs = []
Ys = []
n = 20
for i in range(0, 4):
    for j in range(0, 3):
        x1 = numpy.random.rand(n) + i*1.1
        x2 = numpy.random.rand(n) + j*1.1
        Xs.append(numpy.vstack([x1,x2]).T)
        Ys.extend([i*3+j] * n)
X = numpy.vstack(Xs)
Y = numpy.array(Ys)
X.shape, Y.shape
```

```
((240, 2), (240,))
```

```
fig, ax = plt.subplots(1, 1, figsize=(6,4))
for i in range(0, 12):
    ax.plot(X[Y==i,0], X[Y==i,1], 'o', label="cl%d"%i, color=plt.cm.tab20.colors[i])
ax.legend()
ax.set_title("Classification à neuf classes\ndans un quadrillage");
```

94. http://www.xavierdupre.fr/app/mlstatpy/helpsphinx//mlstatpy/ml/voronoi.html




```
from sklearn.linear_model import LogisticRegression
clr = LogisticRegression()
clr.fit(X, Y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
from mlstatpy.ml import voronoi_estimation_from_lr
points = voronoi_estimation_from_lr(clr.coef_, clr.intercept_, max_iter=20,␣
↪verbose=True)
points
```

```
[voronoi_estimation_from_lr] iter=1/20 score=0.0953 tol=3.48e-10 del P2,9 d=3.19
[voronoi_estimation_from_lr] iter=2/20 score=0.0939 tol=3.48e-10 del P1,9 d=2.72
[voronoi_estimation_from_lr] iter=3/20 score=0.089 tol=3.48e-10 del P2,6 d=2.5
[voronoi_estimation_from_lr] iter=4/20 score=0.0892 tol=3.48e-10 del P0,11 d=2.46
[voronoi_estimation_from_lr] iter=5/20 score=0.0894 tol=3.48e-10 del P2,10 d=2.42
[voronoi_estimation_from_lr] iter=6/20 score=0.0882 tol=3.48e-10 del P1,10 d=2.44
[voronoi_estimation_from_lr] iter=7/20 score=0.0889 tol=3.48e-10 del P0,10 d=2.3
[voronoi_estimation_from_lr] iter=8/20 score=0.0877 tol=3.48e-10 del P5,9 d=2.29
[voronoi_estimation_from_lr] iter=9/20 score=0.0869 tol=3.48e-10 del P1,11 d=2.18
[voronoi_estimation_from_lr] iter=10/20 score=0.088 tol=3.48e-10 del P2,3 d=2.2
[voronoi_estimation_from_lr] iter=11/20 score=0.089 tol=3.48e-10 del P0,8 d=2.14
[voronoi_estimation_from_lr] iter=12/20 score=0.0884 tol=3.48e-10 del P1,6 d=2.2
[voronoi_estimation_from_lr] iter=13/20 score=0.0871 tol=3.48e-10 del P2,11 d=2.07
[voronoi_estimation_from_lr] iter=14/20 score=0.0874 tol=3.48e-10 del P0,5 d=2.1
[voronoi_estimation_from_lr] iter=15/20 score=0.0868 tol=3.48e-10 del P0,2 d=2.1
[voronoi_estimation_from_lr] iter=16/20 score=0.087 tol=3.48e-10 del P0,9 d=2.06
```

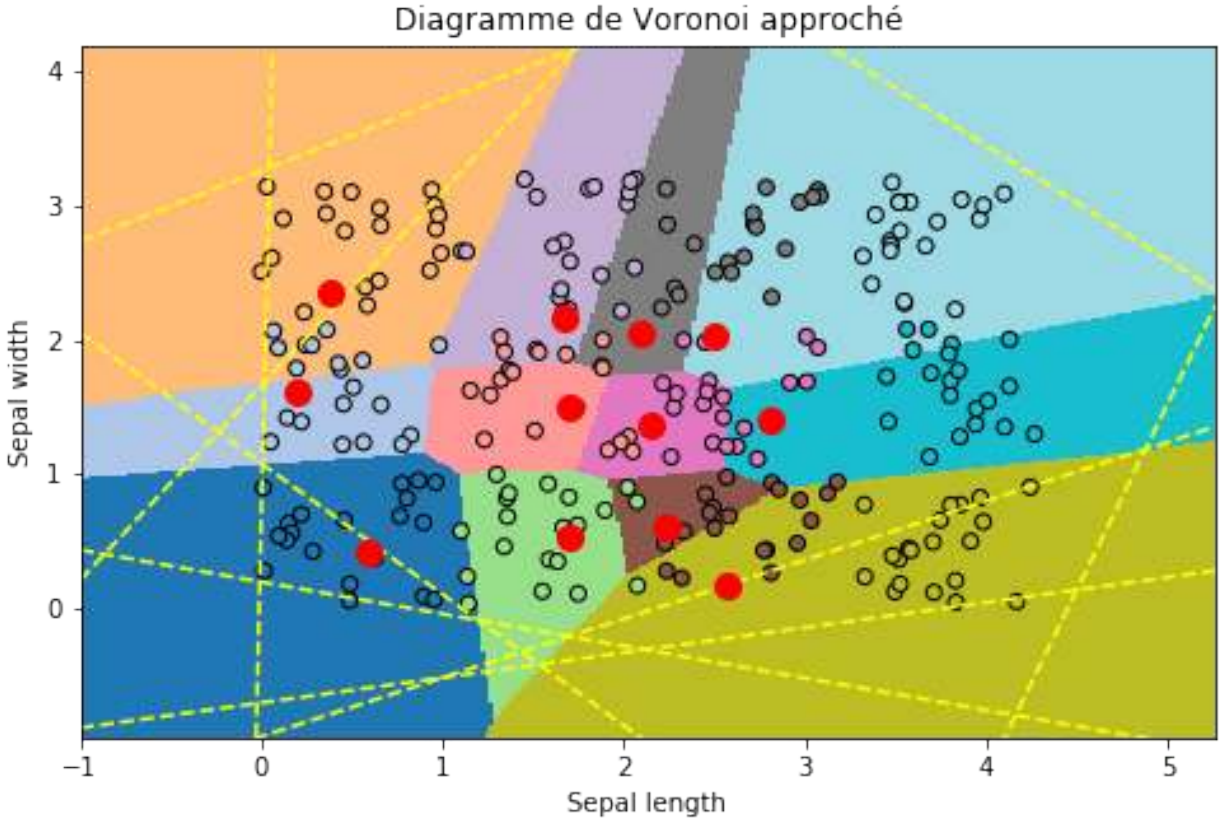
(suite sur la page suivante)

(suite de la page précédente)

```
[voronoi_estimation_from_lr] iter=17/20 score=0.0876 tol=3.48e-10 del P8,9 d=1.99
[voronoi_estimation_from_lr] iter=18/20 score=0.0878 tol=3.48e-10 del P2,7 d=1.93
[voronoi_estimation_from_lr] iter=19/20 score=0.0889 tol=3.48e-10 del P9,11 d=1.93
[voronoi_estimation_from_lr] iter=20/20 score=0.0875 tol=3.48e-10 del P1,7 d=1.97
```

```
array([[0.59042773, 0.41675379],
       [0.19276405, 1.61586254],
       [0.38750542, 2.34848342],
       [1.70510075, 0.5341869 ],
       [1.69940467, 1.50388896],
       [1.66571087, 2.15827251],
       [2.23834543, 0.6114512 ],
       [2.14600591, 1.3636044 ],
       [2.08762755, 2.04091816],
       [2.5732091 , 0.170076  ],
       [2.81087731, 1.40217985],
       [2.49984364, 2.02978587]])
```

```
ax = draw_border(clr, X, Y, incx=1, incy=1, figsize=(8,5), border=False)
ax.plot(points[:, 0], points[:, 1], 'ro', ms=10)
ax.set_title("Diagramme de Voronoi approché");
```



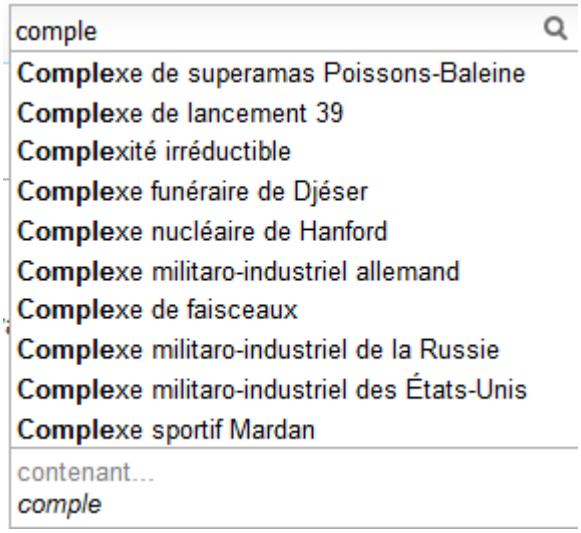
CHAPITRE 4

Natural Language Processing

Ou traitement du langage naturel [95].

4.1 Complétion

La complétion [96] est un méchanisme qui permet à un utilisateur de saisir les mots de sa recherche avec moins de caractères qu'elle n'en contient. L'utilisateur saisit plus rapidement.



Si ces outils sont appréciables du point de vue utilisateurs, ils le sont tout autant côté site web en réduisant la variabilité dans le texte saisie, en particulier les fautes d'orthographes. L'utilisateur a besoin de moins de requêtes pour trouver son produits et cela diminue d'autant la charge du serveur qui lui fournit ses résultats.

Ce chapitre aborde différentes problématiques liées à ce genre de systèmes qui sont présents partout sur Internet, moteurs de recherches, sites de ventes en ligne, journaux... Il existe de nombreuses librairies qui les implémentent. La

95. https://fr.wikipedia.org/wiki/Traitement_automatique_du_langage_naturel
96. https://fr.wikipedia.org/wiki/Compl%C3%A8tement

plus connue en Python est whoosh [97].

Quelques éléments de codes sont disponibles dans le module `completion` et le notebook *Complétion* (page 344). Vous pouvez également lire How to Write a Spelling Corrector [98] de Peter Norvig [99] et découvrir le sujet avec On User Interactions with Query Auto-Completion [100] de Bhaskar Mitra, Milad Shokouhi, Filip Radlinski, Katja Hofmann.

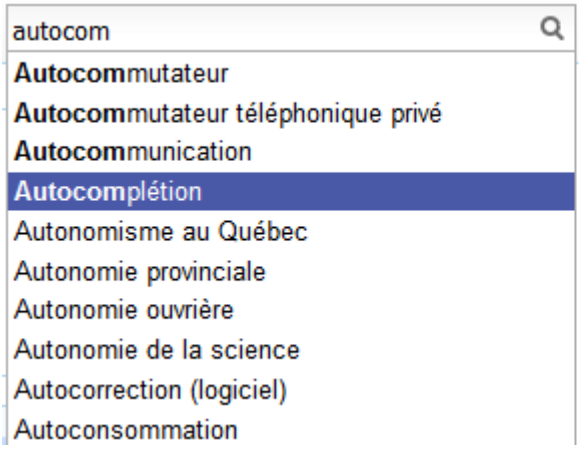
4.1.1 Formalisation

Problème d'optimisation

Je me réfère pour cela à l'article *[Sevenster2013]* (page **??**) (voir aussi *[Bampoulidis2017]* (page **??**)) qui introduit différentes façons de construire un système d'autocompétion et qui les compare à l'usage. Et s'il existe plusieurs façons de faire, il faut d'abord mettre au point une façon de les comparer. Je me place dans le cadre d'un moteur de recherche car c'est l'usage principal, que celui-ci soit un moteur de recherche ou une recherche incluse sur un site de vente. A la fin de la journée, on sait quelles sont les requêtes saisies par les utilisateurs et le nombre de fois qu'elles ont été saisies : (q_i, w_i) pour $i \in [[1, N]]$.

Sans système de complétion, les utilisateurs saisissent donc $K = \sum_{i=1}^{N} l(q_i)w_i$ où $l(q_i)$ est la longueur de la complétion q_i. Avec le système de complétion, les utilisateurs saisissent moins de caractères, c'est ce chiffre là qu'on cherche à minimiser. L'unité est le charactère saisi ou *keystroke* en anglais.

Même avec le même système de complétion, il n'est pas dit que tous les utilisateurs saisissent la même requête de la même façon. Pour simplifier, on va supposer que si malgré tout et ne considérer que la façon minimale de saisir une requête.



L'exemple précédent illustrent deux façons de saisir le terme *autocomplétion* (sur Wikipédia), *autocom* + 4 touches vers le bas ou *autocomp* + 1 touche vers le bas, soit 7+4=11 touches dans le premier cas ou 8+1=9 touches dans le second cas.

Définition D1 : Minimum Keystroke

97. https://whoosh.readthedocs.io/en/latest/
98. http://norvig.com/spell-correct.html
99. http://norvig.com/
100. https://www.semanticscholar.org/paper/On-user-interactions-with-query-auto-completion-Mitra-Shokouhi/71e953caa2542a61b52e684649b3569c00251021/pdf

On définit la façon optimale de saisir une requête sachant un système de complétion S comme étant le minimum obtenu :

$$M(q,S) = \min_{0 \leqslant k \leqslant l(q)} k + K(q,k,S) \tag{4.1}$$

La quantité $K(q,k,S)$ représente le nombre de touche vers le bas qu'il faut taper pour obtenir la chaîne q avec le système de complétion S et les k premières lettres de q.

De façon évidente, $K(q,l(q),S) = 0$ et $M(q,S) \leqslant l(q)$ et $K(q,k,S) > 0$ si $k < l(q)$. On prend également comme convention $\forall q \notin S,\ K(q,k,S) = \infty$ et $\forall q \notin S,\ M(q,S) = l(q)$. Certains systèmes proposent des requêtes avant de saisir quoique ce soit, c'est pourquoi on inclut la valeur $M(q,0)$ qui représente ce cas. Construire un système de complétion revient à minimiser la quantité :

$$M(S) = \sum_{i=1}^{N} M(q_i,S) w_i$$

Ensemble des complétions

Il n'y a pas de restriction sur la fonction $K(q,k,S)$ mais on se limitera dans un premier temps à une fonction simple. On suppose que le système d'autocomplétion dispose d'un ensemble de requêtes ordonnées $S = (s_i)$ et la fonction :

$$K(q,k,S) = position(q, S(q[1..k]))$$

Où $S(q[1..k])$ est le sous-ensemble ordonné de S des complétions qui commencent par les k premières lettres de q et de longueur supérieure strictement à k. $position(q, S(q[1..k]))$ est la position de q dans cet ensemble ordonné ou ∞ si elle n'y est pas. Cette position est strictement positive $K(q,k,S) \geqslant 1$ sauf si $k = l(q)$ auquel cas, elle est nulle. Cela signifie que l'utilisateur doit descendre d'au moins un cran pour sélectionner une complétion. On note $\sigma(q)$ la position de la complétion q dans l'ensemble S. Par construction, $s \not\succ s_2 \Longrightarrow \sigma(s_1) \neq \sigma(s_2)$.

$$K(q,k,S) = \#\left\{i | s_i \succ q[1..k], s_i \in S, \sigma(s_i) < \sigma(q)\right\} \tag{4.2}$$

$\#$ désigne le cardinal de l'ensemble. Trouver le meilleur système de complétion S revient à trouver la meilleure fonction $K(q,k,S)$ et dans le cas restreint l'ordre sur S qui minimise cette fonction. Le plus souvent, on se contente de trier les complétions par ordre décroissant de popularité. On considérera par la suite qu'on est dans ce cas.

Gain

On définit le gain en keystroke comme étant le nombre de caractères saisis en moins :

$$G(q,S) = l(s) - M(q,S)$$

Minimier $M(S)$ ou maximiser $G(S) = \sum_{i=1}^{N} G(q_i,S) w_i$ revient au même.

$$G(S) = \sum_{i=1}^{N} w_i (l(s) - M(q,S)) = \sum_{i=1}^{N} w_i l(s) - \sum_{i=1}^{N} w_i M(q,S)) = K - M(S)$$

Où $K = \sum_{i=1}^{N} l(q_i) w_i$ l'ensemble des caractères tapés par les utilisateurs. $\frac{G(S)}{K}$ est en quelque sorte le ratio de caractères économisés par le système de complétion.

4.1.2 Fausses idées reçues

Il faut trier les complétions par fréquence décroissante

En pratique, cela marche plutôt bien. En théorie, cette assertion est fausse. Prenons les quatre complétions suivantes :

q	fréquence	ordre
a	4	1
ab	3	2
abc	2	3
abcd	1	4

Dans cet exemple, si l'utilisateur tape `ab`, il verra les complétions :

```
abc
abcd
```

Dans tous les cas, $K(q, k, S) = l(q) - k$. Cela veut dire que l'utilisateur ne gagnera rien. En revanche, avec l'ordre suivant :

q	ordre
a	4
ab	2
abc	3
abcd	1

Si l'utilisateur tape `ab`, il verra les complétions :

```
abcd
abc
```

Le nombre de caractères économisés sera :

q	fréquence	ordre	$M(q, S)$
a	4	4	1
ab	3	2	2
abc	2	3	3
abcd	1	1	1 = $K(q, 0, S)$

D'où un gain total de $G(S) = 3$.

Il faut placer les complétions courtes avant

Le cas précédent est déjà un contre exemple. Mais d'un point de vue utilisateur, il n'est pas facile de lire des complétions de longueurs différentes. Cela veut peut-être dire aussi que la métrique considérée pour choisir le meilleur système de complétion est faux. Cela sera discuté à la prochaine section.

Il faut montrer toutes les complétions

Le premier exemple offre aussi un contre exemple. Dans cet exemple, l'ensemble $Q = (q_i)$ des requêtes utilisateurs et l'ensemble $S = (s_i)$ des **complétions** ou **requêtes complètes** est le même. Il suffit de la modifier un peu. On enlève la requête *ab* de S.

q	fréquence	ordre	$M(q, S)$
a	4	1	1
ab	3	∞	2
abc	2	2	2
abcd	1	3	3

D'où un gain total de $G(S) = 2$. En conclusion, si j'enlève une petite complétion pour laquelle le gain est nul, il est possible que le gain pour les suivantes soit positif. On en retient qu'il ne faut pas montrer trop de complétions qui se distinguent d'un caractère. Plus généralement, il ne sert à rien de montrer des complétions plus longue que le préfixe d'un caractère. Par extension, si une complétion est plus longue que le préfixe de d caractères, il faut la montrer à une position inférieure à d.

Et si le poids de chaque complétion est uniforme

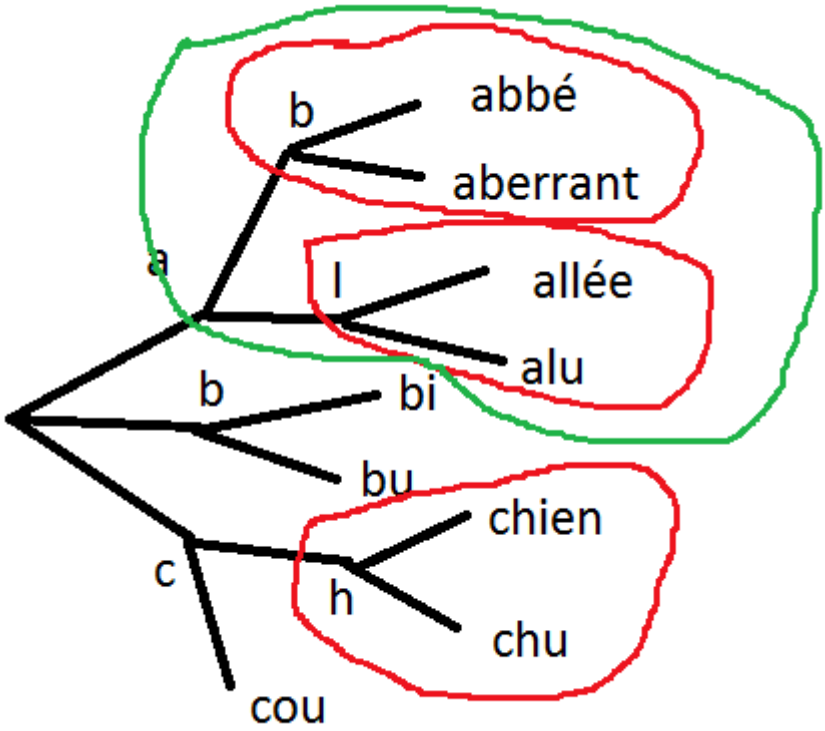
On suppose que les complétions ont toutes le même poids $w_i = 1$. Dans quel ordre faut-il ranger les complétions pour économiser le plus de caractères. On aurait tendance à dire la plus longue d'abord ce qu'on peut vérifier dans le notebook *Complétion* (page 344).

q	fréquence	ordre	$M(q, S)$
a	1	4	1
ab	1	3	2
abc	1	2	2
abcd	1	1	1

Ajouter deux autres complétions disjointes *edf*, *edfh*. Le gain maximum est 6 et il y a plusieurs ordres :

```
'edf', 'edfh', 'abc', 'abcd', 'a', 'ab'
'abcd', 'abc', 'edfh', 'edf', 'ab', 'a'
...
```

On a presque l'impression qu'on peut traiter chaque bloc séparément *a, ab, abc, abcd* d'un côté et *edf, edfh* de l'autre. A l'intérieur des blocs, les règles seront les mêmes.



En résumé, si on connaît le meilleur ordre pour toutes les mots sur les noeuds terminaux dans les bulles rouges, dans la bulle verte, le meilleur ordre sera une fusion des deux listes ordonnées.

Quelques essais sur le notebook ont tendance à montrer que l'ordre a peu d'impact sur le résultat final lorsque les complétions ont le même poids. Avec quatre mots, la somme des gains est identique quelque soit l'ordre.

```
p=poids g=gain

20.0 - actuellement p=1.0 g=11.0 | acte p=1.0 g=2.0 | actes p=1.0 g=2.0 | actualité␣
↪p=1.0 g=5.0
20.0 - acte p=1.0 g=3.0 | actuellement p=1.0 g=10.0 | actualité p=1.0 g=6.0 | actes␣
↪p=1.0 g=1.0
20.0 - acte p=1.0 g=3.0 | actes p=1.0 g=3.0 | actualité p=1.0 g=6.0 | actuellement␣
↪p=1.0 g=8.0
```

Mais si on change le poids de l'une d'elles, elle se retrouve en première position.

```
19.2 - actes p=2.0 g=4.0 | actuellement p=1.0 g=10.0 | acte p=1.0 g=1.0 | actualité␣
↪p=1.0 g=5.0
19.2 - actes p=2.0 g=4.0 | actuellement p=1.0 g=10.0 | actualité p=1.0 g=6.0 | acte␣
↪p=1.0 g=0.0
```

4.1.3 Nouvelle métrique

Intuitions

1. La métrique actuelle n'est pas la meilleure.
2. Si les mots n'ont pas de long préfixes en commun, il vaut mieux placer le mot le plus fréquent en première position. Pour les mots de fréquence identique, l'ordre a peu d'importance.
3. S'il existe une séquence de mots emboîtés, les gains sont minimes à moins d'enlever des mots ou de poser les grandes complétions d'abord.

Les intuitions 2 et 3 seront sans doute remise en question en considérant une nouvelle métrique. On considère l'ensemble des complétions S composé de deux mots *actuellement*, *actualité*. Le gain moyen par mots est de 9 caractères économisés. Si on ajoute le grand préfixe commun à la liste *actu*, ce gain moyen tombe à 6.33 (voir *Complétion* (page 344)) quelque soit l'ordre choisi pour les complétions. Toutefois, si on ne prend pas en compte le gain sur le mot *actu* car ce n'est pas un mot correct mais plus un mot qui aide la lecture de la liste, ce gain moyen tombe à 8 seulement. En conclusion, si l'utilisateur tape la lettre **a** et qu'on lui montre ceci :

```
actu
actualité
actuellement
```

Au lieu de :

```
actualité
actuellement
```

Il doit taper en moyenne un caractère de plus pour obtenir le mot qu'il cherche. Et la métrique ne montre pas réellement de préférence pour l'ordre d'affichage des complétions. Pourtant, l'utilisateur pourrait très bien utiliser la séquence de touches suivantes :

touche	mot composé
a	a
bas	actu (complétion)
e	actue
bas	actuellement

Dans cet exemple aussi petit, on ne gagnerait pas grand-chose mais cela vaut le coup d'étudier cette piste pour des vocabulaires plus grand : se servir des préfixes commun comme *tremplin* pour les mots plus grand. L'effect position perdrait un peu de son influence.

Formalisation

On reprend la première métrique *(1)* (page **??**) :

$$M(q,S) \quad = \quad \min_{0 \leqslant k \leqslant l(q)} k + K(q,k,S)$$

La fonction $K(q,k,S)$ est définie par *(2)* (page **??**).

Définition D1 : Dynamic Minimum Keystroke

On définit la façon optimale de saisir une requête sachant un système de complétion S comme étant le minimum obtenu :

$$M'(q,S) \quad = \quad \min_{0 \leqslant k < l(q)} \{M'(q[1..k],S) + \min(K(q,k,S), l(q)-k)\}$$

On prend comme convention $M'(\emptyset, S) = 0$. Le calcul de la métrique se construit comme une suite qui part des chaînes les plus courtes aux plus longues. La métrique est donc bien définie. Contrairement à la première métrique, le calcul dépend du résultat pour tous les préfixes d'une complétion.

---

propriété P1 : métriques

$\forall q,\ M'(q, S) \leqslant M(q, S)$

Si $q \notin S$, c'est évident puisque $M'(q, S) \leqslant M'(\emptyset, S) + l(q)$. Si $q \in S$, cela découle de la constation précédente puisque : $M'(q, S) \leqslant M'(q[[1..k]], S) + K(q, k, S) \leqslant k + K(q, k, S)$.

Quelques résultats

On considère la liste des mots `actuellement`, `actualité`, `actuel`. On compare les ordres qui maximisent la première et la seconde métriques ainsi que le gain obtenu. Première métrique

```
7.0 - actuellement p=1.0 g=11.0 | actuel p=1.0 g=4.0 | actualité p=1.0 g=6.0
7.0 - actuellement p=1.0 g=11.0 | actualité p=1.0 g=7.0 | actuel p=1.0 g=3.0
7.0 - actuel p=1.0 g=5.0 | actuellement p=1.0 g=10.0 | actualité p=1.0 g=6.0
```

Seconde métrique

```
7.333 - actuel p=1.0 g=5.0 | actualité p=1.0 g=7.0 | actuellement p=1.0 g=10.0
7.0 - actuellement p=1.0 g=11.0 | actuel p=1.0 g=4.0 | actualité p=1.0 g=6.0
```

On note que la seconde métrique propose un meilleur gain, ce qui est attendu mais aussi que le mot *actuel* sera placé devant le mot *actuellement*, plus long sans que cela souffre d'ambiguïté.

Définition avancée

Dans les faits, le Dynamic Minimum Keystroke sous-estime le nombre de caractères nécessaires. Lorsqu'on utilise un mot comme tremplin, on peut aisément le compléter mais il faut presser une touche ou attendre un peu pour voir les nouvelles complétions associées à la première complétion choisie et maintenant considéré comme préfixe. C'est ce que prend en compte la définition suivante.

Définition D2 : Dynamic Minimum Keystroke modifié

On définit la façon optimale de saisir une requête sachant un système de complétion S comme étant le minimum obtenu :

$$M"(q, S) \quad = \quad \min \left\{ \begin{array}{l} \min_{1 \leqslant k \leqslant l(q)} \{M"(q[1..k-1], S) + 1 + \min(K(q, k, S), l(q) - k)\} \\ \min_{0 \leqslant k \leqslant l(q)} \{M"(q[1..k], S) + \delta + \min(K(q, k, S), l(q) - k)\} \end{array} \right.$$

Si on prend comme exemple la requête *machine learning*, le premier cas correspond à la séquence :

- — sélection de la complétion *machine*
- — pression de la touche espace
- — sélection de la complétion *machine learning*

Et le second cas à la séquence :

- — sélection de la complétion *machine*
- — pression de la touche droite pour afficher les nouvelles complétions
- — sélection de la complétion *machine learning*

Le coût de la pression de la touche droite est noté $\delta \leqslant 1$ qu'on prendra inférieur à 1. On remarque également qu'avec cette nouvelle métrique, il est possible de diminuer le nombre minimum de touches à presser pour des requêtes en dehors de l'ensemble S à partir du moment où elles prolongent une complétion existante. C'est là un point très intéressant de cette métrique. De manière évidente, $\forall q,\ M'(q, S) \leqslant M"(q, S)$.

Questions

Grâce à cette métrique, on peut envisager de trouver des réponses à certaines questions :

1. Les différences entre les trois métriques sont-elles négligeables ou non ?
2. Ajouter des complétions non présentes dans le corpus améliore-t-elle la métrique ? Même question pour la suppression ?
3. Existe-t-il un moyen de construire de façon itérative l'ensemble des complétions ou plutôt l'ordre qui minimise la métrice $M'(q, S)$?
4. Comment calculer rapidement les métriques pour les requêtes dans l'ensemble S et en dehors ?

Pour la première question, une expérience devrait donner une piste à défaut d'y répondre. Pour la seconde, il n'est pas nécessaire d'envisager la suppression de complétions car celles-ci devraient naturellement se positionner en fin de liste. L'ajout correspond à la situation où beaucoup de complétions partagent le même préfixe sans pour autant que ce préfixe fasse partie de la liste des complétions.

```
macérer
maline
machinerie
machinerie infernale
machinerie infernalissime
machine artistique
machine automatique
machine chaplin
machine intelligente
machine learning
```

L'idée consiste à ajouter la complétion *machine* qui sert de préfixe commun à beaucoup de complétions et cela améliore le gain moyen dans le cas présent (sans compter le gain sur la requête *machine*). Enfin, la troisième et la quatrième question, la réponse requiert la démonstration de quelques propriétés mathématiques. Mais avant j'ajouterai que la première métrique M correspond à la ligne de commande Windows et la métrique M' correspond à la ligne de commande Linux.

4.1.4 Propriétés mathématiques

On s'intéresse principalement à la métrique M' définie par Dynamic Minimum Keystroke mais les résultats seront étendues aux autres quand cela est possible.

Calcul pour une complétion

On a besoin d'une propriété pour calculer élégamment les métriques pour l'ensemble des complétions.

Lemme L1 : Dynamic Minimum Keystroke

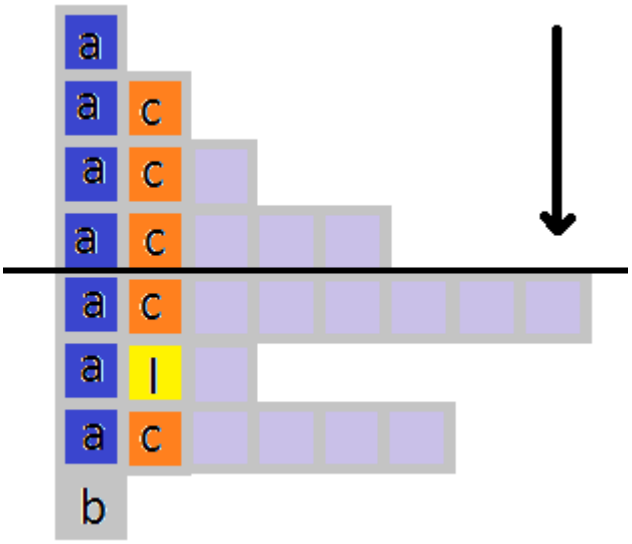
On note $d(q, S)$ la longueur du plus long préfixe de q inclus dans S.

$$d(q, S) = \max \{l(p) | p \prec q,\ p \in S,\ p \neq q\}$$

$$M'(q,S) \quad = \quad \min_{d(q,S) \leqslant k < l(q)} \{M'(q[1..k],S) + \min(K(q,k,S), l(q)-k)\}$$

Il n'est pas nécessaire de regarder tous les préfixes mais seulement ceux entre le plus long préfixe qui est aussi une complétion et la requête q. La démonstration est identique à la démonstration du lemme qui suit. L'idée de cette propriété est de pouvoir réduire le coût de l'algorithme de calcul des métriques. Ce n'est pas la seule écriture qu'on puisse en fait.

Le calcul de la métrique M' suggère qu'on doit faire ce calcul dans le sens des préfixes croissants mais il serait plus simple de le faire dans le sens des complétions de poids croissant (les complétions de moindre poids sont toujours affichées avant).



Si l'algorithme est plus simple (sens de la fléche dans le figure précédente), il faut parfois plusieurs itérations pour obtenir la valeur finale.

Calcul pour une requête en dehors

Mais il est faux de dire que pour deux requêtes en dehors de l'ensemble des complétions, $q_1 \preceq q_2 \Longrightarrow M'(q_1,S) \leqslant M'(q_2,S)$. Le lemme suivant précise pourquoi

Lemme L2 : calcul de *M¿(q, S)*

On suppose que $p(q,S)$ est la complétion la plus longue de l'ensemble S qui commence q :

$$\begin{array}{rcl} k^* & = & \max \{k | q[[1..k]] \prec q \text{ et } q \in S\} \\ p(q,S) & = & q[[1..k^*]] \end{array}$$

La métrique $M'(q,S)$ vérifie la propriété suivante :

$$M'(q,S) = M'(p(q,S),S) + l(q) - l(p(q,S))$$

La métrique $M'(q,S)$ est égale à celle du plus long préfixe inclus dans l'ensemble les complétions à laquelle on ajoute la différence des longueurs. Cela correspond aux caractères que l'utilisateur doit saisir. La démonstration est assez

simple. On suppose que cela n'est pas vrai et qu'il existe un existe $k < k^*$ tel que :

$$\begin{aligned} & M'(q[[1..k]], S) + l(q) - l(q[[1..k]]) < M'(q[[1..k^*]], S) + l(q) - l(q[[1..k^*]]) \\ \Longrightarrow \quad & M'(q[[1..k]], S) - k < M'(q[[1..k^*]], S) - k^* \\ \Longrightarrow \quad & M'(q[[1..k]], S) + (k^* - k) < M'(q[[1..k^*]], S) \end{aligned}$$

Cela signifie qu'on a réussi une façon plus efficace d'écrire le préfixe $q[[1..k^*]]$. Or par définition $M'(q[[1..k^*]], S)$ est censée être le nombre de caractères minimal pour obtenir $q[[1..k^*]]$. Ce n'est donc pas possible. Cette propriété est importante puisque pour calculer $M'(q[[1..k^*]], S)$, il suffit de regarder le plus long préfixe appartenant à l'ensemble des complétions et seulement celui-ci. Cet algorithme et implémenté par la méthode `enumerate_test_metric`. En ce qui concerne la métrique M, par définition $\forall q \notin S,\ M(q, S) = 0$. La métrique M" m'évoque la côte anglaise [101]. L'itération n fonctionne de la même manière à partir du moment où la requête considérée ne fait pas partie de l'ensemble des complétions mais il y a l'étage d'en dessous qui pose un doute. Il y a un brin de poésie dans ce +1. L'application de l'implémentation du calcul de la métrique montre que M' et M" sont très souvent égales. Je vais laisser ce δ sous forme de poésie pour le moment.

A faire : terminer la démonstration pour *M*

La côte anglaise.

Complétions emboîtées

On considère les complétions suivantes :

```
actu
actualité
actualités
actuel
actuellement
```

Pour le préfixe *actue*, on suggère *actuel* at *actuellement*. Pour le préfixe *actua*, on suggère *actualité* at *actualités*. Pour le préfixe *actu*, on suggère la concaténation de ces deux listes. Par conséquent, pour construire les listes de complétions associées à chaque préfixe, il paraît de partir des feuilles de l'arbre puis de fusionner les listes de complétions jusqu'au noeud racine. Plus concrètement, si deux complétions vérifie $q_1 \preceq q_2$ alors l'ensemble des complétions vérifie $C(q_1) \supset C(q_2)$. On peut même dire que : $C(q) = \cup \{C(s) | s \succ q \in S\}$. Cela signifie qu'une fois qu'on a construit un trie représentant l'ensemble des complétions, il suffit de partir des feuilles de l'arbre jusqu'à la racine pour construire la liste des complétions à chaque étape et que pour un noeud précis, la liste des complétions est l'union des listes de complétions des noeuds fils.

Listes tronquées de complétions

On reprend la première métrique *(1)* (page **??**) qui utilise la fonction $K(q, k, S)$ définie en *(2)* (page **??**).

$$M(q, S) \quad = \quad \min_{0 \leqslant k \leqslant l(q)} k + K(q, k, S)$$

Etant donné que le nombre minimum de caractères pour obtenir une complétion dans le trie ne peut pas être supérieur à la longueur, si $K(q, k, S) > l(q) - k$, on sait déjà que que le préfixe $q[1..k]$ ne sera pas le minimum. Cette remarque est applicable aux métriques M' et M".

101. https://www.youtube.com/watch?v=YV54e3R-rLg

4.1.5 Problème d'optimisation

Enoncé 1

Problème P1 : Optimiser un système de complétion

On suppose que l'ensemble des complétions $C = \{c_j\}$ est connu. On souhaite ordonner cet ensemble pour obtenir l'ensemble ordonné des complétions $S = (s_i)$ qu'on considère comme une permutation σ de l'ensemble de départ : $S(\sigma) = (s_i) = (c_{\sigma(j)})$. Ce système de complétion est destiné à un des utilisateurs qui forment des recherches ou requêtes $Q = (q_i, w_i)_{1 \leqslant i \leqslant N_Q}$. q_i est la requête, w_i est la fréquence associée à cette requête. On définit l'effort demandé aux utilisateurs par ce système de complétion :

$$E(C, Q, \sigma) = \sum_{i=1}^{N_Q} w_i M'(q_i, S(\sigma))$$

Déterminer le meilleur système de complétion revient à trouver la permutation σ qui minimise $E(C, Q, \sigma)$.

La métrique M' peut être remplacée par M". La différence peut paraître insignifiante mais elle ne l'est pas tant que ça. Le système de complétion peut se concevoir comme une *compression* : le système de complétion permet de coder l'ensemble des recherches d'un utilisateur Q avec moins de caractères que celui-ci en a besoin pour les taper. On ajoute les caractères $rightarrow$ et $\downarrow$ aux lettres de l'alphabet et cela permet de coder plus rapidement une requêtes. La quantité suivante peut être considérée comme le taux de compression :

$$t(C, Q, \sigma) = \frac{E(C, Q, \sigma)}{\sum_{i=1}^{N_Q} w_i l(q_i)}$$

L'idée derrière cette métaphore est le fait d'avoir une idée de la borne inférieure pour ce taux de compression. On s'inspirer de la complexité de Lempel-Ziv [102] (calculating Lempel-Ziv (LZ) complexity (aka sequence complexity) of a binary string [103]) ou du codage de Huffman [104]. M' permet une compression avec perte et M" sans perte. Le calcul de M' autorise deux *jumps* de suite :

$$abb(\downarrow\downarrow\downarrow\downarrow\downarrow)$$

Mais les deux dernières touches $\downarrow$ peuvent s'appliquer au premier préfixe ou aux suggestions montrées par la complétion obtenue après trois $\downarrow$.

$$abb(\downarrow\downarrow\downarrow)(\downarrow\downarrow)$$

La métrique M" interdit ce cas.

Enoncé 2

Problème P2 : Optimiser un système de complétion filtré

102. https://fr.wikipedia.org/wiki/Complexit%C3%A9_de_Lempel-Ziv
103. http://stackoverflow.com/questions/4946695/calculating-lempel-ziv-lz-complexity-aka-sequence-complexity-of-a-binary-str
104. https://fr.wikipedia.org/wiki/Codage_de_Huffman

On suppose que l'ensemble des complétions $C = \{c_j\}$ est connu. On souhaite ordonner cet ensemble pour obtenir l'ensemble ordonné des complétions $S = (s_i)$ qu'on considère comme une permutation σ de l'ensemble de départ : $S(\sigma) = (s_i) = (c_{\sigma(j)})$. On utilise aussi une fonction f qui filtre les suggestions montrées à l'utilisateur, elle ne change pas l'ordre mais peut cacher certaines suggestions si elles ne sont pas pertinentes. Ce système de complétion est destiné à un des utilisateurs qui forment des recherches ou requêtes $Q = (q_i, w_i)_{1 \leqslant i \leqslant N_Q}$. q_i est la requête, w_i est la fréquence associée à cette requête. On définit l'effort demandé aux utilisateurs par ce système de complétion :

$$E(C, Q, \sigma, f) = \sum_{i=1}^{N_Q} w_i M'(q_i, S(\sigma), f)$$

Déterminer le meilleur système de complétion revient à trouver la permutation σ qui minimise $E(C, Q, \sigma, f)$.

Comme suggéré au paragraphe *Il faut montrer toutes les complétions* (page 115), le filtre f peut rejetter une suggestion si elle est montrée à une position qui ne permet aucun gain à l'utilisateur, c'est-à-dire que la différence des longueurs complétion - préfixe est plus petite que la position où elle est montrée.

Une idée

On aimerait bien pouvoir trouver l'ordre optimal par morceau, supposer que l'ordre optimal pour l'ensemble des complétions correspond à l'ordre des complétions sur un sous-ensemble partageant le même préfixe.

Lemme L1 : M¿et sous-ensemble

On suppose que la complétion q est préfixe pour la requête q' et $\sigma(q) < \sigma(q')$ ce qui signifie que la complétion q est toujours affichée avant la complétion q' si elles apparaissent ensemble. Alors $M'(q, S) < M'(q', S)$. Plus spécifiquement, si on considère l'ensemble $S'(q) = \{s - q \in S | q \prec s\}$ ($s - q$ est la complétion s sans son préfixe q).

$$M'(q', S) = M'(q' - q, S') + M'(q, S)$$

On sait déjà, par construction que $M'(q', S) \leqslant M'(q' - q, S') + M'(q, S)$. Par l'absurde, on suppose que $M'(q', S) < M'(q' - q, S') + M'(q, S)$, comme la requête $q - q'$ est toujours affichée avant la requête q', cela voudrait dire qu'on aurait trouvé une façon plus optimale d'écrire la requête $q - q'$ avec le système S ce qui impossible d'après la définition de la métrique M'. Cette propriété n'aide pas forcmément à trouver un algorithme pour optimiser l'ordre des complétions dans la mesure où la propriété suppose qu'une complétion soit affiché avant toutes celles dont elle est le préfixe. La propriété suivante est évidemment vraie pour le cas particulier qu'on vient de mentionner. Si elle est vraie, cela devrait permettre de procéder par sous-ensemble pour trouver l'ordre optimal.

Théorème T1 : M¿, ordre et sous-ensemble

Soit q une requête de l'ensemble de complétion S ordonnées selon $sigma$. Si cet ordre vérifie :

$$\forall k,\ \sigma(q[1..k]) \leqslant \sigma(q[1..k+1]) \tag{4.3}$$

On note l'ensemble $S'(q[1..k]) = \{q[k+1..len(q)] \in S\}$:

alors :

$$\forall k,\ M'(q[1..k], S) = M'(q[k+1..l(q)], S'(q[1..k]) + M'(q[1..k], S)$$

Ceci découle de l'application du lemme précédent. Ce théorème permet presque de déterminer le meilleur ordre *sigma* parmi ceux qui vérifie la contrainte *(1)* (page **??**), à savoir une requête courte est toujours affichée avant celles qui la complètent. On procède par récurrence, on suppose connu les ordres $\sigma(q)$ pour l'ensemble des complétions qui commencent par le préfixe $p = q[1..k]$, $S'(q[1..k]) = \{q | q[1..k] = p, q \in S\}$. Pour $i = k - 1$, le meilleur ordre : σ revient à fusionner les listes ordonnées obtenues pour chaque préfixe de longueur k. Il faut démontrer la possibilité de traiter les complétions par ordre croissant.

4.1.6 Implémentation

J'allais vous raconter en détail ce qu'est un trie [105] et le paragraphe suivant vous en dira sans doute un peu plus à ce sujet. Le trie [106] est le moyen le plus efficace de trouver un mot aléatoire ou un préfixe aléatoire dans une liste. Mais il y a mieux et plus simple dans notre cas où il faut trouver une longue liste de mots connue à l'avance - donc pas aléatoire -. Et puis, c'était sous mes yeux. Il y a plus simple et aussi efficace quand les listes des mots et des complétions sont connues à l'avance.

Notion de trie

Une implémentation des tries est décrites dans deux notebooks : Arbre et Trie [107]. Les résultats de ce chapitre ont été produits avec le module `completion` et le notebook *Complétion* (page 344). Le notebook *Completion profiling* (page 333) montre les résultats du profiling. L'implémentation Python est très gourmande en mémoire et elle serait plus efficace en C++.

utilisation ou recherche

C'est différent de construire toutes les complétions pour un préfixe plutôt que toutes les complétions pour tous les préfixes. Le premier cas correspond à un utilisateur qui cherche quelque chose. Il faut être rapide quitte à retourner un résultat tronqué.

Le second cas correspond à objectif de recherche des d'optimisation. Les enjeux sont plus de réussir à calculer toutes les complétions en un temps raisonnable et avec une utilisation mémoire raisonnable également.

mémoire

D'après la remarque précédente, il n'est pas utile de conserver pour un préfixe donné l'intégralité des complétions qui commencent par ce préfixe. Dans le pire des cas, cette liste a besoin de contenir autant de complétions que le nombre de caractères de la plus longue complétioms.

Algorithme élégant

Il faut relire le premier problème d¿*optimisation* (page 122) pour commencer à se poser la question : comment calculer la quantité $E(C, C, \sigma)$ lorsque σ correspond à l'ordre alphabétique ? La réponse est simple : il suffit de parcourir les complétions une et une seule fois. Supposons qu'au cours de ce parcours, on est à la complétion d'indice i. On conserve un compteur $p(k, i) = K(c(i), k, C)$ qui représente la position de la complétion $c(i)$ dans la liste des complétions affichées par le système de complétion pour le préfixe $c(i)[[1..k]]$. Le coût de l'algorithme est en $O(N \ln N + LN)$ où N est le nombre de complétions et L la longueur maximale d'une complétion.

Dans le cas où σ est quelconque et $C \neq Q$, on procède en deux étapes. Dans un premier temps, on utilise une variante de l'algorithme précédent pour calculer $M'(q, C)$ pour les requêtes q dans l'ensemble des complétions.

Dans un second temps, on effectue une sorte de fusion entre les deux listes triées alphabétiquement. Le coût de l'algorithme est en $O(ILN + 2N \ln N + M \ln M + max(N, M))$ où M est le nombre de requêtes dans l'ensemble Q. Cette partie repose sur le *lemme* (page 120) lié au calcul des métriques pour les réquêtes hors de l'ensemble des complétions. I est un nombre d'itération nécessaires pour que les métriques M' convergent pour l'ensemble des complétions. En pratique, c'est très petit.

L'algorithme est implémenté dans le module `completion_simple` et plus particulièrement la fonction `CompletionSystem.compute_metrics`.

105. https://fr.wikipedia.org/wiki/Trie_(informatique)
106. https://fr.wikipedia.org/wiki/Trie_(informatique)
107. http://www.xavierdupre.fr/app/ensae_teaching_cs/helpsphinx3/notebooks/_gs1a_A_arbre_trie.html

4.1.7 Digressions

Synonymes, Contexte

On utilise dabord les préfixes pour chercher les mots dans un trie mails il est tout à fait possible de considérer des synonymes. Avec les préfixes, un noeud a au plus 27 (26 lettres + espaces) caractères suivant possibles. Si le préfixe a des synonymes, rien n'empêche de relier ce noeud avec les successeurs de ses synonymes. A ce sujet, voir Context-Sensitive Query Auto-Completion [108], de Ziv Bar-Yossef et Naama Kraus.

Source

Dans le cas d'un moteur de recherche, le trie ou l'ensemble S des requêtes complètes est construit à partir des requêtes des utilisateurs. Lorsque le système de complétion est mise en place, la distribution des requêtes changent. Les requêtes les plus utilisées vont être encore plus utilisées car les utilisateurs vont moins s'égarer en chemin comme s'égarer vers une faute d'orthographe. Comment corriger la distribution des requêtes malgré l'intervention du système de complétion ? Cela pourrait faire l'objet d'un sujet de recherche.

Fonction de gain

Jusqu'à présent, on a considéré uniquement le nombre de caractères économisés pour déterminer le meilleur ordre. Rien n'empêche d'ajouter une coût supplémenaires lié à l'ordre des complétions. Une requête est pénalisée si les complétions associées sont loin de l'ordre alphabétique. On peut pénaliser un ordre éloigné à chaque caractère ajouté.

Minuscules, majuscules

C'est bien connu, on fait peu de ces des accents sur internet. De fait, même si l'accent apparaît à l'écran, le système de complétion verra peut de différences entre le `e` et `é`. Sur Wikpédia, les homonymes sont distingués par un sous-titre entre parenthèse l'année pour un événement sportif régulier. On peut imaginer que plusieurs séquences de caractères aboutissent à la même entrée.

Suppression de caractères

Nous pourrions considérer le fait de pouvoir supprimer des caractères afin de trouver le chemmin le plus court pour obtenir une requête.

108. http://technion.ac.il/~nkraus/papers/fr332-bar-yossef.pdf

Coût d'un caractère

Jusqu'à présent, la pression d'une touche a le même coût quelque soit la source, un caractère, une touche vers le bas. Pourtant, plus il y a de lettres dans l'alphabet, plus le système de complétion sera performant à supposer que les mots soient plus ou moins équirépartis selon les caractères (la probabilité du prochain caractère est uniforme). On peut concevoir que chercher une touche lorsque l'alphabet est grand peut prendre un certain temps. Le cas du chinois est intéressant car la saisie des caractères [109] peut prendre plusieurs touches. Faut-il considérer un caractère chinois comme unité de décomposition d'un mot où la séquence des touches qui le construisent ? Dans le premier cas, il faudrait sans doute pénaliser la saisie d'un caractère en fonction du nombre de touches nécessaires pour le former par rapport à la sélection d'une complétion.

Complétion partielle

On rappelle la métrique *(1)* (page **??**) (voir aussi *(2)* (page **??**)).

$$M'(q,S) \quad = \quad \min_{0 \leqslant k \leqslant l(q)} \{M'(q[1..k],S) + K(q,k,S)\}$$

Si on note $L(p,S)$ l'ensemble des complétions pour le préfixe p. Que dire de la définition suivante ?

$$M'_p(q,S) \quad = \quad \min_{0 \leqslant k \leqslant l(q)} \left\{ \begin{array}{l} \mathbb{1}\!\!1_{\{L(q[1..k],S) \neq \emptyset\}} \left[M'_p(q[1..k],S) + K(q,k,S)\right] + \\ \mathbb{1}\!\!1_{\{L(q[1..k],S) = \emptyset\}} \left[\min_j M'_p(q[1..j],S) + M'_p(q[j+1..],S)\right] \end{array} \right\}$$

Cela revient à considérer que si le système de complétion ne propose aucune complétion avec le préfixe en cours, on propose des complétions avec un préfixe qui ne tient compte que des dernières lettres.

Utilisateurs

La modélisation mathématique aboutit à l'optimisation d'une métrique qui ne coïncide pas forcément à la logique de l'utilisateur. Le fait de montrer une suggestion dans la liste donne en quelques sortes un signal qui indique que cette requête a une bonne d'aboutir. C'est en particulier vrai si elle est identique à celle que l'utilisateur est en train de saisir quand même même elle n'aurait aucun sens. La présence d'une suggestion conforte l'utilisateur dans le chemin qu'il choisit.

Un autre aspect est le fait qu'une suggestion, la première par exemple, doit rester à l'écran si celle-ci inclut le préfixe saisit par l'utilisateur. Supposons que ce dernier saisisse *fac*, le moteur de suggestions lui montrera en premier *facebook*. Il faudra que *facebook* demeure en première position jusqu'à ce l'utilisateur ait soit cliqué dessus, soit saisi quelque chose de plus long. Enlever la suggestion *facebook* alors que le préfixe en cours est *faceboo* perturbe.

Revenir en arrière

On pourrait également imaginer une métrique qui calcule le meilleur chemin en donnant la possibilité de revenir en arrière. Par exemple, pour saisir *éléphant*, un utilisateur pourrait utiliser la suggestion *éléphants* puis enlever le dernier *s*. On note la métrique M'_b.

Définition D1 : Dynamic Minimum Keystroke arrière

On définit la façon optimale de saisir une requête sachant un système de complétion S comme étant le minimum obtenu :

$$M'_b(q,S) \quad = \quad \min \left\{ \begin{array}{l} \min_{0 \leqslant k < l(q)} \{M'_b(q[1..k],S) + \min(K(q,k,S), l(q)-k)\} \\ \min_{s \succ q} \{M'_b(s,S) + l(s) - l(q)\} \end{array} \right\}$$

---

109. https://fr.wikipedia.org/wiki/Saisie_du_chinois_sur_ordinateur

Le second terme de la métrique nécessite de considérer toutes les suggestions dans S qui commencent par le préfixe q. Le calcul de la métrique peut être fait en adaptant l'algorithme reprend les idées du paragraphe *Formalisation* (page 117) mais il faudra parcourir les suggestions dans l'ordre alphabétique puis mettre à jour la métrique en les parcourant dans l'autre sens. On recommence jusqu'à convergence.

Notebooks associés :

— *Complétion* (page 344)
— *Completion profiling* (page 333)
— *Completion Trie and metrics* (page 328)
— *Complétion Simple* (page 350)

CHAPITRE 5

Métriques

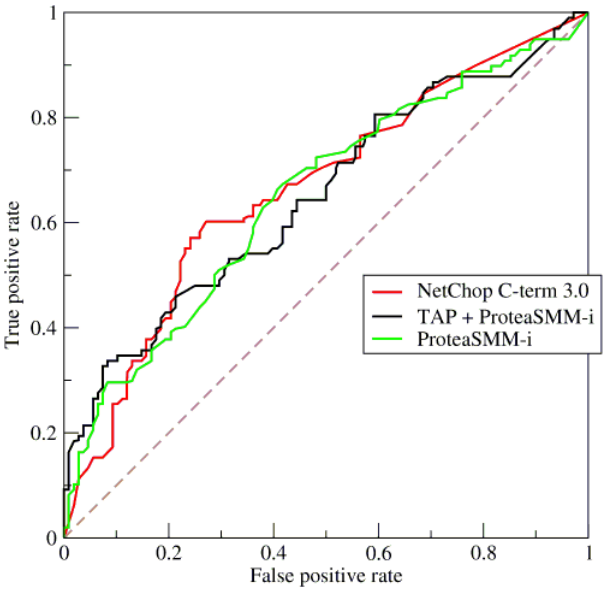
5.1 Courbe ROC

Ce document introduit la courbe ROC [110] (Receiving Operator Characteristic) qui est communément utilisée pour mesurer la performance d'un classifieur. Il introduit aussi des termes comme précision, rappel, AUC [111], qui sont présents dans la plupart des articles qui traitent de machine learning. Le module `roc` implémente les calculs ci-dessous qu'on peut tester avec le notebook *ROC* (page 304).

5.1.1 Définitions

Supposons que nous avons un classifieur qui classe des observations en un ensemble de classes. De plus, il donne cette réponse accompagnée d'un score de pertinence. Deux cas sont possibles : soit la réponse est bonne (1), soit la réponse est fausse (0). Pour chaque observation, on associe un couple (r, x) où r est égal à 0 ou 1. x est le score de pertinence. On cherche à déterminer à partir de quel seuil de pertinence, la réponse du classifieuur est fiable. En faisant varier x, on obtient une courbe (source : wikipedia [112]) :

110. https://en.wikipedia.org/wiki/Receiver_operating_characteristic
111. https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve
112. http://en.wikipedia.org/wiki/File:Roccurves.png



Cette courbe sert également à comparer différents classifieurs. Plus une courbe a des valeurs élevées, plus l'aire sous la courbe est grande, moins le classifieur fait d'erreur.

D'une manière simplifiée, le classifieur retourne une réponse qui est soit mauvaise (-) soit bonne (+). On peut l'évaluer car pour construire un classifier on dispose toujours d'une base contenant les réponses attendues. En fonction du score x et d'un seuil s, on définit quatre cas :

cas	réponse prédite est bonne (+)	réponse prédite est mauvaise (-)
$x \geqslant s$	TP : vrai (true) positif	FP : faux positif
$x < s$	TN : vrai (true) négatif	FN : faux négatif

Ces résultats sont souvent présentés selon une matrice confusion :

réponse attendue	0	1
0	TN	FP
1	FN	TP

A partir de ces définitions, on définit :

— la précision [113] : $\frac{TP}{TP+FP}$

— le rappel ou recall [114] : $\frac{TP}{TP+TN}$

En choisissant un seuil relatif au score de pertinence x, au-dessus, on valide la réponse du classifieur, en-dessous, on ne la valide pas. On peut toujours calculer la précision et le rappel pour toutes les réponses dont le score est au-dessus d'un seuil s. La courbe ROC s'obtient en faisant varier s.

Définition D1 : Courbe ROC

On suppose que Y est la variable aléatoire des scores des expériences qui ont réussi. X est celle des scores des expériences qui ont échoué. On suppose également que tous les scores sont indépendants. On note F_X et F_Y les fonctions de répartition de ces variables. On définit en fonction d'un seuil $s \in \mathbb{R}$:

— $R(s) = 1 - F_Y(s)$

— $E(s) = 1 - F_X(s)$

La courbe ROC est le graphe $(E(s), R(s))$ lorsque s varie dans $\mathbb{R}$.

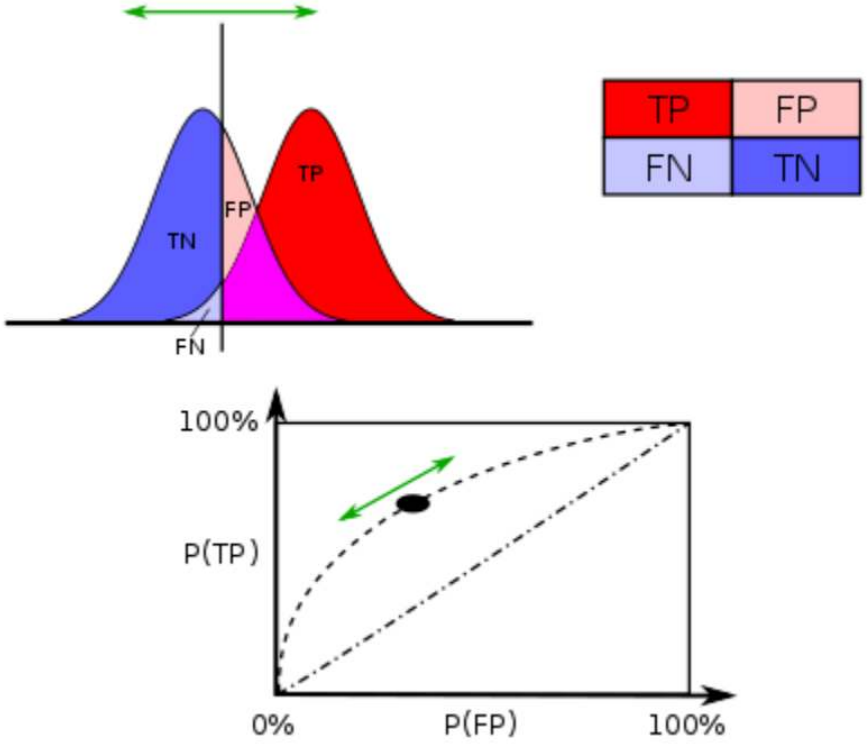
---

113. https://en.wikipedia.org/wiki/Information_retrieval#Precision
114. https://en.wikipedia.org/wiki/Information_retrieval#Recall

$TP(s)$ désigne les true positifs au-dessus du seuil s, avec les notations *TP*, *FP*, *FN*, *TN*, cela revient à :

$$\begin{aligned} E(s) &= 1 - \frac{TP(s)}{TP(s) + TN(s)} \\ R(s) &= 1 - \frac{FN(s)}{FP(s) + FN(s)} \end{aligned}$$

On remarque que $\forall s,\ TP(s) + TN(s)$ est constant. De même pour $FP(s) + FN(s)$.



On remarque que les fonctions $s \longrightarrow E(s)$ et $s \longrightarrow R(s)$ sont décroissantes toutes deux. Elles sont donc inversibles. Dans le cas où la variable aléatoire θ est indépendante de la variable X, la courbe ROC est une droite reliant les points $(0, 0)$ et $(1 - p, p)$ où $p = \mathbb{P}(\theta = 1)$. Ceci signifie que la connaissance du score X n'apporte pas d'information quant à la réussite de l'expérience.

Il peut paraître complexe de distinguer la réponse et le score du classifieur. C'est pourtant nécessaire dans le cas où le classifieur retourne un entier qui désigne une classe parmi n. Un cas positif est lorsque la classe prédite est égale à la classe attendue, il est négatif dans le cas contraire. La courbe peut être adaptée pour d'autres problèmes tels que le ranking (voir *[Agarwal2005]* (page **??**)).

5.1.2 Aire sous la courbe

Expression

L'aire sous la courbe (AUC) correspond à l'intégrale de la fonction ROC. Elle se calcule à partir du théorème suivant :

Théorème T1 : Aire sous la courbe (AUC)

On utilise les notations de la définition de la *Courbe ROC* (page 130). L'aire sous la courbe ROC est égale à $\mathbb{P}(Y > X)$.

Rappel

Soit X une variable aléatoire de densité f et de fonction de répartition F. Si $U = F(X)$, alors :

$$\mathbb{P}(U \leqslant t) = \mathbb{P}(F(X) \leqslant t) = \mathbb{P}\left(X \leqslant F^{-1}(t)\right) = F\left(F^{-1}(t)\right) = t$$

La variable U est de loi uniforme sur $[0,1]$. De plus, soit g une fonction intégrable quelconque, on pose $u = F(x)$ et :

$$\int_{\mathbb{R}} g(x)\, f(x)\, dx = \int_{[0,1]} g(F^{-1}(u))\, du$$

Démonstration

On note f_X la densité de la variable X et f_Y celle de la variable Y. On peut alors définir la probabilité $\mathbb{P}(Y > X)$ par une intégrale :

$$P(Y > X) = \int_x \int_y f_X(x)\ f_Y(y)\ \mathbb{1}_{\{y>x\}} dxdy$$

On note F_X la fonction de répartition de X soit $F_X(x) = \int_{-\infty}^{x} f_X(u)du$. On pose comme changement de variable : $u = F_X(x)$. On en déduit que $du = f_X(x)dx$. La variable aléatoire $U = F_X(X)$ est uniforme et comprise dans $[0,1]$.

$$\begin{aligned} P(Y > X) &= \int_x f_X(x)dx \int_y f_Y(y)\ \mathbb{1}_{\{y>x\}} dy \\ &= \int_u du \int_y f_Y(y)\ \mathbb{1}_{\left\{y>F_X^{-1}(u)\right\}} dy \\ &= \int_u du\ \mathbb{P}\left(Y > F_X^{-1}(u)\right) \end{aligned}$$

Or si $u = F_X(s) = E(s)$, alors $F_X^{-1}(u) = s$ et $\mathbb{P}\left(Y > F_X^{-1}(u)\right) = R'(s)$. Par conséquent :

$$P(Y > X) = \int_u du\ \mathbb{P}\left(Y > F_X^{-1}(u)\right) = \int_u du\ R'(F_X^{-1}(u))$$

Cette dernière expression est l'aire recherchée. Ce théorème nous permet de définir un estimateur pour l'aire sous la courbe ROC à l'aide des U-statistiques [115] de Mann-Whitney [116] (voir *[Saporta1990]* (page **??**)).

Corollaire C1 : Estimateur de l'aire sous la courbe ROC

On dispose des scores $(Y_1, \ldots, Y_n)$ des expériences qui ont réussi et $(X_1, \ldots, X_m)$ les scores des expériences qui ont échoué. On suppose également que tous les scores sont indépendants. Les scores (Y_i) sont identiquement distribués, il en est de même pour les scores (X_i). Un estimateur de l'aire A sous la courbe ROC¿est :

$$\hat{A} = \frac{1}{nm} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\mathbb{1}_{\{Y_j > X_i\}} + \frac{1}{2} \mathbb{1}_{\{Y_j = X_i\}} \right) \tag{5.1}$$

Démonstration

La démonstration est évidente :

$$\mathbb{E}\left(\hat{A}\right) = \frac{1}{nm} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\mathbb{P}(Y_j > X_i) + \frac{1}{2}\mathbb{P}(X_i = Y_j) \right) = \mathbb{P}(Y > X) + \frac{1}{2}\mathbb{P}(Y = X)$$

Dans le cas où X ou Y sont continues, $\mathbb{P}(X = Y) = 0$.

115. https://en.wikipedia.org/wiki/U-statistic
116. https://fr.wikipedia.org/wiki/Test_de_Wilcoxon-Mann-Whitney

Intervalles de confiance

Il est possible de déterminer un intervalle de confiance pour cet estimateur. Le théorème central limite nous permet de dire que cet estimateur tend vers une loi normale lorsque n et m tendent vers l'infini.

Corollaire C2 : Variance de l'estimateur AUC

On note $P_X = \mathbb{P}(X < \min\{Y_i, Y_j\})$ et $P_Y = \mathbb{P}(\max\{X_i, X_j\} < Y)$. X_i et X_j sont de même loi que X, Y_i, Y_j sont de même loi que Y. La variance de l'estimateur $\hat{A}$ définie par *(1)* (page **??**) est :

$$\mathbb{V}\hat{A} = \frac{\hat{A}(1-\hat{A})}{nm}\left[1 + (n-1)\frac{P_Y - \hat{A}^2}{\hat{A}(1-\hat{A})} + (m-1)\frac{P_X - \hat{A}^2}{\hat{A}(1-\hat{A})}\right]$$

Démonstration

Cette démonstration n'est vraie que dans le cas continu. Par conséquent, $\mathbb{P}(X = Y) = 0$. On calcule tout d'abord $\mathbb{E}\hat{A}^2$ et on utilise le fait que $\mathbb{V}\hat{A} = \mathbb{E}\left(\hat{A}^2\right) - \hat{A}^2$.

$$\hat{A}^2 = \frac{1}{n^2m^2}\left[\sum_{i=1}^{m}\sum_{j=1}^{n}\mathbb{1}_{\{X_i<Y_j\}}\right]^2 = \frac{1}{n^2m^2}\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{m}\sum_{l=1}^{n}\mathbb{1}_{\{X_i<Y_j\}}\mathbb{1}_{\{X_k<Y_l\}}$$

$$\begin{array}{rcl}\hat{A}^2 &=& \frac{1}{n^2m^2}\sum_{i=1}^{m}\sum_{j=1}^{n}\mathbb{1}_{\{X_i<Y_j\}} \\ && +\frac{1}{n^2m^2}\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k\neq i}\mathbb{1}_{\{X_i<Y_j\}}\mathbb{1}_{\{X_k<Y_j\}} \\ && +\frac{1}{n^2m^2}\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{l\neq j}\mathbb{1}_{\{X_i<Y_j\}}\mathbb{1}_{\{X_i<Y_l\}} \\ && +\frac{1}{n^2m^2}\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k\neq i}\sum_{l\neq j}\mathbb{1}_{\{X_i<Y_j\}}\mathbb{1}_{\{X_k<Y_l\}}\end{array}$$

On en déduit que :

$$\begin{array}{rcl}\mathbb{E}\hat{A}^2 &=& \frac{\hat{A}}{nm} + \frac{n-1}{nm}\mathbb{P}(\max\{X_i, X_k\} < Y_j) + \\ && \frac{m-1}{nm}\mathbb{P}(X_i < \min\{Y_j, Y_l\}) + \frac{nm-n-m-1}{nm}\hat{A}^2 \\ \mathbb{V}\hat{A}^2 &=& \frac{1}{nm}\left[\hat{A} + (n-1)P_Y + (m-1)P_X - (n+m+1)\hat{A}^2\right] \\ &=& \frac{1}{nm}\left[\hat{A} + (n-1)\left(P_Y - \hat{A}^2\right) + (m-1)\left(P_X - \hat{A}^2\right) + \hat{A}^2\right]\end{array}$$

On retrouve l'expression cherchée.

5.1.3 Intervalles de confiance sur la courbe

Les systèmes de reconnaissance sont souvent ajustés de telle manière que le taux d'erreur soit constant, par exemple 1%. C'est la proportion de documents reconnus qui détermine la performance de ce système. L'objectif ce paragraphe est de déterminer un intervalle de confiance du taux de reconnaissance pour un taux d'erreur fixé.

Construction de la courbe ROC

Ce premier paragraphe détaille la manière dont est construite une courbe ROC (voir *Courbe ROC* (page 130)).

Algorithme A1 : Courbe ROC

On suppose qu'on dispose d'un ensemble de points $(X_i, \theta_i) \in \mathbb{R} \times \{0, 1\}$ pour $i \in \{1, \ldots, n\}$. *X_i* est le score obtenu pour l'expérience i, *theta_i* vaut 1 si elle a réussi et 0 si elle a échoué. On suppose également que cette liste est triée

par ordre croissant : *forall i, ; X_i infegal X_{i+1}*. On souhaite également tracer k points sur la courbe, on détermine pour cela k seuils *ensemble{s_1}{s_k}* définis par : $\forall j, s_k = X_{\frac{j\,k}{n}}$.

On construit ensuite les points (R_j, E_j) définis par :

$$R_j \quad = \quad \frac{1}{n}\sum_{i=1}^{n} \theta_i \mathbb{1}_{\{X_i \geqslant s_j\}} \text{ et } E_j = \frac{1}{n}\sum_{i=1}^{n} (1-\theta_i)\ \mathbb{1}_{\{X_i \geqslant s_j\}}$$

La courbe ROC est composée de l'ensemble $R_{OC} = \{(E_j, R_j)\,|1 \leqslant j \leqslant k\}$.

Les deux suites $(R_j)_j$ et $(E_j)_j$ sont toutes les deux décroissantes d'après leur définition. La courbe peut être rendue continue par interpolation.

Définition D2 : taux de classification à erreur fixe

On cherche un taux de reconnaissance pour un taux d'erreur donné. On dispose pour cela d'une courbe ROC obtenue par l'algorithme de la *courbe ROC* (page 133) et définie par les points $R_{OC} = \{(e_j, r_j)\,|1 \leqslant j \leqslant k\}$. On suppose ici que $(e_1, r_1) = (1, 1)$ et $(e_k, r_k) = (0,)$. Si ce n'est pas le cas, on ajoute ces valeurs à l'ensemble R_{OC}.

Pour un taux d'erreur donné e^*, on cherche j^* tel que :

$$e_{j^*+1} \leqslant e^* \leqslant e_{j^*}$$

Le taux de reconnaissance ρ cherché est donné par :

$$\rho = \frac{e^* - x_{j^*}}{x_{j^*+1} - x_{j^*}}\ [r_{j^*+1} - r_{j^*}] + r_{j^*}$$

Il ne reste plus qu'à détailler la méthode *bootstrap*.

Méthode boostrap

Une seule courbe ROC ne permet d'obtenir qu'un seul taux. On cherche ici à construire plusieurs courbes ROC à partir de la même expérience de façon à obtenir plusieurs taux de reconnaissance pour le même taux d'erreur. De cette manière, il sera possible de déterminer un intervalle de confiance. On s'inspire pour cela des méthodes de bootstrap [117].

Algorithme A2 : Courbe ROC, méthode boostrap

On dispose toujours du nuage de points $E = (X_i, \theta_i) \in \mathbb{R} \times \{0, 1\}$ avec $i \in \{1, \ldots, n\}$. On choisit $C \in \mathbb{N}$ le nombre de courbes ROC qu'on désire tracer. Pour chaque courbe $c \in \{1, \ldots, C\}$:

- On construit un nouvel ensemble $(X'_i, \theta'_i)_{1 \leqslant i \leqslant n}$ construit par un tirage aléatoire dans l'ensemble E avec remise.
- L'algorithme de la *courbe ROC* (page 133) permet de constuire la courbe R^k_{OC}.
- L'algorithme de *taux de classification à erreur fixe* (page 134) permet ensuite de déterminer un taux de reconnaissance ρ_k pour le taux d'erreur e^*.

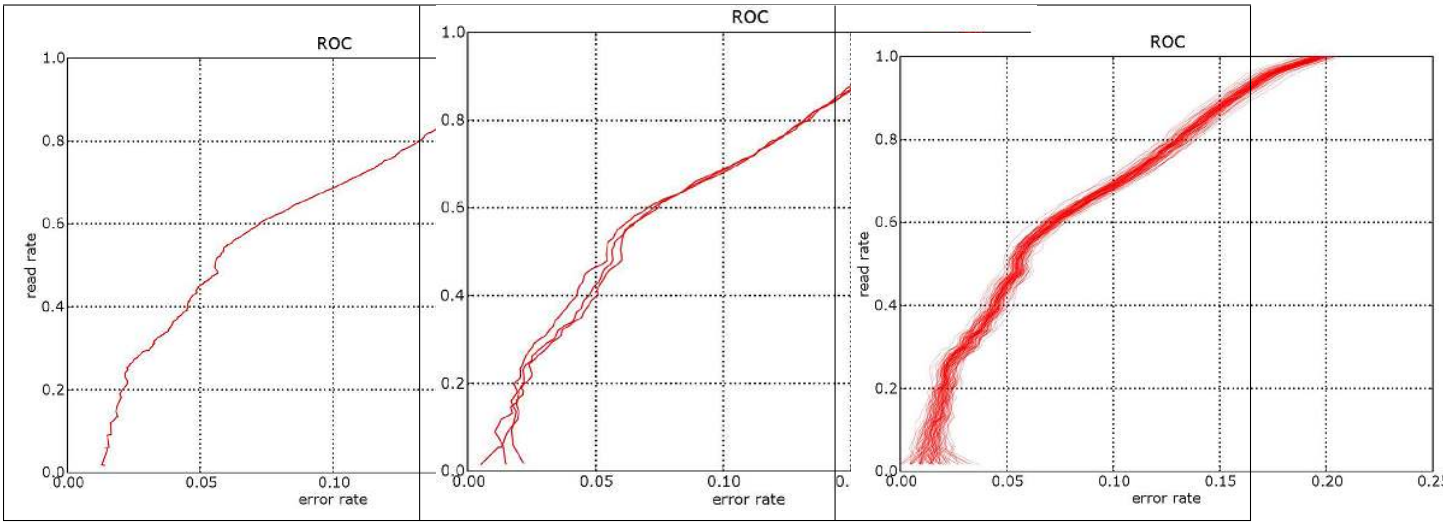
La liste $(\rho_1, \ldots, \rho_C)$ est triée par ordre croissant. Les quantiles sont ensuite utilisés pour déterminer l'intervalle de confiance $[\rho_1, \rho_2]$ du taux de reconnaissance pour le taux d'erreur e^* de telle sorte que :

$$\mathbb{P}\left(\rho \in [\rho_1, \rho_2]\right) = 1 - \alpha$$

On prend généralement $\alpha = 0.05$.

Cet algorithme aboutit aux résultats suivants :

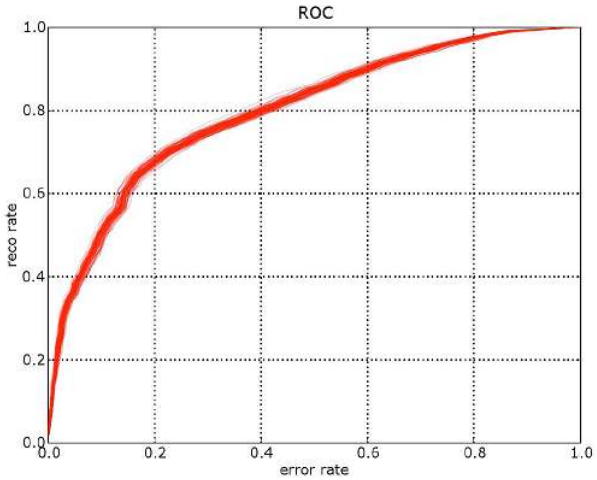
117. https://fr.wikipedia.org/wiki/Bootstrap_(statistiques)



La première image est celle d'une courbe ROC (l'axe des abscisses est inversé), la seconde représente toutes celles obtenues par la méthode bootstrap pour trois courbes. La troisième image superpose cent courbes. Moins il y a de points pour estimer une partie de la courbe, plus les courbes sont espacées. Ces courbes ont été construites avec 12000 points. Le taux de lecture pour 1% d'erreur est égal à 68,09%. L'intervalle de confiance à 95% est [66,10% ; 70,16%] (construit avec 500 courbes). Moyenne (68,25) et médiane (68,12) sont sensiblement égales au taux calculé sur la première courbe construite sans tirage aléatoire. L'écart-type est $1,10$, cela donne un intervalle de confiance équivalent au précédent si on considère que la moyenne des taux suit asymptotiquement une loi normale. Cette expérience a été reproduite plusieurs fois et ces bornes sont assez stables contrairement (*pm 0,05 %*) aux extremas (*pm 1%*).

Aire sous la courbe

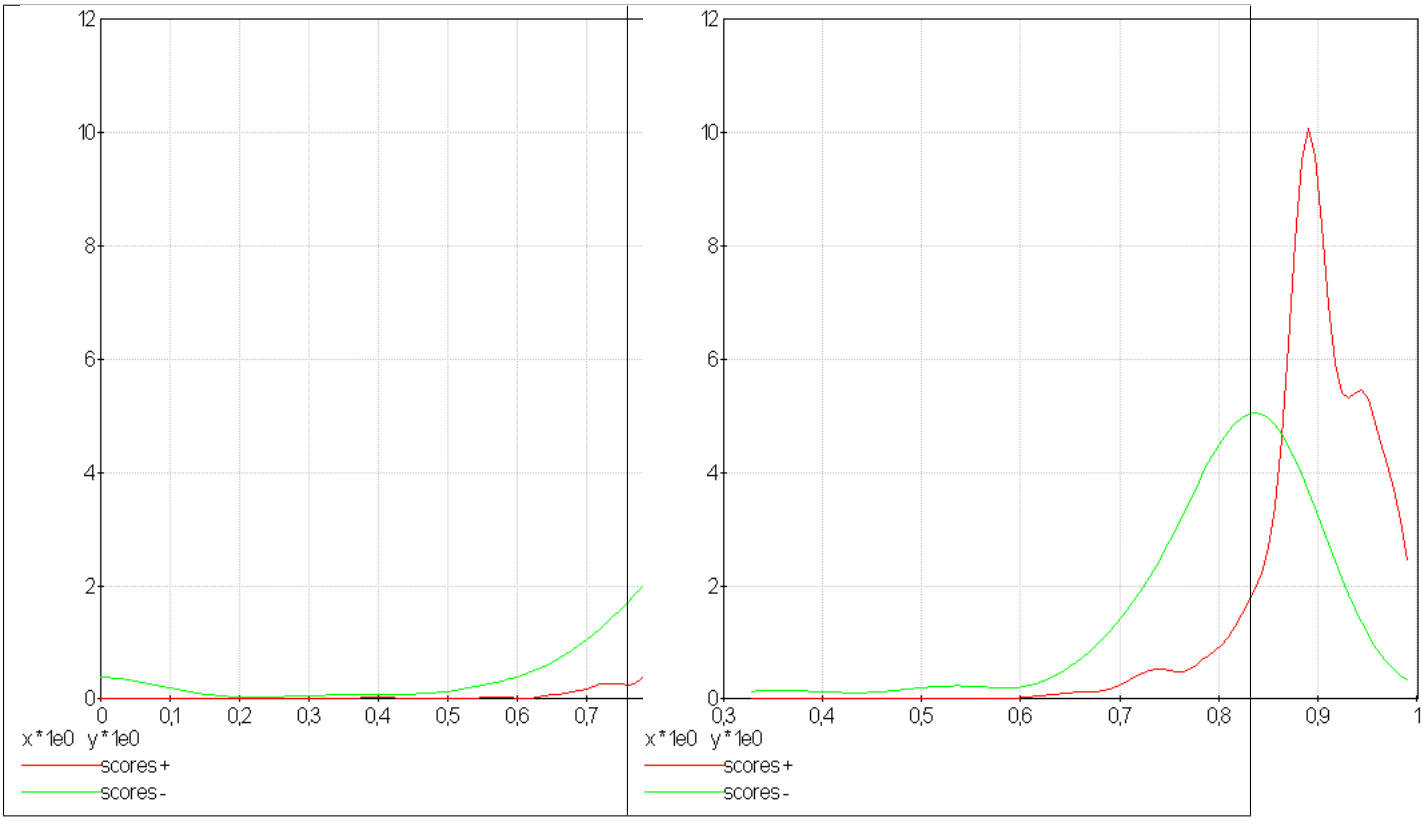
La méthode bootstrap peut elle aussi être appliquée pour calculer un intervalle de confiance pour l'aire sous la courbe (AUC).



Courbe ROC (l'axe des abscisse est inversé) obtenue pour 100 tirages aléatoires. L'aire sous la courbe est égale à 0.80 et l'intervalle de confiance à 95% mesurée par la méthode bootsrap est : $[0.79,\ 0.80]$. Les extremas sont presque identiques à ces chiffres.

5.1.4 Distribution des scores mauvais et bons

On appelle un mauvais score un score associé à un mauvais résultat, de même, un bon score est le score d'un bon résultat. Si le score est une probabilité, on s'attend à trouver les bons scores regroupés autour de la valeur 1. Si le score est un mauvais score, il devrait être plus proche de zéro. La figure qui suit montre des distributions obtenues pour deux problèmes différents. Dans les deux cas, le but recherché est la détermination d'un seuil séparant le score d'un bon résultat de celui d'un mauvais résultat. Lorsque ceci n'est pas possible, le score ne peut correspondre à un quelconque critère confiance.

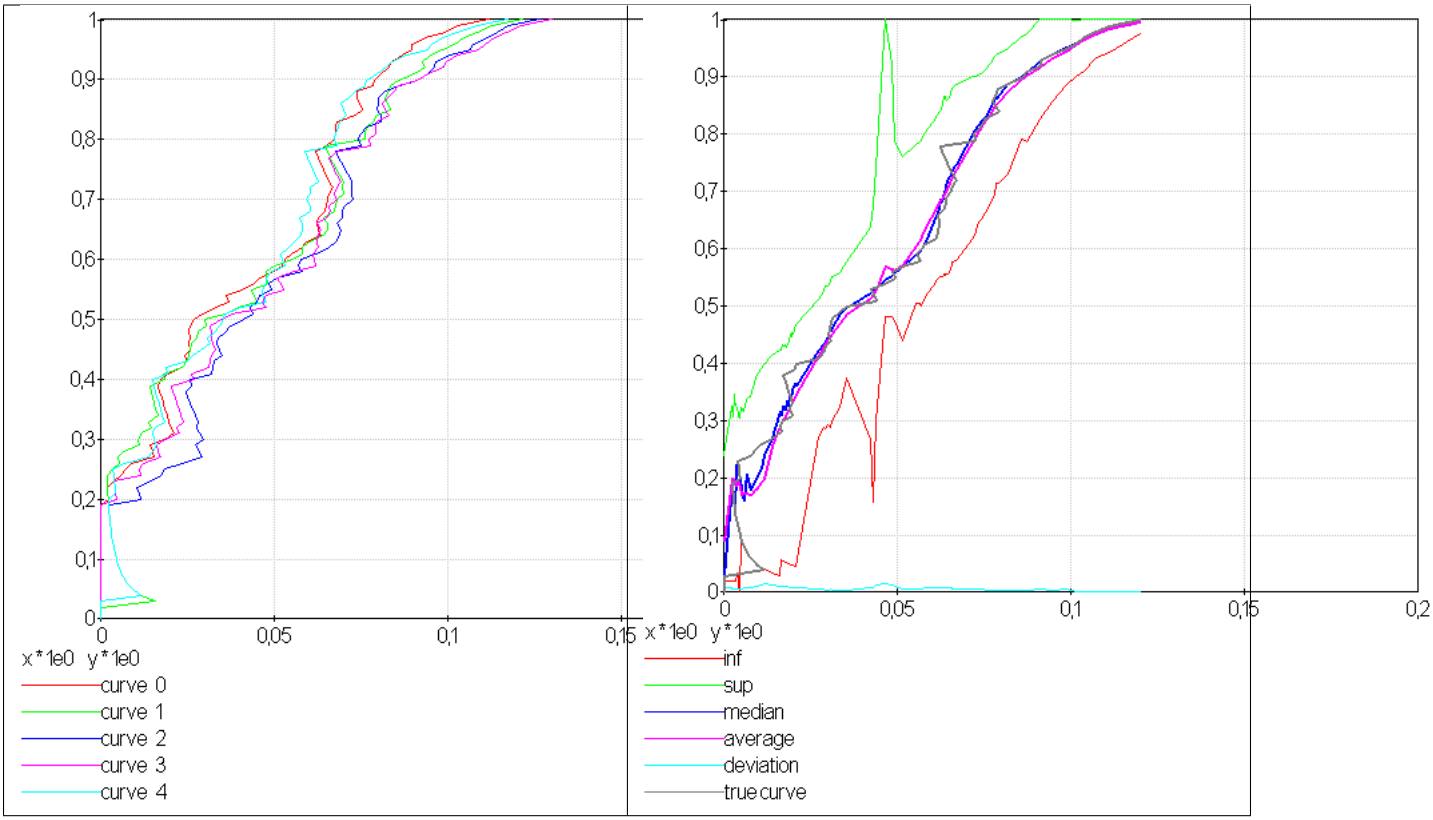


La première courbe montre deux distributions qui se chevauchent même si les bons scores semblent plus concentrés autour des grandes valeurs. Le seconde courbe montre un problème mieux séparable. L'existence d'un seuil entre un bon et un mauvais score est plus plausible.

5.1.5 Variantes

Taux de lecture ou de reconnaissance

Il n'existe pas une grande différence lorsque le taux d'erreur est faible. Le taux de lecture est simplement la proportion de documents pour lesquels le score est aussi d'un seuil s que la réponse du classifieur soit bonne ou mauvaise. Par exemple, pour un taux de *substitution* de 1%, si on a 70% en taux de lecture, cela signifie que sur 100 documents, le système va en accepter 70 et parmi ces 70, 1% seront mal traités. Le taux de substitution est un taux d'erreur rapporté à un taux de lecture donné. L'inconvénient du taux de lecture rapporté au taux de substitution est que la méthode développée au paragraphe *Intervalles de confiance sur la courbe* (page 133) ne s'applique plus aussi bien car pour un taux de substitution donné, il peut exister plusieurs taux de lecture.



La première image montre 5 courbes taux de lecture / taux de substitutions. Les courbes ne sont pas monotones et montre qu'il existe parfois plusieurs taux de lecture pour un même taux de substitution. Comme le calcul des intervalles de confiance fait intervenir une interpolation linéaire, lorsque les courbes sont trop cahotiques, le calcul retourne des valeurs fausses.

On peut démontrer que la courbe taux de lecture / taux de substitution n'est pas une courbe ni monotone ni inversible. Pour cela on dispose d'une suite de couple (X_i, θ_i) croissante selon les X_i. θ_i vaut 1 si l'expérience a réussi, 0 sinon. Pour un seuil donné s, on note $E'(s)$ le taux de substitution et $R'(s)$ le taux de lecture, on obtient :

$$\begin{aligned} R'(s) &= \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{X_i \geqslant s\}} \\ E'(s) &= \frac{1}{n\,R'(s)} \sum_{i=1}^{n} (1-\theta_i)\ \mathbb{1}_{\{X_i \geqslant s\}} \end{aligned}$$

On écrit différemment ces expressions en supposant que $X_{i(s_1)-1} < s_1 \leqslant X_{i(s_1)}$: $math$: :

$$\begin{aligned} R'(s_1) &= \frac{n - i(s_1)}{n} \\ E'(s_1) &= \frac{1}{n - i(s_1)} \sum_{i=i(s_1)}^{n} (1-\theta_i) \end{aligned}$$

On suppose maintenant que $X_{i(s_2)-1} < s_2 \leqslant X_{i(s_2)}$: $math$: et $i(s_1)+1 = i(s_2)$:

$$\begin{aligned}
R'(s_2) &= \frac{n-i(s_2)}{n} < R'(s_1) \\
E'(s_2) &= \frac{1}{n-i(s_2)} \sum_{i=i(s_2)}^{n} (1-\theta_i) = \frac{1}{n-i(s_2)} \frac{n-i(s_1)}{n-i(s_1)} \left(-\left(1-\theta_{i(s_1)}\right) + \sum_{i=i(s_1)}^{n} (1-\theta_i) \right) \\
&= -\frac{\left(1-\theta_{i(s_1)}\right)}{n-i(s_2)} + \frac{\sum_{i=i(s_1)}^{n}(1-\theta_i)}{n-i(s_1)} \frac{n-i(s_1)}{n-i(s_2)} = -\frac{\left(1-\theta_{i(s_1)}\right)}{n-i(s_2)} + E'(s_1)\frac{n-i(s_1)}{n-i(s_2)}
\end{aligned}$$

Si on suppose que $\theta_{i(s_1)} = 1$, autrement dit, l'expérience s_1 a réussi, on en déduit que :

$$E'(s_2) = E'(s_1)\frac{n-i(s_1)}{n-i(s_2)} = E'(s_1)\frac{n-i(s_2)+1}{n-i(s_2)} > E'(s_1)$$

En revanche si $\theta_i = 0$:

$$E'(s_2) = E'(s_1)\left(1+\frac{1}{n-i(s_2)}\right) - \frac{1}{n-i(s_2)} = E'(s_1) + \frac{E(s_1)-1}{n-i(s_2)} < E'(s_1)$$

Il n'existe donc pas toujours une fonction f reliant $R'(s)$ à $E'(s)$ à moins de construire cette courbe de telle sorte qu'elle soit monotone en ne choisissant qu'une sous-suite $\left(E'(X_i), R'(X_i)\right)_i$ qui vérifie cette hypothèse.

5.1.6 Classification multi-classe

Une courbe ROC se construit avec deux informations : une réponse binaire et un score. Que signifie cette réponse binaire ? Elle peut être :

1. Le fait que le prédicteur ait bien prédit une classe en particulier. Le score associé est celui que le prédicteur donne pour cette classe.
2. Le fait que le prédicteur ait bien prédit, c'est-à-dire que la réponse binaire signifie que la classe prédite est la classe attendue, le score associé est celui de la classe prédite, c'est-à-dire le score maximum obtenu pour l'une des classes.

Plus formellement, le prédicteur retourne un vecteur S_i qui contient les probabilités d'appartenance à toutes les classes aussi appelées plus généralement score de confiance ou juste score. $S_i(c)$ est la probabilité de prédire la classe c. La classe attendue est notée y_i, c'est celle que le prédicteur doit prédire. Dans le premier cas, on construit le couple (b_i, s_i) de telle sorte que :

$$\begin{aligned}
&b_i = 1 \; si \; y_i = c \; sinon \; 0 \\
&s_i = S_i(c)
\end{aligned}$$

Dans le second cas :

$$\begin{aligned}
&b_i = 1 \; si \; \max S_i = S_i(y_i) \; sinon \; 0 \\
&s_i = \max S_i
\end{aligned}$$

Le premier cas correspond par exemple à des problèmes de détection de fraude [118]. Le second cas correspond à taux de classification global. La courbe ROC pour ce cas est en règle général moins bonne que la plupart des courbes ROC obtenues pour chacune des classes prise séparément (voir Régression logistique [119]).

5.1.7 Exemple

Voir ROC [120].

118. https://en.wikipedia.org/wiki/Predictive_analytics#Fraud_detection
119. http://www.xavierdupre.fr/app/papierstat/helpsphinx/notebooks/wines_color.html
120. http://www.xavierdupre.fr/app/ensae_teaching_cs/helpsphinx3/antiseches/ml_basic/plot_regression.html#sphx-glr-antiseches-ml-basic-plot-roc-py

5.2 Confidence Interval and p-Value

This document explains the relationship between p-value and confidence intervals. It goes on with the specific case of a binamial law. Assuming we want to determine whether or not two binomial laws are significantly different, how many observations we need to get the p-value under a given threshold.

5.2.1 Introduction

The term p-value [121] is very popular in the world of search engines. I usually prefer confidence interval 95%, I think it is easier to understand. Plus, because p-Value are real values, we could be tempted to compare them and it is usually wrong. On the other hand, it is more difficult to compare confidence intervals, especially if they are related to complete different variables. Their nature prevents you from doing that. Howerver p-Values and confidence interval are similar : they tell you whether or not a metric difference is significant.

Usually, it starts from a set of identically distributed random variables $(X_i)_{1 \leqslant i \leqslant N}$. We then estimate the average $\widehat{\theta}_N = \frac{1}{N} \sum_{i=1}^{N} X_i$ and we ask the question is $\widehat{\theta}_N$ null ? In others terms, we want to know if the average is significantly different from zero. If the random variable X follows a random law which has a standard deviation, we can use the central limit theorem [122] which tells us :

$$\sqrt{N}\widehat{\theta}_N \underset{N \to \infty}{\longrightarrow} \mathcal{N}(0, \sigma)$$

Not all of them have a standard deviation. For example, if X follows a Cauchy law [123], $\mathbb{E}X^2 \sim \int \frac{x^2}{1+x^2} dx$ which does not exist. This remark also concerns every distribution known as heavy tail distribution.

If $Y \sim \mathcal{N}(0, \sigma)$, then we have $\mathbb{P}(|Y| \leqslant 1.96) = 0.95$. That is why we can say :

$$\widehat{\theta}_N \text{ is not null with 95\% confidence if } \sqrt{N}\frac{|\widehat{\theta}_N|}{\sigma} > 1.96$$

And the confidence intervalle at 95% would be :

$$\left[-\frac{1.96\sigma}{\sqrt{N}}, \frac{1.96\sigma}{\sqrt{N}} \right]$$

When $\mathbb{E}\widehat{\theta}_N = \theta_0 \neq 0$, it becomes :

$$\sqrt{N}\left[\widehat{\theta}_N - \theta_0\right] \underset{N \to \infty}{\longrightarrow} \mathcal{N}(0, \sigma)$$

We usually want to check if the mean is equal to a specific value using a statistical test :

$$\begin{aligned} H0 : &\quad \widehat{\theta}_N = \theta_0 \\ H1 : &\quad \widehat{\theta}_N \neq \theta_0 \end{aligned}$$

121. https://fr.wikipedia.org/wiki/Valeur_p
122. https://en.wikipedia.org/wiki/Central_limit_theorem
123. https://en.wikipedia.org/wiki/Cauchy_distribution

We validate $H0$ if :

$$\widehat{\theta}_N \in \left[\theta_0 - \frac{1.96\sigma}{\sqrt{N}}, \theta_0 + \frac{1.96\sigma}{\sqrt{N}}\right]$$

5.2.2 p-value

With confidence intervals, you first choose a confidence level and then you get an interval. You then check if your value is inside or outside your interval. Inside, the gain is not significant, outside, it is.

With a p-value, we consider the problem the other way, given $\widehat{\theta}_N$, what is the probability that the difference $\left|\widehat{\theta}_N - \theta_0\right|$ is significant ? Let's consider Y following a normal law $\mathcal{N}(0,1)$. We are looking for :

$$\mathbb{P}\left(|Y| > \sqrt{N}\frac{|\widehat{\theta}_N|}{\sigma}\right) = \alpha$$

α is the p-value.

$$\begin{aligned} \alpha &= 1 - \int_{-\beta_N}^{\beta_N} \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} dx = 2\int_{\beta_N}^{\infty} \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} dx \\ \text{where } \beta_N &= \sqrt{N}\frac{|\widehat{\theta}_N|}{\sigma} \end{aligned}$$

At this point, we should not forget that we use a theorem which tells us that $\sqrt{N}\frac{|\widehat{\theta}_N|}{\sigma} \sim \mathcal{N}(0,1)$ when $N \to \infty$, which means everything we said is true when N is great enough.

5.2.3 Significant difference between samples mean

Usually, we do not want to test if an average is null but if the difference between two averages is null. We consider two random samples having the same size, each of them described by (X_i) and (Y_i). All variables are independant. (X_i) are distributed according the same law, we assume the same for (Y_i). We expect the following difference to be null.

$$\widehat{\eta}_N = \frac{1}{N}\sum_{i=1}^{N} X_i - \frac{1}{N}\sum_{i=1}^{N} Y_i = \frac{1}{N}\left[\sum_{i=1}^{N} X_i - Y_i\right]$$

Considering expression *(2)* (page **??**), we can applying the central limit theorem on variable $Z = X - Y$, we get $(\eta_0 = 0)$:

$$\sqrt{N}\widehat{\eta}_N \underset{N\to\infty}{\longrightarrow} \mathcal{N}\left(\eta_0, \sqrt{\frac{\mathbb{V}Z}{N}}\right)$$

If both samples do not have the same number of observations, this expression becomes :

$$\sqrt{N}\widehat{\eta}_N \underset{\substack{N_1\to\infty \\ N_2\to\infty \\ \frac{N_1}{N_2}\to x}}{\longrightarrow} \mathcal{N}\left(\eta_0, \sqrt{\frac{\mathbb{V}X}{N_1} + \frac{\mathbb{V}Y}{N_2}}\right)$$

5.2.4 Application on binomial variables

A binomial variable $X \sim \mathcal{B}(p)$ is defined by :

$$\begin{aligned} \mathbb{P}(X=0) &= 1-p \\ \mathbb{P}(X=1) &= p \end{aligned}$$

Let's consider two series of observations $(X_i) \sim \mathcal{B}(p)$ and $(Y_i) \sim \mathcal{B}(q)$. We assume $p \neq q$ and we want to determine how many observations we need to get a p-value below 5%. We know that $\mathbb{V}(X_i) = p(1-p)$ and $\mathbb{V}(Y_i) = q(1-q)$. Next table shows the values. First column contains values for p, first row contains values for $q-p$. We also assume we have the same number N of random observations for each variable. The statistical test cen be defined like following :

$$\begin{aligned} H0 &: \quad p = q = p_0 \\ H1 &: \quad p \neq q \end{aligned}$$

If H0 is true, then :

$$\sqrt{N}\widehat{\theta}_N \underset{N\to\infty}{\longrightarrow} \mathcal{N}\left(0, \sqrt{p_0(1-p_0)}\sqrt{\frac{1}{N_1}+\frac{1}{N_2}}\right) \tag{5.2}$$

p/d	**-0.200**	**-0.100**	**-0.020**	**-0.010**	**-0.002**	**-0.001**	**0.001**	**0.002**	**0.010**	**0.020**	**0.100**	**0.200**
0.05			913	3650	91235	364939	364939	91235	3650	913	37	10
0.10		70	1729	6915	172866	691463	691463	172866	6915	1729	70	18
0.15		98	2449	9796	244893	979572	979572	244893	9796	2449	98	25
0.20	31	123	3074	12293	307317	1229267	1229267	307317	12293	3074	123	31
0.25	37	145	3602	14406	360137	1440548	1440548	360137	14406	3602	145	37
0.30	41	162	4034	16135	403354	1613413	1613413	403354	16135	4034	162	41
0.35	44	175	4370	17479	436966	1747864	1747864	436966	17479	4370	175	44
0.40	47	185	4610	18440	460976	1843901	1843901	460976	18440	4610	185	47
0.45	48	191	4754	19016	475381	1901523	1901523	475381	19016	4754	191	48
0.50	49	193	4802	19208	480183	1920730	1920730	480183	19208	4802	193	49
0.55	48	191	4754	19016	475381	1901523	1901523	475381	19016	4754	191	48
0.60	47	185	4610	18440	460976	1843901	1843901	460976	18440	4610	185	47
0.65	44	175	4370	17479	436966	1747864	1747864	436966	17479	4370	175	44
0.70	41	162	4034	16135	403354	1613413	1613413	403354	16135	4034	162	41
0.75	37	145	3602	14406	360137	1440548	1440548	360137	14406	3602	145	37
0.80	31	123	3074	12293	307317	1229267	1229267	307317	12293	3074	123	31
0.85	25	98	2449	9796	244893	979572	979572	244893	9796	2449	98	
0.90	18	70	1729	6915	172866	691463	691463	172866	6915	1729	70	
0.95	10	37	913	3650	91235	364939	364939	91235	3650	913		

Given a binomial law with parameter p and a difference d, this table gives the number of observations needed on both sides to get a significant difference assuming p is the expected pourcentage

5.2.5 Estimate a p-value by using the distribution function

Expression *(1)* (page **??**) gives a way to estimate the p-value. Computing the integral is not always possible but there is a way to do it using Monte Carlo method [124]. Let's assume $X \sim \mathcal{N}(0,1)$. We denote f_X as the density function of X. We also consider an interval $I = [-a, a]$. Then we have $f(a) = f(-a)$ and :

$$\mathbb{P}(X \in I) = \mathbb{P}(|X| \leqslant a) = \mathbb{P}(f(X) \geqslant f(a))$$

This is true because f is decreasing for $x > 0$. The p-value α for a estimator β using Monte Carlo method is :

$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\{f(X_i)<f(\beta)\}} \longrightarrow \alpha \tag{5.3}$$

Assuming every $(X_i)_i$ follows a normal law $\mathcal{N}(0,1)$.

124. https://en.wikipedia.org/wiki/Monte_Carlo_method

5.2.6 Correlated variables

Let's assume we now have a vector a correlated variables $X = (X_1, ...X_d)$ drawn following a law $\mathcal{N}(\theta_0, \Sigma)$.
The central limit theorem is still valid :

$$\sqrt{N}\widehat{\theta}_N \underset{N\to\infty}{\longrightarrow} \mathcal{N}(\theta_0, \Sigma)$$

We know estimators for the average and the covariance matrix defined as follows :

$$\begin{aligned}\widehat{\theta_N} &= \frac{1}{n}\sum_{i=1}^{N} X_i \\ \widehat{\Sigma_N} &= \frac{1}{n}\sum_{i=1}^{N}(X_i - \widehat{\theta_N})(X_i - \widehat{\theta_N})'\end{aligned}$$

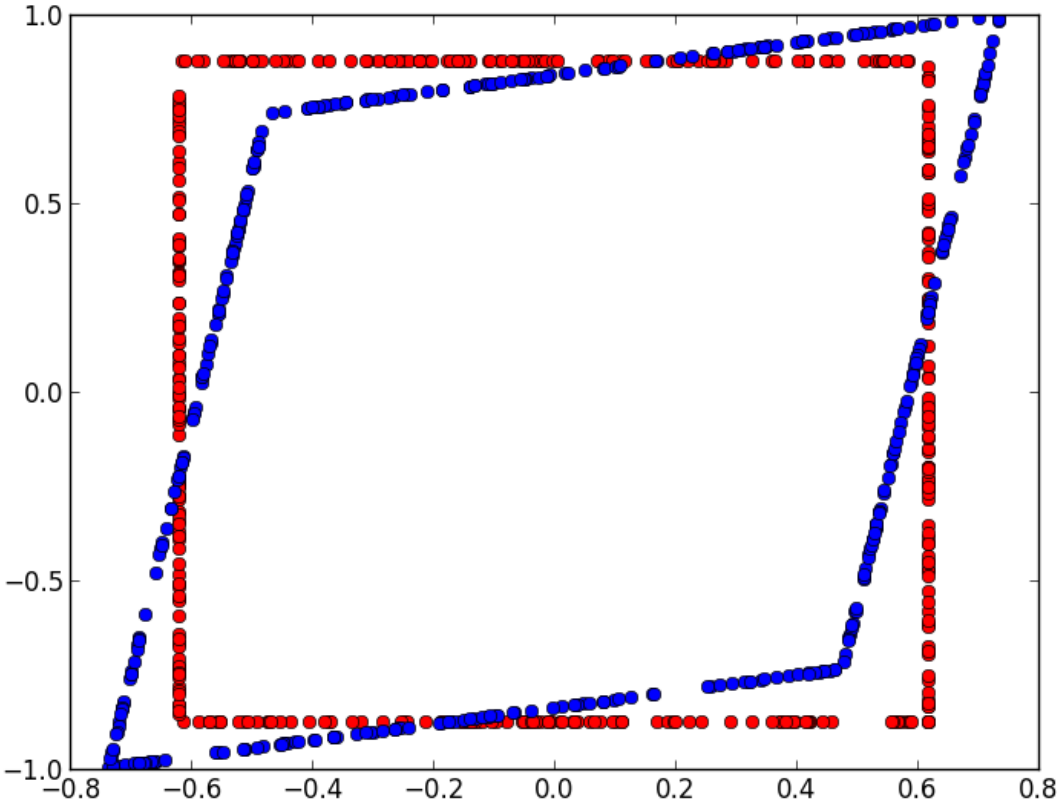
We usually want to check if :

$$\begin{aligned}H0 : &\quad \widehat{\theta}_N = \theta_0 \\ H1 : &\quad \widehat{\theta}_N \neq \theta_0\end{aligned}$$

If Λ is diagonal matrix of Σ (diagonal matrix with eigen values). All eigen values are real and positive, we then define :

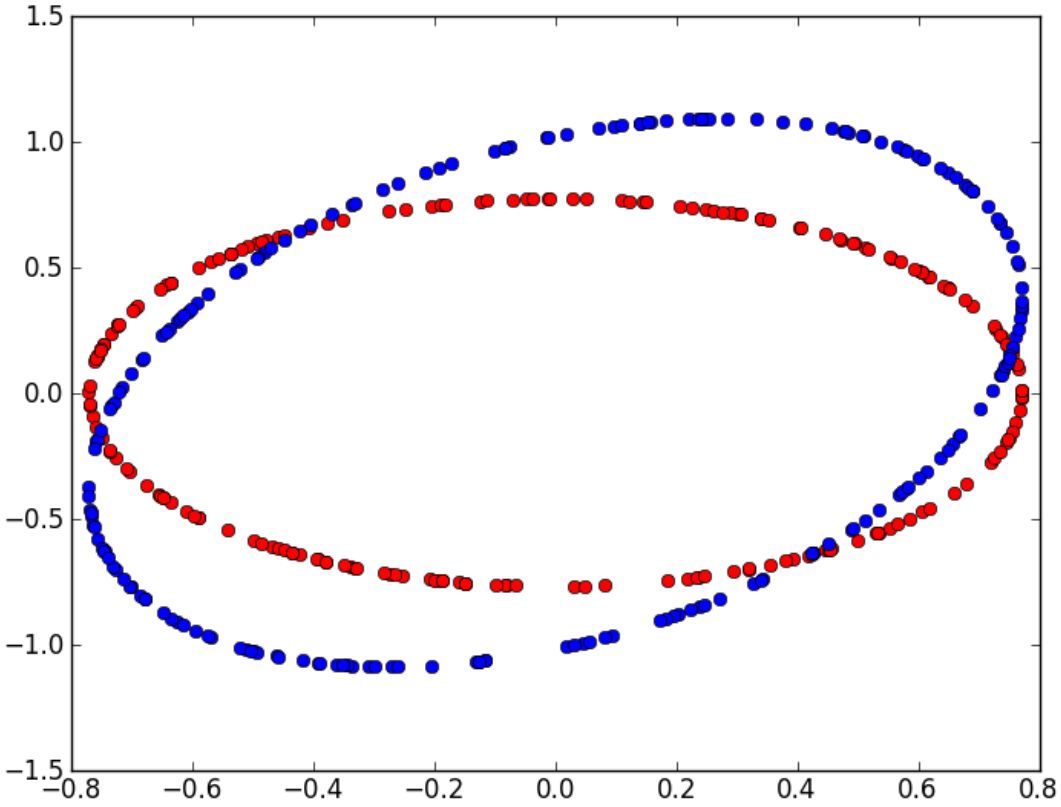
$$\Sigma = P\Lambda P' \text{ and } \Sigma^{\frac{1}{2}} = P\Lambda^{\frac{1}{2}}P'$$

We consider $Z_i = (X_i - \widehat{\theta_N})\Sigma^{-\frac{1}{2}}$. We then have $\mathbb{E}Z_i = 0$ and $\mathbb{V}Z_i = I_2$ where I_2 is the identity matrix. We could now consider each dimension of Z_i independently as illustrated in next figure : it shows the difference on an example if we consider the correlation of two variables correlated such as $\Sigma = \begin{pmatrix} 0.1 & 0.05 \\ 0.05 & 0.2 \end{pmatrix}$.



We assume we observe two Bernouilli [125] variables correlated. Red points represents the area for which we would accept hypothesis *H0* in case both variables are independant. Blue area represents the same but with the correlation. But that would not be the best way to do it. The confidence interval for a couple of indenpendant gaussian (N_1, N_2) variables is an ellipse. Two independent normal variables $N_1^2 + N_2^2$ with a null mean and standard deviation equal to one follows a χ_2 law. Based on that, we can deduce a boundary for the confidence zone at 95%. Next figure shows this zone for a non-correlated couple and a correlated couple ($\Sigma = \begin{pmatrix} 0.1 & 0.05 \\ 0.05 & 0.2 \end{pmatrix}$).

125. https://en.wikipedia.org/wiki/Bernoulli_distribution



We assume we observe two Bernouilli variables correlated. Red points represents the area for which we would accept hypothesis *H0* in case both variables are independant. Blue area represents the same but with the correlation.

5.2.7 Multiple comparisons problem

The problem of Multiple comparisons [126] happens when dealing with many metrics measyring a change. That's allways the case when two version of the sam websire are compared in a test A/B [127]. The metrics are correlated but it is unlikely that all metrics differences will be significant or not. The Holm–Bonferroni method [128] proposes a way to define an hypthesis on the top of the existing ones.

5.2.8 Algorithm Expectation-Maximization

We here assume there are two populations mixed defined by random variable C. Let's X be a mixture of two binomial laws of parameters p and q. It is for example the case for a series draws coming from two different coins.

$$\mathbb{P}(X) \quad = \quad \mathbb{P}(X|C=a)\,\mathbb{P}(C=a) + \mathbb{P}(X|X=b)\,\mathbb{P}(C=b)$$

The likelihood of a random sample $(X_1, ...X_n)$, the class we do not observe are $(C_1, ...C_n)$:

$$L(\theta) = \prod_i \left[p^{X_i}(1-p)^{(1-X_i)}\pi\right]^{1-C_i} \left[q^{X_i}(1-q)^{(1-X_i)}(1-\pi)\right]^{C_i} \tag{5.4}$$

126. https://en.wikipedia.org/wiki/Multiple_comparisons_problem
127. https://fr.wikipedia.org/wiki/Test_A/B
128. https://en.wikipedia.org/wiki/Holm%E2%80%93Bonferroni_method

The parameters are $\theta = (\pi, p, q)$. We use an algorithm Expectation-Maximization (EM) [129] to determine the parameters. We define at iteration t :

$$\begin{aligned} w_i &= \mathbb{E}_{C_i|X_i,\theta_t}(X_i) \\ &= \frac{p_t^{X_i}(1-p_t)^{1-X_i}\pi_t}{p_t^{X_i}(1-p_t)^{1-X_i}\pi_t + q_t^{X_i}(1-q_t)^{1-X_i}(1-\pi_t)} \end{aligned}$$

We then update the parameters :

$$\begin{aligned} \widehat{\pi} &= \frac{1}{n}\sum_{i=1}^{n} w_i \\ \widehat{p} &= \frac{\sum_{i=1}^{n} w_i X_i}{\sum_{i=1}^{n} w_i} \\ \widehat{q} &= \frac{\sum_{i=1}^{n} (1-w_i) X_i}{\sum_{i=1}^{n} (1-w_i)} \end{aligned}$$

See also Applying the EM Algorithm : Binomial Mixtures [130].

5.2.9 Notebooks

The following notebook produces the figures displayed in this document.

p-values

Compute p-values.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```



```
%matplotlib inline
```

p-value table

```
from scipy.stats import norm
import pandas
import numpy


def pvalue(p, q, N):
    theta   = abs(p-q)
    var     = p*(1-p)
```

(suite sur la page suivante)

129. https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm
130. http://statisticalrecipes.blogspot.fr/2012/04/applying-em-algorithm-binomial-mixtures.html

(suite de la page précédente)

```
    bn      = (2*N)**0.5 * theta / var**0.5
    ret     = (1 - norm.cdf(bn))*2
    return ret

def pvalue_N(p, q, alpha):
    theta   = abs(p-q)
    var     = p*(1-p)
    rev     = abs(norm.ppf (alpha/2))
    N       = 2 * (rev * var**0.5 / theta)** 2
    return int(N+1)

def alphatable(ps, dps, alpha):
    values = []
    for p in ps :
        row=[]
        for dp in dps :
            q = p+dp
            r = pvalue_N(p,q,alpha) if 1 >= q >= 0 else numpy.nan
            row.append (r)
        values.append (row)
    return values

def dataframe(ps,dps,table):
    columns = dps
    df = pandas.DataFrame(data=table, index=ps)
    df.columns = dps
    return df


print ("norm.ppf(0.025)",norm.ppf (0.025)) # -1.9599639845400545
ps  = [0.001, 0.002] + [ 0.05*i for i in range (1,20) ]
dps = [ -0.2, -0.1,  -0.02, -0.01, -0.002, -0.001,
         0.2,  0.1,   0.02,  0.01,  0.002,  0.001, ]
dps.sort()
t = alphatable(ps, dps, 0.05)
dataframe (ps, dps, t)
```

```
norm.ppf(0.025) -1.95996398454
```

p-values in 2D

```
import numpy, matplotlib, random, math
import matplotlib.pyplot as pylab


def matrix_square_root(sigma) :
    eigen, vect = numpy.linalg.eig(sigma)
    dim = len(sigma)
    res = numpy.identity(dim)
    for i in range(0,dim) :
        res[i,i] = eigen[i]**0.5
    return vect * res * vect.transpose()

def chi2_level (alpha = 0.95) :
```

(suite sur la page suivante)

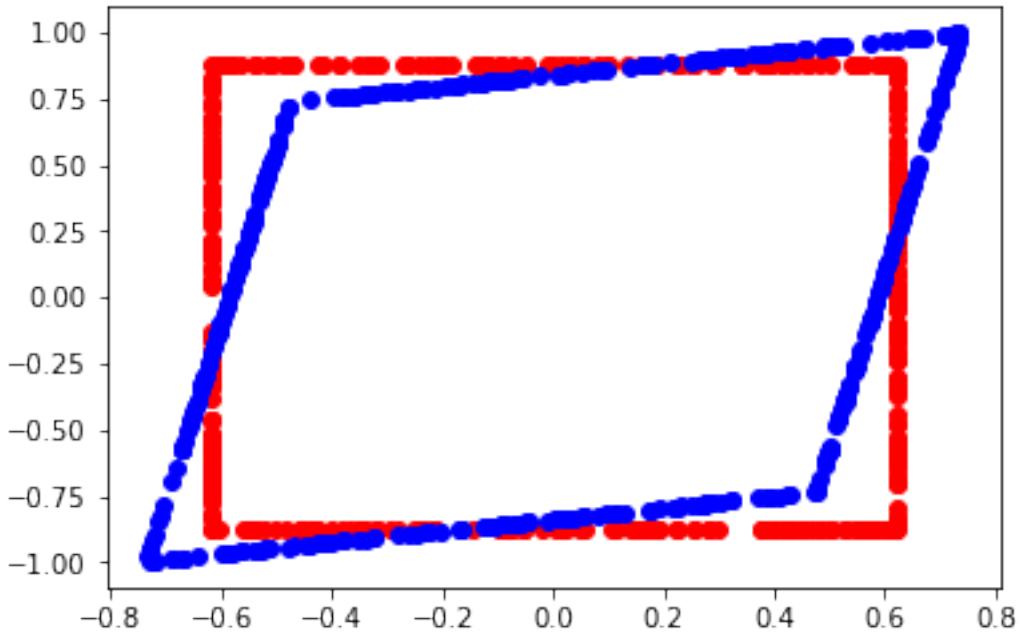
(suite de la page précédente)

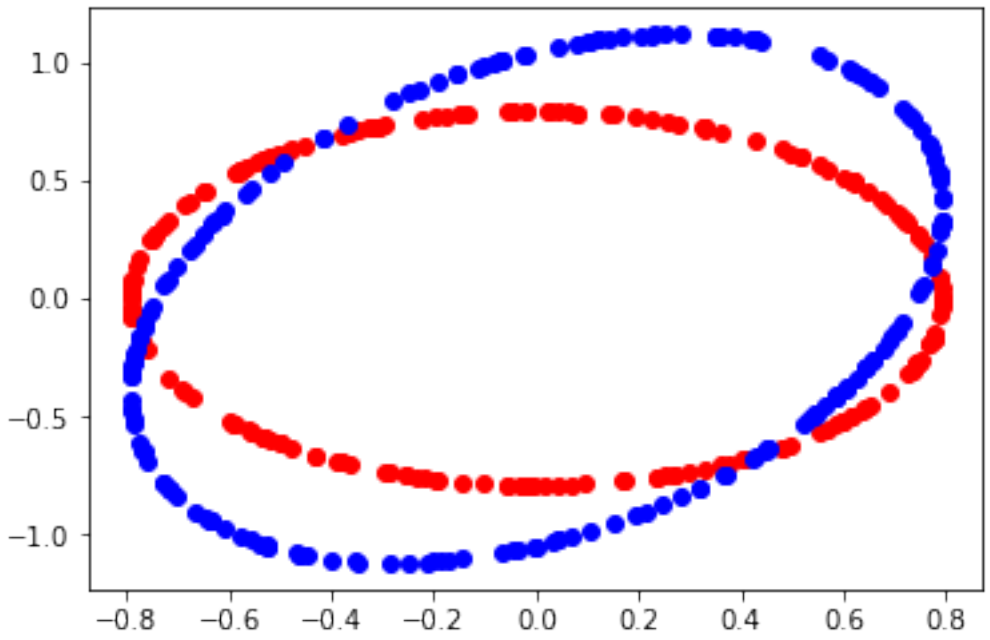
```
    N = 1000
    x = [ random.gauss(0,1) for _ in range(0,N) ]
    y = [ random.gauss(0,1) for _ in range(0,N) ]
    r = map ( lambda c : (c[0]**2+c[1]**2)**0.5, zip(x,y))
    r = list(r)
    r.sort()
    res = r [ int (alpha * N) ]
    return res

def square_figure(mat, a) :
    x = [ ]
    y = [ ]
    for i in range (0,100) :
        x.append ( a * mat[0][0]**0.5 )
        y.append ( (random.random ()-0.5) * a * mat[1][1]**0.5*2 )
        x.append ( -a * mat[0][0]**0.5 )
        y.append ( (random.random ()-0.5) * a * mat[1][1]**0.5*2 )

        y.append ( a * mat[1][1]**0.5 )
        x.append ( (random.random ()-0.5) * a * mat[0][0]**0.5*2 )
        y.append ( -a * mat[1][1]**0.5 )
        x.append ( (random.random ()-0.5) * a * mat[0][0]**0.5*2 )

    pylab.plot(x,y, 'ro')

    x = [ ]
    y = [ ]
    for i in range (0,100) :
        x.append ( a )
        y.append ( (random.random ()-0.5) * a*2 )
        x.append ( -a )
        y.append ( (random.random ()-0.5) * a*2  )

        y.append ( a )
        x.append ( (random.random ()-0.5) * a*2  )
        y.append ( -a )
        x.append ( (random.random ()-0.5) * a*2  )

    xs,ys = [],[]
    for a,b in zip (x,y) :
        ar = numpy.matrix( [ [a], [b] ] ).transpose()
        we = ar * root
        xs.append ( we [0,0] )
        ys.append ( we [0,1] )

    pylab.plot(xs,ys, 'bo')
    pylab.show()

def circle_figure (mat, a) :
    x = [ ]
    y = [ ]
    for i in range (0,200) :
        z = random.random() * math.pi * 2
        i = a * mat[0][0]**0.5 * math.cos(z)
        j = a * mat[0][0]**0.5 * math.sin(z)
        x.append ( i )
        y.append ( j )
```

(suite sur la page suivante)

(suite de la page précédente)

```
    pylab.plot(x,y, 'ro')

    x = [ ]
    y = [ ]
    for i in range (0,200) :
        z = random.random() * math.pi * 2
        i = a * math.cos(z)
        j = a * math.sin(z)
        x.append ( i )
        y.append ( j )

    xs,ys = [],[]
    for a,b in zip (x,y) :
        ar = numpy.matrix( [ [a], [b] ] ).transpose()
        we = ar * root
        xs.append ( we [0,0] )
        ys.append ( we [0,1] )

    pylab.plot(xs,ys, 'bo')
    pylab.show()

level = chi2_level ()
mat   = [ [0.1, 0.05], [0.05, 0.2] ]
npmat = numpy.matrix(mat)
root  = matrix_square_root (npmat)
square_figure (mat, 1.96)
circle_figure (mat, level)
```





p-value ratio

```
import random, math

def densite_gauss (mu, sigma, x) :
    e = -(x - mu)**2 / (sigma**2 * 2)
    d = 1. / ((2*math.pi)**0.5 * sigma)
    return d * math.exp (e)

def simulation_vector (N, mu, sigma) :
    return [ random.gauss(mu,sigma) for n in range(N) ]

def ratio (vector, x, fdensite) :
    under = 0
    above = 0
    fx    = fdensite(x)
    for u in vector :
        f = fdensite (u)
        if f >= fx : above += 1
        else : under += 1
    return float(above) / float (above + under)

x     = 1.96
N     = 10000
mu    = 0
sigma = 1

v = simulation_vector (N, mu, sigma)
g = ratio (v, x, lambda y : densite_gauss (mu, sigma, y) )
print (g)
```

```
0.9513
```

p-values and EM

See Applying the EM Algorithm : Binomial Mixtures [131].

```
from scipy.stats import norm
import random, math


def average_std_deviation (sample) :
    mean = 0.
    var  = 0.
    for x in sample :
        mean += x
        var += x*x
    mean /= len(sample)
    var /= len(sample)
    var -= mean*mean
    return mean,var ** 0.5

def bootsample (sample) :
    n = len(sample)-1
    return [ sample[ random.randint(0,n) ] for _ in sample ]

def bootstrap_difference (sampleX, sampleY, draws = 2000, confidence = 0.05) :
    diff = [ ]
    for n in range (0,draws) :
        if n % 1000 == 0:
            print(n)
        sx = bootsample(sampleX)
        sy = bootsample(sampleY)
        px = sum(sx) * 1.0/ len(sx)
        py = sum(sy) * 1.0/ len(sy)
        diff.append (px-py)
    diff.sort()
    n = int(len(diff) * confidence / 2)
    av = sum(diff) / len(diff)
    return av, diff [n], diff [len(diff)-n]

# generation of a sample

def generate_obs (p) :
    x = random.random()
    if x <= p : return 1
    else : return 0

def generate_n_obs (p, n) :
    return [ generate_obs(p) for i in range (0,n) ]

# std deviation

def diff_std_deviation (px, py) :
    s = px*(1-px) + py*(1-py)
    return px, py, s**0.5

def pvalue(diff, std, N) :
    theta = abs(diff)
```

(suite sur la page suivante)

131. http://statisticalrecipes.blogspot.fr/2012/04/applying-em-algorithm-binomial-mixtures.html

(suite de la page précédente)

```
    bn = (2*N)**0.5 * theta / std
    pv = (1 - norm.cdf(bn))*2
    return pv

def omega_i (X, pi, p, q) :
    np = p * pi     if X == 1 else (1-p)*pi
    nq = q * (1-pi) if X == 1 else (1-q)*(1-pi)
    return np / (np + nq)

def likelihood (X, pi, p, q) :
    np = p * pi     if X == 1 else (1-p)*pi
    nq = q * (1-pi) if X == 1 else (1-q)*(1-pi)
    return math.log(np) + math.log(nq)

def algoEM (sample) :
    p   = random.random()
    q   = random.random()
    pi  = random.random()
    iter   = 0
    while iter < 10 :
        lk = sum ( [ likelihood (x, pi, p, q) for x in sample ] )
        wi  = [ omega_i (x, pi, p, q) for x in sample ]
        sw  = sum(wi)
        pin = sum(wi) / len(wi)
        pn  = sum([ x * w     for x,w in zip (sample,wi) ]) / sw
        qn  = sum([ x * (1-w) for x,w in zip (sample,wi) ]) / (len(wi) - sw)

        pi,p,q = pin,pn,qn
        iter += 1

    lk = sum ( [ likelihood (x, pi, p, q) for x in sample ] )
    return pi,p,q, lk


# mix
p,q   = 0.20, 0.80
pi    = 0.7
N     = 1000
na    = int(N * pi)
nb    = N - na

print("------- sample")
sampleX = generate_n_obs(p, na) + generate_n_obs (q, nb)
random.shuffle(sampleX)
print("ave", p * pi + q*(1-pi))
print("mea", sum(sampleX)*1./len(sampleX))

lk = sum ( [ likelihood (x, pi, p, q) for x in sampleX ] )
print ("min lk", lk, sum (sampleX)*1. / len(sampleX))
res = []
for k in range (0, 10) :
    r = algoEM (sampleX)
    res.append ( (r[-1], r) )
res.sort ()

rows = []
for r in res:
```

(suite sur la page suivante)

(suite de la page précédente)

```
    pi,p,q,lk = r[1]
    rows.append( [p * pi + q*(1-pi)] + list(r[1]))

df = pandas.DataFrame(data=rows)
df.columns = ["average", "pi", "p", "q", "likelihood"]
df
```

```
------- sample
ave 0.38
mea 0.364
min lk -3393.2292120130046 0.364
```

p-value and heavy tail

```
from scipy.stats import norm, zipf
from pyensae.mlhelper import TableFormula
import sys


def generate_n_obs_zipf (tail_index, n) :
    return list(zipf.rvs(tail_index, size=n))

def hill_estimator (sample) :
    sample = list(sample)
    sample.sort(reverse=True)
    end = len(sample)/10
    end = min(end,100)
    s = 0.
    res = []
    for k in range (0,end) :
        s += math.log(sample[k])
        h = (s - (k+1)*math.log(sample[k+1]))/(k+1)
        h = 1./h
        res.append( [k, h] )
    return res

# mix
tail_index  = 1.05
N           = 10000

sample = generate_n_obs_zipf(tail_index, N)
sample[:5]
```

```
[6, 1, 9437, 1046824677, 1135742]
```

```
def draw_variancen(sample) :
    avg = 0.
    std = 0.
    n   = 0.
    w   = 1.
    add = []
    for i,x in enumerate(sample) :
        x    = float (x)
```

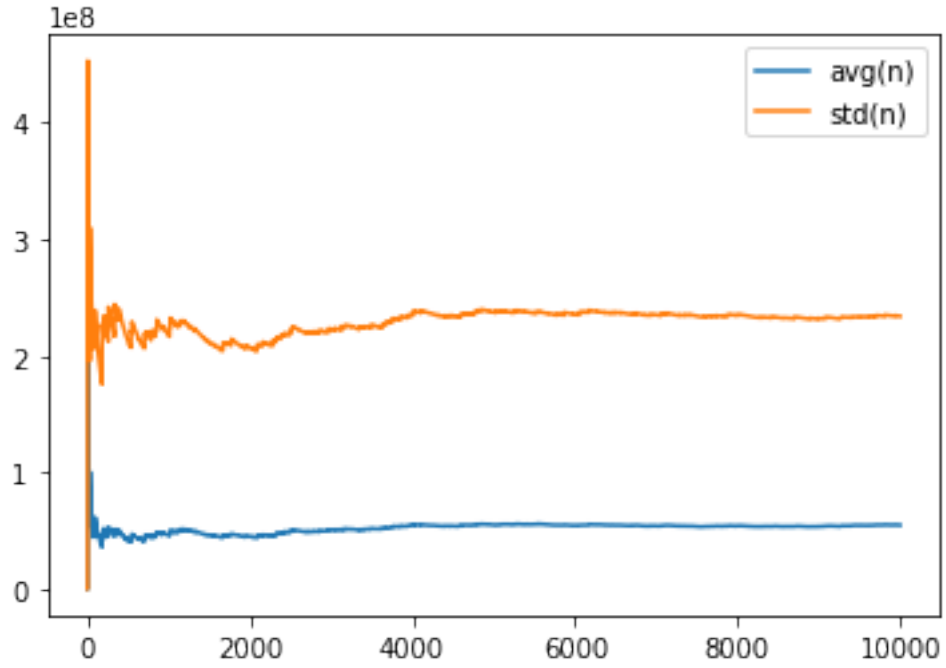
(suite sur la page suivante)

(suite de la page précédente)

```
        avg += x * w
        std += x*x * w
        n   += w
        val  = (std/n - (avg/n)**2)**0.5
        add.append ( [ i, avg/n, val] )

    print(add[-1])
    table = TableFormula(add, columns=["index", "avg(n)", "std(n)"])
    return table.graph_XY ( [
            [ lambda v: v["index"], lambda v: v["avg(n)"], "avg(n)"],
            [ lambda v: v["index"], lambda v: v["std(n)"], "std(n)"]],
                    marker=False, link_point=True)

draw_variancen(sample)
```

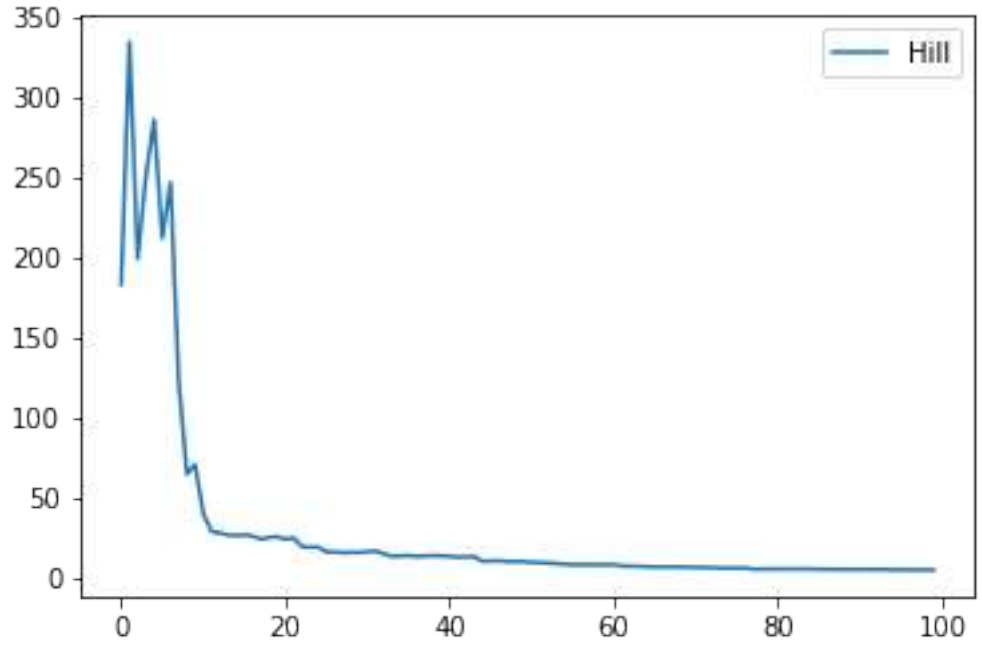
```
[9999, 54761151.7614, 233970252.48746485]
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x21b16175c88>
```




```
def draw_hill_estimator (sample) :
    res = hill_estimator(sample)
    table = TableFormula(res, columns=["d", "hill"])
    return table.graph_XY ( [
            [ lambda v: v["d"], lambda v: v["hill"], "Hill"],],
                    marker=False, link_point=True)

draw_hill_estimator(sample)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x21b12ca6470>
```



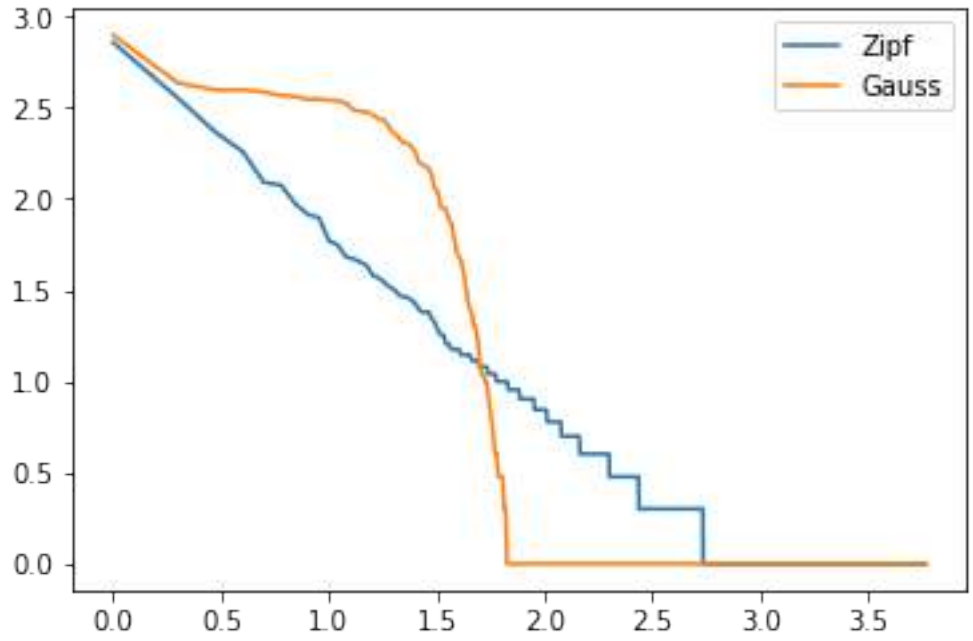

```
def draw_heavy_tail (sample) :
    table   = TableFormula ([ [_] for _ in sample ], columns=["obs"] )
    std = 1

    normal = norm.rvs(size = len(sample))
    normal = [ x*std  for x in normal ]
    nortbl = TableFormula ([ [_] for _ in normal ], columns=["obs"] )
    nortbl.addc ("iobs", lambda v : int(v ["obs"] * 10))
    histon  = nortbl.fgroupby ( lambda v :v["iobs"], [ lambda v : v["iobs"] ], ["nb"],
↪ [ len ] )
    histon.sort (lambda v : v["nb"], reverse = True)

    histo   = table.fgroupby ( lambda v :v["obs"], [ lambda v : v["obs"] ], ["nb"], [␣
↪len ] )

    histo.sort (lambda v : v["nb"], reverse = True)
    histo.reset_index(drop=True, inplace=True)
    histo["index"] = histo.index + 1

    vec  = list(histon["nb"])
    vec += [0,] * len(histo)
    histo.add_column_vector("nb_normal", vec [:len(histo) ] )

    histo.addc ("log(index)", lambda v : math.log(v ["index"]) / math.log(10) )
    histo.addc ("log(nb)", lambda v : math.log(v ["nb"]) / math.log(10) )
    histo.addc ("log(nb_normal)", lambda v : math.log(v ["nb_normal"]) / math.log(10)␣
↪if v["nb_normal"] > 0 else 0)
    return histo.graph_XY ( [
            [ lambda v: v["log(index)"], lambda v: v["log(nb)"], "Zipf"],
            [ lambda v: v["log(index)"], lambda v: v["log(nb_normal)"], "Gauss"], ],
                    marker=False, link_point=True)
```

(suite sur la page suivante)

(suite de la page précédente)

```
draw_heavy_tail(sample)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x21b163c7f98>
```



5.2.10 Bibliographie

- p-Value and Statistical Practice [132]
- An investigation of the false discovery rate and the misinterpretation of p-values [133]
- Holm–Bonferroni method [134]

132. http://www.stat.columbia.edu/~gelman/research/published/pvalues3.pdf
133. http://rsos.royalsocietypublishing.org/content/royopensci/1/3/140216.full.pdf
134. https://en.wikipedia.org/wiki/Holm%E2%80%93Bonferroni_method

CHAPITRE 6

Distances

6.1 Distance d'édition

Les distances d'édition permettent de comparer deux mots entre eux ou plus généralement deux séquences de symboles entre elles. L'usage le plus simple est de trouver, pour un mot mal orthographié, le mot le plus proche dans un dictionnaire, c'est une option proposée dans la plupart des traitements de texte. La distance présentée est la distance de Levenshtein [135] (voir *[Levenstein1966]* (page **??**)) Elle est parfois appelée Damerau Levenstein Matching (DLM) [136] (voir également *[Damerau1964]* (page **??**)). Cette distance fait intervenir trois opérations élémentaires :

- comparaison entre deux caractères
- insertion d'un caractère
- suppression d'un caractère

Pour comparer deux mots, il faut construire une méthode associant ces trois opérations afin que le premier mot se transforme en le second mot. L'exemple suivant utilise les mots `idstzance` et `distances`, il montre une méthode permettant de passer du premier au second. La distance sera la somme des coûts associés à chacune des opérations choisies. La comparaison entre deux lettres identiques est en général de coût nul, toute autre opération étant de coût strictement positif.

Tableau 1 – distance d'édition

mot 1	mot 2	opération	coût
i	d	comparaison entre `i` et `d`	1
d	i	comparaison entre `d` et `i`	1
s	s	comparaison entre `s` et `s`	0
t	t	comparaison entre `t` et `t`	0
z		suppression de `z`	1
a	a	comparaison entre `a` et `a`	0
n	n	comparaison entre `n` et `n`	0
c	c	comparaison entre `c` et `c`	0
e	e	comparaison entre `e` et `e`	0
	s	insertion de `s`	1
		somme	4

135. https://fr.wikipedia.org/wiki/Distance_de_Levenshtein
136. https://fr.wikipedia.org/wiki/Distance_de_Damerau-Levenshtein

Pour cette distance d'édition entre les mots `idstzance` et `distances`. La succession d'opérations proposée n'est pas la seule qui permettent de construire le second mot à partir du premier mais c'est la moins coûteuse.

6.1.1 Définition et propriétés

Définition

Tout d'abord, il faut définir ce qu'est un mot ou une séquence :

Définition D1 : mot

On note $\mathcal{C}$ l'espace des caractères ou des symboles. Un mot ou une séquence est une suite finie de $\mathcal{C}$. On note $\mathcal{S}_\mathcal{C} = \cup_{k=1}^{\infty} C^k$ l'espace des mots formés de caractères appartenant à $\mathcal{C}$.

On peut définir la distance d'édition :

Définition D2 : distance d'édition

La distance d'édition d sur $\mathcal{S}_\mathcal{C}$ est définie par :

$$\begin{array}{rccc} d: & \mathcal{S}_\mathcal{C} \times \mathcal{S}_\mathcal{C} & \longrightarrow & \mathbb{R}^+ \\ & (m_1, m_2) & \longrightarrow & \min\limits_{\substack{O \text{ séquence} \\ \text{d'opérations}}} d(m_1, m_2, O) \end{array}$$

La distance est le coût de la transformation du mot m_1 en m_2 la moins coûteuse. Il reste à démontrer que cette distance en est bien une puis à proposer une méthode de calcul plus rapide que celle suggérée par cette définition.

Propriétés

Ce paragraphe a pour objectif de démontrer que la *distance* (page 158) en est bien une.

Définition D3 : distance entre caractères

Soit $\mathcal{C}' = \mathcal{C} \bigcup \{.\}$ l'ensemble des caractères ajouté au caractère vide $..$ On note $c : (\mathcal{C}')^2 \longrightarrow \mathbb{R}^+$ la fonction coût définie comme suit :

$$\forall (x,y) \in (\mathcal{C}')^2,\ c(x,y) \text{ est le coût } \begin{cases} \text{d'une comparaison} & \text{si } (x,y) \in (\mathcal{C})^2 \\ \text{d'une insertion} & \text{si } (x,y) \in (\mathcal{C}) \times \{.\} \\ \text{d'une suppression} & \text{si } (x,y) \in \{.\} \times (\mathcal{C}) \\ 0 & \text{si } (x,y) = (\{.\}, \{.\}) \end{cases}$$

On note $\mathcal{S}^2_{\mathcal{C}'} = \cup_{n=1}^{\infty} \left(\mathcal{C}'^2\right)^n$ l'ensemble des suites finies de $\mathcal{C}'$.

Pour modéliser les transformations d'un mot vers un autre, on définit pour un mot m un *mot acceptable* :

Définition D4 : mot acceptable

Soit $m = (m_1, \ldots, m_n)$ un mot tel qu'il est défini précédemment. Soit $M = (M_i)_{i \geqslant 1}$ une suite infinie de caractères, on dit que M est un mot acceptable pour m si et seulement si la sous-suite extraite de M contenant tous les caractères différents de $\{.\}$ est égal au mot m. On note $acc(m)$ l'ensemble des mots acceptables pour le mot m.

Par conséquent, tout mot acceptable m' pour le mot m est égal à m si on supprime les caractères $\{.\}$ du mot m'. En particulier, à partir d'un certain indice, m' est une suite infinie de caractères $\{.\}$. Il reste alors à exprimer la définition de la distance d'édition en utilisant les mots acceptables :

Définition D5 : distance d'édition

Soit c la *distance d'édition* (page 158), d définie sur $\mathcal{S}_\mathcal{C}$ est définie par :

$$\begin{array}{rccl} d: & \mathcal{S}_\mathcal{C} \times \mathcal{S}_\mathcal{C} & \longrightarrow & \mathbb{R}^+ \\ & (m_1, m_2) & \longrightarrow & \min\left\{\sum_{i=1}^{+\infty} c\left(M_1^i, M_2^i\right) \mid (M_1, M_2) \in acc(m_1) \times acc(m_2)\right\} \end{array} \tag{6.1}$$

Il est évident que la série $\sum_{i=1}^{+\infty} c\left(M_1^i, M_2^i\right)$ est convergente. La :ref'distance de caractères <edition_distance_definition_1>' implique que les distance d'édition définies en *1* (page 158) et *2* (page 159) sont identiques.

Théorème T1 : distance d'édition

Soit c et d les fonctions définies respectivement par *(1)* (page **??**) et *(2)* (page **??**), alors :

c est une distance sur $\mathcal{C} \iff d$ est une distance sur $\mathcal{S}_\mathcal{C}$

On cherche d'abord à démontrer que

c est une distance sur $\mathcal{C}' \impliedby d$ est une distance sur $\mathcal{S}_\mathcal{C}$

Cette assertion est évidente car, si (m_1, m_2) sont deux mots de un caractère, la distance d sur $\mathcal{S}_\mathcal{C}$ définit alors la distance c sur $\mathcal{C}'$.

On démontre ensuite que :

c est une distance sur $\mathcal{C}' \implies d$ est une distance sur $\mathcal{S}_\mathcal{C}$

Soient deux mots (m_1, m_2), soit $(M_1, M_2) \in acc(m_1) \times acc(m_2)$, comme c est une distance sur $\mathcal{C}'$ alors $d(M_1, M_2) = d(M_2, M_1)$.

D'où, d'après la définition *2* (page 159) :

$$d(m_1, m_2) = d(m_2, m_1) \tag{6.2}$$

Soit $(N_1, N_2) \in acc(m_1) \times acc(m_2)$ tels que $d(m_1, m_2) = d(N_2, N_1)$ alors :

$$\begin{array}{rcl} d(m_1, m_2) = 0 & \implies & d(N_1, N_2) = 0 \\ & \implies & \sum_{i=1}^{+\infty} c\left(N_1^i, N_2^i\right) = 0 \\ & \implies & \forall i \geqslant 1,\ N_1^i = N_2^i \\ & \implies & N_1 = N_2 \\ d(m_1, m_2) = 0 & \implies & m_1 = m_2 \end{array}$$

Il reste à démontrer l'inégalité triangulaire. Soient trois mots (m_1, m_2, m_3), on veut démontrer que $d(m_1, m_3) \leqslant d(m_1, m_2) + d(m_2, m_3)$. On définit :

$$\begin{array}{rcl} (N_1, N_2) \in acc(m_1) \times acc(m_2) & \text{tels que} & d(m_1, m_2) = d(N_1, N_2) \\ (P_2, P_3) \in acc(m_2) \times acc(m_3) & \text{tels que} & d(m_2, m_3) = d(P_2, P_3) \\ (O_1, O_3) \in acc(m_1) \times acc(m_3) & \text{tels que} & d(m_1, m_3) = d(O_1, O_3) \end{array}$$

Mais il est possible, d'après la définition d'un *mot acceptable* (page 158) d'insérer des caractères $\{.\}$ dans les mots $N_1, N_2, P_2, P_3, O_1, O_3$ de telle sorte qu'il existe $(M_1, M_2, M_3) \in acc(m_1) \times \in acc(m_2) \times \in acc(m_3)$ tels que :

$$\begin{array}{l} d(m_1, m_2) = d(M_1, M_2) \\ d(m_2, m_3) = d(M_2, M_3) \\ d(m_1, m_3) = d(M_1, M_3) \end{array}$$

Or comme la fonction c est une distance sur $\mathcal{C}'$, on peut affirmer que : $d(M_1, M_3) \leqslant d(M_1, M_2) + d(M_2, M_3)$. D'où :

begin{eqnarray} dpa{m_1,m_3} infegal dpa{m_1,m_2} + d pa{m_2,m_3} label{edit_demo_eq_3} end{eqnarray}

Les assertions 1, 2, 3 montrent que d est bien une distance. Le tableau suivant illustre la démonstration pour les suites M_1, M_2, M_3 pour les mots et les mots `idtzance`, `tonce`, `distances`.

M_1	i	d		t	z	a	n	c	e	
M_2				t		o	n	c	e	
M_3	d	i	s	t		a	n	c	e	s

La distance d'édition *2* (page 159) ne tient pas compte de la longueur des mots qu'elle compare. On serait tenté de définir une nouvelle distance d'édition inspirée de la précédente :

Définition D6 : distance d'édition étendue

Soit *d^** la distance d'édition définie en *2* (page 159) pour laquelle les coûts de comparaison, d'insertion et de suppression sont tous égaux à 1. La distance d'édition d' sur $\mathcal{S}_\mathcal{C}$ est définie par :

$$\begin{array}{rccl} d' : & \mathcal{S}_\mathcal{C} \times \mathcal{S}_\mathcal{C} & \longrightarrow & \mathbb{R}^+ \\ & (m_1, m_2) & \longrightarrow & d'(m_1, m_2) = \dfrac{d^*(m_1, m_2)}{\max\{l(m_1), l(m_2)\}} \end{array}$$

où $l(m)$ est la longueur du mot m

Le tableau suivant donne un exemple pour lequel l'inégalité triangulaire n'est pas vérifiée. La fonction d^* n'est donc pas une distance.

mot 1	mot 2	distance : d^*	distance d'
APPOLLINE	APPOLINE	1	1 / 9
APPOLLINE	APOLLINE	1	1 / 9
APOLLINE	APPOLINE	2	2 / 8

Par conséquent : $d(APOLLINE, APPOLINE) > d(APOLLINE, APPOLLINE) + d(APPOLLINE, APPOLINE)$ et la la fonction d^* ne vérifie pas l'inégalité triangulaire.

6.1.2 Factorisation des calculs

La définition de la distance d'édition ne permet pas d'envisager le calcul de la distance dans un temps raisonnable. Il est possible néanmoins d'exprimer cette distance d'une autre manière afin de résoudre ce problème (voir *[Wagner1974]* (page **??**)). On définit la suite suivante :

Définition D7 : distance d'édition tronquée

Soient deux mots (m_1, m_2), on définit la suite :

$$\left(d_{i,j}^{m_1,m_2}\right)_{\substack{0 \leqslant i \leqslant n_1 \\ 0 \leqslant j \leqslant n_2}} \left(= (d_{i,j})_{\substack{0 \leqslant i \leqslant n_1 \\ 0 \leqslant j \leqslant n_2}} \text{ pour ne pas alourdir les notations} \right)$$

Par :

$$\left\{ \begin{array}{l} d_{0,0} = 0 \\ d_{i,j} = \min \left\{ \begin{array}{lcl} d_{i-1,j-1} & + & \text{comparaison}\left(m_1^i, m_2^j\right), \\ d_{i,j-1} & + & \text{insertion}\left(m_2^j\right), \\ d_{i-1,j} & + & \text{suppression}\left(m_1^i\right) \end{array} \right. \end{array} \right.$$

Cette suite tronquée permet d'obtenir le résultat de la propriété suivante :

Propriété P1 : calcul rapide de la distance d'édition

La suite définie par *3* (page 160) vérifie $d(m_1, m_2) = d_{n_1,n_2}$ où d est la distance d'édition définie en *1* (page 158) ou *2* (page 159).

Cette factorisation des calculs est illustrée par les tableaux de cette figure. La démonstration s'effectue par récurrence, la définition *3* (page 160) est bien sûr équivalente *1* (page 158) pour des mots de longueur un. On suppose donc que ce résultat est vrai pour un couple de mots (m_1, m_2) de longueur (l_1, l_2) vérifiant $l_1 \leqslant i$ et *l_2 infegal j* avec au plus une égalité. Soit m un mot, on note n le nombre de lettres qu'il contient. On note $m(l)$ le mot formé des l premières lettres de m. Alors :

$$\begin{array}{rcl} d_{i,j}^{m_1,m_2} & = & d\left(m_1(i), m_2(j)\right) \\ d\left(m_1(i), m_2(j)\right) & = & \min \left\{ \begin{array}{lcl} d\left(m_1(i-1), m_2(j-1)\right) & + & \text{comparaison}\left(m_{1,i}, m_{2,j}\right), \\ d\left(m_1(i), m_2(j-1)\right) & + & \text{insertion}\left(m_{2,j}\right), \\ d\left(m_1(i-1), m_2(j)\right) & + & \text{suppression}\left(m_{1,i}\right) \end{array} \right\} \end{array}$$

Le calcul factorisé de la distance d'édition entre deux mots de longueur l_1 et l_2 a un coût de l'ordre $O(l_1 l_2)$. Il est souvent illustré par un tableau comme celui de la figure suivante qui permet également de retrouver la meilleure séquence d'opérations permettant de passer du premier mot au second.

↘ dans ce sens, c'est une comparaison

$\longrightarrow j$ dans ce sens, c'est une insertion

↓ i dans ce sens, c'est une suppression

		0	d	i	s	t	a	n	c	e	s
	0	0	1	2	3	4	5	6	7	8	9
i	1	1	2	2	3	4	5	6	7	8	9
d	2	2	3	3	4	5	6	7	8	9	10
s	3	3	4	4	5	6	7	8	9	10	11
t	4	4	5	5	6	7	8	9	10	11	12
z	5	5	6	6	7	8	9	10	11	12	13
a	6	6	7	7	8	9	10	11	12	13	14
n	7	7	8	8	9	10	11	12	13	14	15
c	8	8	9	9	10	11	12	13	14	15	16
e	9	9	10	10	11	12	13	14	15	16	17

Chaque case (i, j) contient la distance qui sépare les i premières lettres du mot 1 des j premières lettres du mot 2 selon le chemin ou la méthode choisie. La dernière case indique la distance qui sépare les deux mots quel que soit le chemin choisi.

6.1.3 Extension de la distance d'édition

Jusqu'à présent, seuls trois types d'opérations ont été envisagés pour constuire la distance d'édition, tous trois portent sur des caractères et aucunement sur des paires de caractères. L'article *[Kripasundar1996]* (page **??**) (voir aussi *[Seni1996]* (page **??**) suggère d'étendre la définition *3* (page 160) aux permutations de lettres :

Définition D8 : distance d'édition tronquée étendue

Soit deux mots (m_1, m_2), on définit la suite :

$$\left(d_{i,j}^{m_1,m_2}\right)_{\substack{0 \leqslant i \leqslant n_1 \\ 0 \leqslant j \leqslant n_2}} \left(= \left(d_{i,j}\right)_{\substack{0 \leqslant i \leqslant n_1 \\ 0 \leqslant j \leqslant n_2}} \text{ pour ne pas alourdir les notations} \right)$$

par :

$$\left\{ \begin{array}{l} d_{0,0} = 0 \\ d_{i,j} = \min \left\{ \begin{array}{lcl} d_{i-1,j-1} & + & \text{comparaison } \left(m_1^i, m_2^j\right), \\ d_{i,j-1} & + & \text{insertion } \left(m_2^j, i\right), \\ d_{i-1,j} & + & \text{suppression } \left(m_1^i, j\right), \\ d_{i-2,j-2} & + & \text{permutation } \left(\left(m_1^{i-1}, m_1^i\right), \left(m_2^{j-1}, m_2^j\right)\right) \end{array} \right. \end{array} \right.$$

La distance d'édition cherchée est toujours $d(m_1, m_2) = d_{n_1,n_2}$ mais la démonstration du fait que d est bien une distance ne peut pas être copiée sur celle du théorème *1* (page 159) mais sur les travaux présentés dans l'article *[Wagner1974]* (page **??**).

6.1.4 Apprentissage d'une distance d'édition

L'article *[Waard1995]* (page **??**) suggère l'apprentissage des coûts des opérations élémentaires associées à une distance d'édition (comparaison, insertion, suppression, permutation, ...). On note l'ensemble de ces coûts ou paramètres $\Theta = (\theta_1, \ldots, \theta_n)$. On considère deux mots X et Y, la distance d'édition $d(X, Y)$ est une fonction linéaire des coûts. Soit $D = ((X_1, Y_1), \ldots, (X_N, Y_N))$ une liste de couple de mots pour lesquels le résultat de la distance d'édition est connu et noté $(c_1, \ldots, c_N)$, il est alors possible de calculer une erreur s'exprimant sous la forme :

$$E = \sum_{i=1}^{N} \left(d(X_i, Y_i) - c_i\right)^2 = \sum_{i=1}^{N} \left(\sum_{k=1}^{n} \alpha_{ik}(\Theta)\ \theta_k - c_i\right)^2$$

Les coefficients $\alpha_{ik}(\Theta)$ dépendent des paramètres Θ car la distance d'édition correspond au coût de la transformation de moindre coût d'après la définition :ref‘2 <defition_distance_edition_2>‘, $\alpha_{ik}(\Theta)$ correspond au nombre de fois que le paramètre θ_k intervient dans la transformation de moindre coût entre X_i et Y_i. Cette expression doit être minimale afin d'optenir les coûts Θ optimaux. Toutefois, les coûts θ_k sont tous strictement positifs et plutôt que d'effectuer une optimisation sous contrainte, ces coûts sont modélisés de la façon suivante :

$$E = \sum_{i=1}^{N} \left(\sum_{k=1}^{n} \alpha_{ik}(\Omega)\ \frac{1}{1 + e^{-\omega_k}} - c_i\right)^2$$

Les fonctions $\alpha_{ik}(\Omega)$ ne sont pas dérivable par rapport Ω mais il est possible d'effectuer une optimisation sans contrainte par descente de gradient. Les coûts sont donc appris en deux étapes :

Algorithme A1 : Apprentissage d'une distance d'édition

Les notations sont celles utilisés pour l'équation *(6)* (page **??**). Les coûts Ω sont tirés aléatoirement.

estimation

Les coefficients $\alpha_{ik}(\Omega)$ sont calculées.

calcul du gradient

Dans cette étape, les coefficients $\alpha_{ik}(\Omega)$ restent constants. Il suffit alors de minimiser la fonction dérivable $E(\Omega)$ sur $\mathbb{R}^n$, ceci peut être effectué au moyen d'un algorithme de descente de gradient similaire à ceux utilisés pour les réseaux de neurones.

Tant que l'erreur $E(\Omega)$ ne converge pas, on continue. L'erreur E diminue jusqu'à converger puisque l'étape qui réestime les coefficients $\alpha_{ik}(\Omega)$, les minimise à $\Omega = (\omega_1, \ldots, \omega_n)$ constant.

6.1.5 Bibliographie

CHAPITRE 7

Graphes

Les graphes sont très utilisés en informatique. Arbre de décision en machine learning, réseaux sociaux, recommandations, agencement de tâche, optimisation de flux, la structure de graphe apparaît naturellement dans de nombreux domaines.

7.1 Distance between two graphs

This page gathers some thoughts about comparing two graphs without any cycle. This problem is known as *graph similarity* or graph matching [137]. One of the solution is the Graph Edit Distance [138], a better solution is described in *[Blondel2004]* (page **??**). You can also read Graph similarity [139].

7.1.1 Definitions

The first approach is implemented in module `graph_distance`. Example of use :

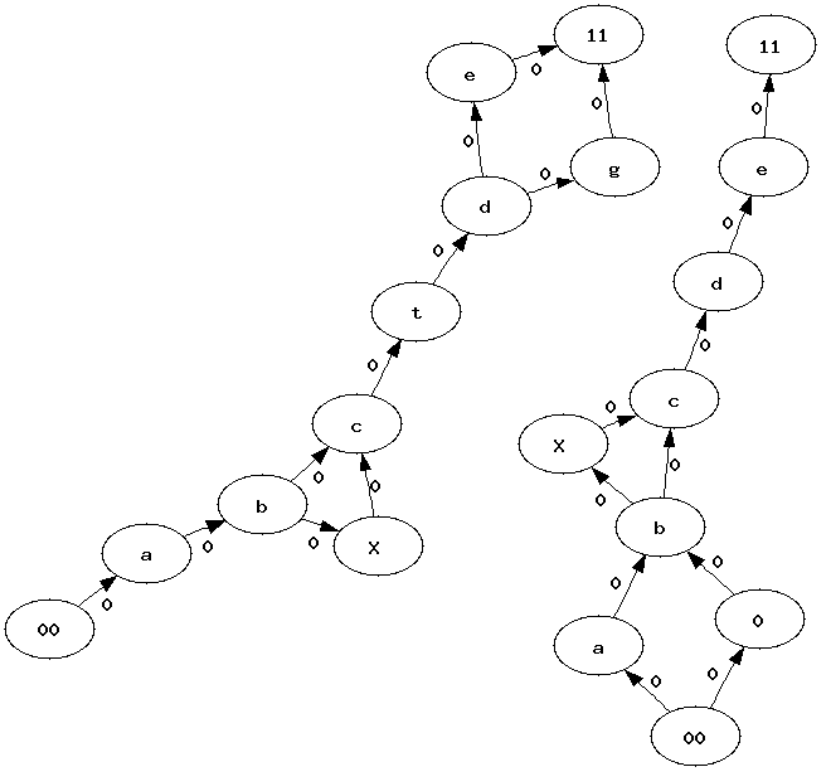
137. https://en.wikipedia.org/wiki/Graph_matching
138. https://en.wikipedia.org/wiki/Graph_edit_distance
139. http://www.cs.uoi.gr/~pvassil/downloads/GraphDistance/LauraZager.pdf

```
graph1 = [ ("a","b"), ("b","c"), ("b","d"), ("d","e"), \
           ("e","f"), ("b","f"), ("b","g"), ("f", "g"),
           ("a","g"), ("a","g"), ("c","d"), ("d", "g"),
           ("d","h"), ("aa","h"), ("aa","c"), ("f", "h"), ]
graph2 = copy.deepcopy(graph1) + \
         [ ("h", "m"), ("m", "l"), ("l", "C"), ("C", "r"),
           ("a", "k"), ("k", "l"), ("k", "C"),
           ]

graph1 = Graph(graph1)
graph2 = Graph(graph2)

distance, graph = graph1.distance_matching_graphs_paths(graph2, use_min=False,␣
↪store=store)
```

graph is the merged graph mentioned below.

7.1.2 Problem

This *graph distance* aims at computing a distance between graphs but also to align two graphs and to merge them into a single one. For example, let's consider the following graphs :



We would like to merge them and to know which vertices were merged, which ones were added and deleted. The following ideas and algorithm are only applicable on graphs without cycles. To simplify, we assume there are only one root and one leave. If there are mulitple, we then create a single root we connect to all the existing ones. We do the same for the unique leave we create if there are multiple. It will have all the existing ones as predecessors. We also assume each vertex and each edge holds a label used during the matching. It is better to match vertices or edges holding the same label. A weight can be introduced to give more important to some elements (vertex, edge).

7.1.3 First approach

Step 1 : edit distance

The main idea consists in using Levenstein's edit distance [140]. This algorithm applies on sequences but not on graphs. But because both graphs do not contain any cycle, we can extract all paths from them. Every path starts with the same vertex - the only root - and ends with the same one - the only leave -. We also consider each edge or vertex as an element of the sequence. Before describing the edit distance, let's denote p_1 as a path from the first graph, p_2 as a path from the second one. $p_k(i)$ is the element *i* of this sequence. Following Levenstein description, we denote *d(i,j)* as the distance between the two subsequences $p_1(1..i), p_2(1..j)$. Based on that, we use an edit distance defined as follows :

$$d(i,j) = \min \begin{cases} d(i-1,j) + insertion(p_1(i)) \\ d(i,j-1) + insertion(p_2(j)) \\ d(i-1,j-1) + comparison(p_1(i), p_2(j)) \end{cases}$$

First of all, we are not only interested in the distance but also in the alignment which would imply to keep which element was chosen as a minimum for each *d(i,j)*. If we denote n_k the length of path *k*, then $d(n_1, n_2)$ is the distance we are looking for.

Second, if two paths do not have the same length, it implies some elements could be compared between each others even if one is an edge and the other one is a vertex. This case is unavoidable if two paths have different lengths.

Third, the weight we use for the edit distance will be involved in a kind of tradeof : do we prefer to boost the structure or the label when we merge the graphs. Those weights should depend on the task, whether or not it is better to align vertices with the same label or to keep the structure. Here are the chosen weights :

operation	weight	condition
insertion(c)		*w(c)*, weight held by the edge or the vertex
comparison(a,b)	0	if vertices *a* and *b* share the same label
comparison(a,b)	0	if edges *a* and *b* share the same label and if vertices at both ends share the same label
comparison(a,b)	$w(a) + w(b)$	if edges *a* and *b* share the same label and if vertices at both ends do not share the same label
comparison(a,b)	$\frac{w(a)+w(b)}{2}$	if *a* and *b* do not share the same kind
comparison(a,b)	$\frac{3(w(a)+w(b))}{2}$	if *a* and *b* share the same kind but not the label

Kind means in this context edge or vertex. In that case, we think that sharing the same kind but not the same label is the worst case scenario. Those weights avoid having multiples time the same distance between two random paths which will be important during the next step. In fact, because the two graphs do not contain cycles, they have a finite number of paths. We will need to compute all distances between all possible pairs. The more distinct values we have for a distance between two paths, the better it is.

Step 2 : Kruskal kind (bijection on paths)

Among all possible distances we compute between two paths, some of them might be irrelevant. If for some reasons, there is an edge which connects the root to the leave, computing the edit distance between this short path and any other one seems weird. That's why we need to consider a kind of paths association. We need to associate a path from a graph to another from the other graph and the association needs to be a bijection assuming two close paths will have a low distance.

140. https://en.wikipedia.org/wiki/Levenshtein_distance

After the first step, we ended up with a matrix containing all possible distances. We convert this matrix into a graph where each path is a vertex, each distance is a weighted edge. We use a kind of Kruskal algorithm to remove heavy weighted edges until we get a kind of bijection :

— We sort all edges by weight (reverse order).
— We remove the first ones until we get an injection on both sides : a path from a graph must be associated to only one path.

Basically, some paths from the bigger graph will not be teamed up with another path.

Step 3 : Matching

Now that we have a kind of bijection between paths, it also means we have a series of alignments between paths : one from the first graph, one from the second graph and an alignment between them computed during the step. We build two matrices, one for the edges M_e, one for the vertices M_v defined as follows :

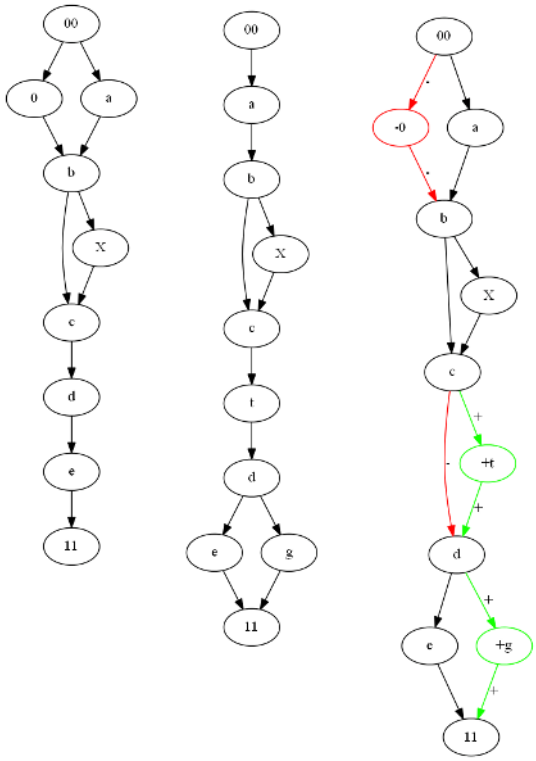
— $M_e(i, j)$ contains the number of times edge *i* from graph 1 is associated to edge *j* from graph 2 among all paths associated by the previous step.
— $M_v(i, j)$ contains the same for the vertices.

Step 4 : Kruskal kind, the return (bijection on edges and vertices)

We now have two matrices which contains pretty much the same information as we have in step 2 : each element is the number of times an edge or a vertex was associated with an edge or a vertex of the other graph. We use the same algorithm until we get a kind of bijection between vertices or edges from both matrices.

Step 5 : Merging the two graphs

Once we finalized the previous steps, we know which vertices and edges will be associated with vertices and edges from the other graph. What's left is to add the left over to the picture which is shown by next Figure :



Red and symbol - means deleted from graph~1 and not present in graph 2. Green and symbol + means not present in graph 1 and added in graph 2. The black pieces remains unchanged.

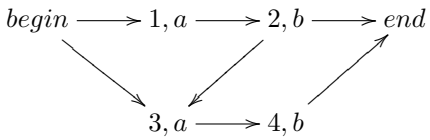
The main drawback of this algorithm is its speed. It is very time consuming. We need to compute distances between all paths which is ok when graphs are small but very long when graphs are bigger. Many paths share the same beginning and we could certainly avoid wasting time computing edit distances between those paths.

7.1.4 Distance between graphs

We defined a distance between two sequences based on the sum of operations needed to switch from the first sequence to the second one, we can follow the same way here. The alignment we were able to build between two graphs shows insertions, deletions and comparisons of different edges of vertices. By giving a weight of each kind, we can sum them to build the distance we are looking for. We use the same weights we defined to compute the alignment between two paths from both graphs. Let's defined an aligned graph *G = { (a,b) }*, *G* is the set of edges and vertices of the final graph, *a*, *b* are an edge of a vertex from the first graph for *a* and from the second graph for *b*. *a* or *b* can be null. We also defined $insertion(a) = comparison(\emptyset, a)$.

$$d(G_1, G_2) = \sum_{\substack{a \in G_1 \cup \emptyset \\ b \in G_2 \cup \emptyset}} comparison(a, b) \mathbb{1}\!\!1_{\{(a,b) \in G\}}$$

It is obvioulsy symmetric. To proove it verifies $d(G_1, G_2) = 0 \iff G_1 = G_2$, we could proove that every path from G_1 will be associated to itself during the first step. It is not necessarily true because two different paths could share the same sequence of labels. Let's consider the following example :



This graph contains three paths :

$path1$	$1, 2$	ab
$path2$	$3, 4$	ab
$path3$	$1, 2, 3, 4$	$abab$

The matrix distance between paths will give (*x> 0*) :

$$\begin{pmatrix} 0 & \mathbf{0}. & x \\ \mathbf{0}. & 0 & x \\ x & x & \mathbf{0}. \end{pmatrix}$$

The bolded values $\mathbf{0}.$ represent one possible association between paths which could lead to the possible association between vertices :

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

In that particular case, the algorithm will not return a null distance mostly because while aligning sequences, we do not pay too much attention to the local structure. One edge could be missing from the alignment. We could try to correct that by adding some cost when two vertices do not have the number of input or output edges instead of considering only the labels.

7.1.5 Second approach : faster

No implemented yet.

7.1.6 Bibliography

CHAPITRE 8

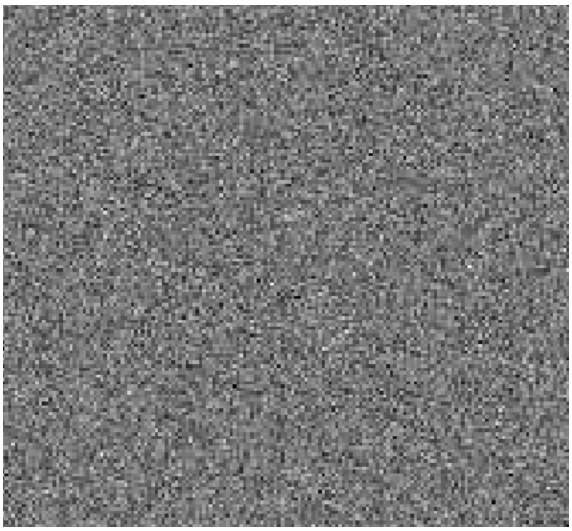
Algorithmes

Le machine learning propose un grand nombre de modèles pour des problèmes standardisés. Au delà, il faut savoir être inventifs, c'est-à-dire la grande majorité des cas.

8.1 Détection de segments

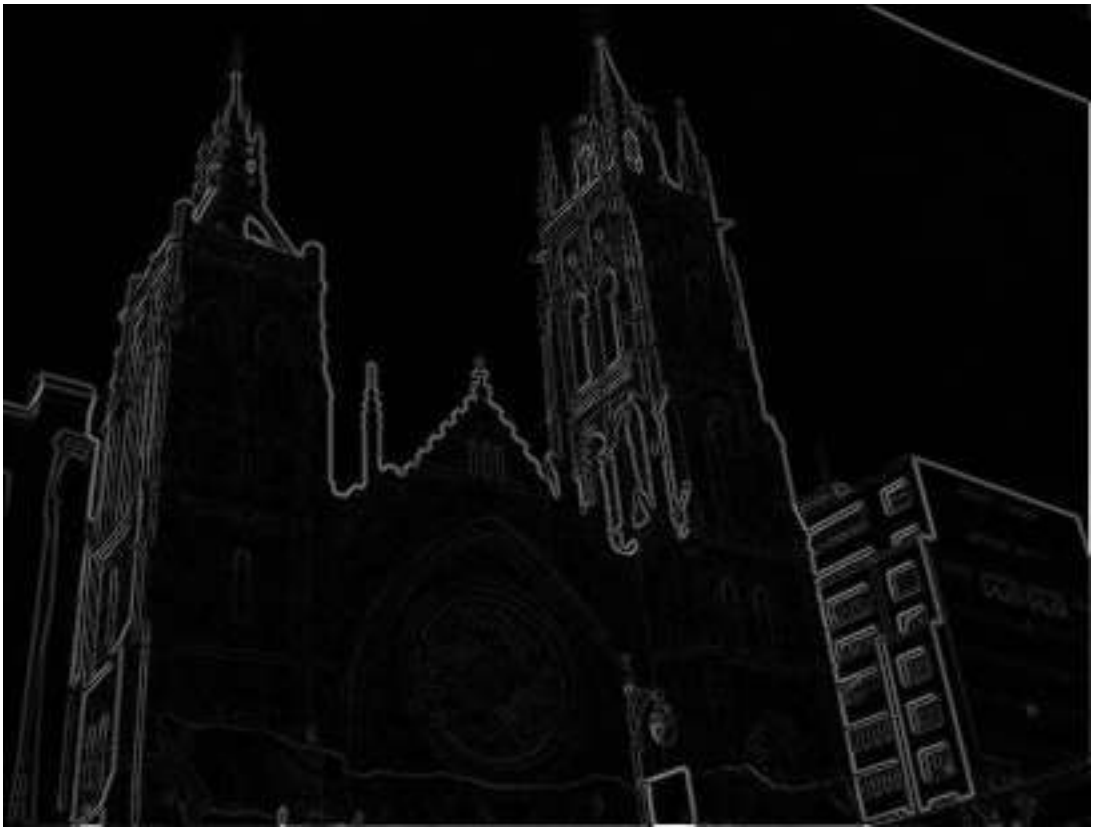
8.1.1 L'idée

Une image aléatoire ressemble à la mire en un temps où la télévision ne rediffusait pas les programmes diurne la nuit.



Dans ce brouillard aléatoire, la probabilité d'avoir des points alignés est très faible, si faible que le simple fait d'en voir est un événement extraordinaire. Trois points alignés ne sont pas rares, quatre un peu plus, cinq encore plus. A partir d'un certain seuil, on peut considérer que trop de points alignés forme une droite et un événement trop rare pour être ignoré. On cherche à détecter les arêtes dans une image comme la suivante.

On calcule le gradient d'une image en noir et blanc.



Puis on extrait les segments en les considérant comme des anomalies par rapport à un champ de pixels aléatoire.

8.1.2 Illustration

Détection de segments dans une image

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```


Une image aléatoire

On considère un bruit aléatoire uniforme dans une image et on ajoute des points aléatoires tirés le long d'une ligne selon une loi gaussienne : uniforme sur la ligne, gaussien autour du segment.

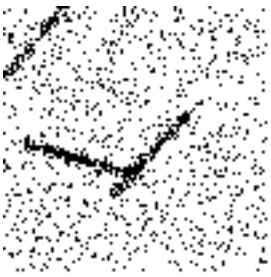
```
from mlstatpy.image.detection_segment import random_noise_image, convert_array2PIL
img = random_noise_image((100, 100))
convert_array2PIL(img, mode="binary")
```

```
from mlstatpy.image.detection_segment import random_segment_image
random_segment_image(img, density=3., lmin=0.3)
```

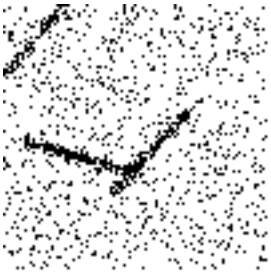
```
{'size': 36,
 'angle': 2.285619160431492,
 'x1': 23.597410654261072,
 'y1': 40,
 'x2': 0,
 'y2': 67.18753777770554,
 'nbpoints': 108}
```

```
convert_array2PIL(img, mode="binary")
```

```
random_segment_image(img, density=5., lmin=0.3)
random_segment_image(img, density=5., lmin=0.3)
convert_array2PIL(img, mode="binary")
```

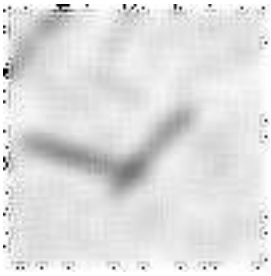


```
pilimg = convert_array2PIL(img, mode="binary").convert('RGB')
pilimg
```



```
from PIL import ImageFilter


pilimg = pilimg.filter(ImageFilter.BLUR).filter(ImageFilter.BLUR).filter(ImageFilter.
↪BLUR).filter(ImageFilter.BLUR)
pilimg
```

```
from PIL import ImageEnhance
enh = ImageEnhance.Sharpness(pilimg)
final_img = enh.enhance(4)
final_img
```

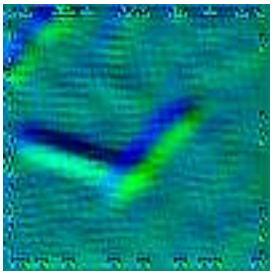


Gradient

La détection des segments est basée sur le gradient.

```
from mlstatpy.image.detection_segment import compute_gradient, plot_gradient
grad = compute_gradient(final_img, color=0)
```

```
plot_gradient(pilimg.copy(), grad, direction=-2)
```



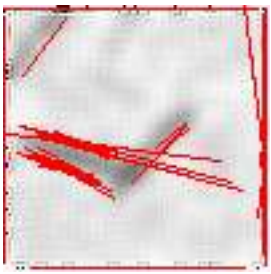
Détection de segments

```
from mlstatpy.image.detection_segment import detect_segments
seg = detect_segments(final_img, verbose=1, seuil_nfa=1e-1)
len(seg)
```

```
n =  1000  ...  82  temps  0.27  sec nalign 298
n =  2000  ...  82  temps  0.52  sec nalign 671
n =  3000  ...  164  temps  0.83  sec nalign 964
n =  4000  ...  164  temps  1.10  sec nalign 1357
n =  5000  ...  249  temps  1.39  sec nalign 1544
n =  6000  ...  252  temps  1.66  sec nalign 1924
n =  7000  ...  374  temps  1.95  sec nalign 2183
n =  8000  ...  375  temps  2.23  sec nalign 2460
n =  9000  ...  379  temps  2.56  sec nalign 2728
```

```
379
```

```
from mlstatpy.image.detection_segment import plot_segments
plot_segments(final_img.copy(), seg)
```



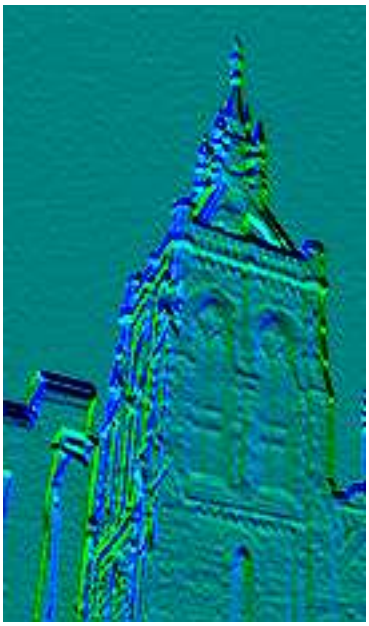
Détection de segments sur une image

```
from PIL import Image
egl = Image.open("eglise_zoom2.jpg")
egl
```

On réduit la taille de l'image.

```
egl2 = egl.crop((0, 0, egl.size[0]//3, 3*egl.size[1]//4))
egl2
```

```
grad2 = compute_gradient(egl2, color=0)
plot_gradient(egl2.copy(), grad2, direction=-2)
```



```
seg2 = detect_segments(egl2)
len(seg2)
```

```
490
```

```
from mlstatpy.image.detection_segment import plot_segments
res = plot_segments(egl2.copy(), seg2)
res
```

Il faudrait fusionner les segments mais cela a l'air de marcher.

La fonction `detect_segments` lance la détection des segments.

8.1.3 Explications

La présentation Détection des images dans les images digitales [142] détaille le principe de l'algorithme. L'idée de l'algorithme est assez proche de la transformée de Hough [143]. Celle-ci est implémentée dans le module scikit-image [144] ou opencv [145].

Bibliographie

— From Gestalt Theory to Image Analysis [146], Agnès Desolneux, Lionel Moisan, Jean-Michel Morel
— Learning Equivariant Functions with Matrix Valued Kernels [147]
— The Hough Transform Estimator [148]
— An Extension to Hough Transform Based on Gradient Orientation [149]

142. https://github.com/sdpython/mlstatpy/blob/master/_todo/segment_detection/presentation.pdf
143. https://fr.wikipedia.org/wiki/Transform%C3%A9e_de_Hough
144. http://scikit-image.org/docs/dev/api/skimage.transform.html
145. https://docs.opencv.org/2.4.13.6/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html?highlight=hough
146. http://www.math-info.univ-paris5.fr/~moisan/papers/2006-9.pdf
147. http://www.jmlr.org/papers/volume8/reisert07a/reisert07a.pdf
148. https://arxiv.org/pdf/math/0503668.pdf
149. https://arxiv.org/ftp/arxiv/papers/1510/1510.04863.pdf

CHAPITRE 9

Pérégrinations d'un data scientist

Ce sont quelques notebooks sur des points particuliers qui surgissent au quotidien quand on traite des données. Souvent plus pratiques que théoriques, c'est l'occasion de découvrir quelques poussières sous le tapis.

9.1 Répartir en base d'apprentissage et de test

C'est un problème plutôt facile puisqu'il s'agit de répartir aléatoirement les lignes d'une base de données d'un côté ou de l'autre. Lorsque le problème de machine learning à résoudre est un problème de classification, il faut s'assurer que chaque côté contient une proportion raisonnable de chaque classe.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```



```
import matplotlib.pyplot as plt
%matplotlib inline
```

9.1.1 Répartition naïve

On considère une base de données qu'on divise en 2/3 apprentissage, 1/3 test. On note cette proportion t. Deux classes 0 et 1, la proportion de la classe 1 est de p qu'on choisit petit.

```
import random
def generate_dataset(n, t):
    return [1 if random.random() < t else 0 for i in range(0,n)]
```

(suite sur la page suivante)

(suite de la page précédente)

```
ens = generate_dataset(4000, 0.01)
sum(ens)
```

```
40
```

Et on divise en base d'apprentissage et de test.

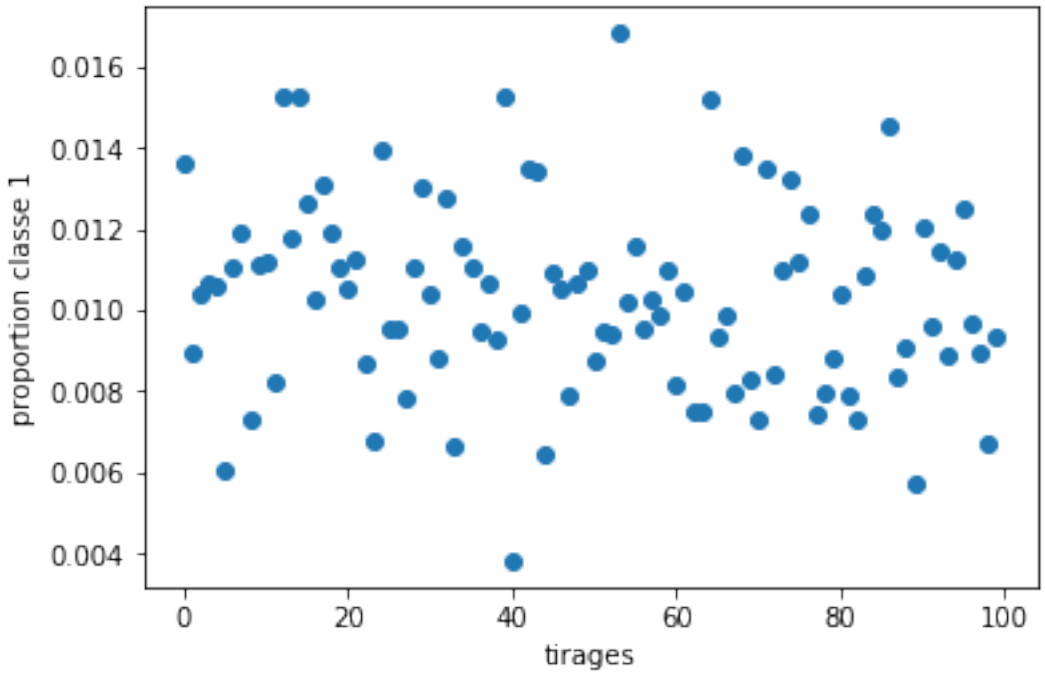
```
def custom_split_train_test(ens, p):
    choice = generate_dataset(len(ens), p)
    train = [x for x, c in zip(ens, choice) if c == 1]
    test = [x for x, c in zip(ens, choice) if c == 0]
    return train, test

train, test = custom_split_train_test(ens, 0.66)
print(len(train), sum(train), sum(train)/len(train))
print(len(test), sum(test), sum(test)/len(test))
```

```
2683 27 0.010063361908311 59
1317 13 0.009870918754745633
```

On recommence un grand nombre de fois et on représente la proportion obtenue dans la base de test.

```
tirages = [sum(test)/len(test) for train, test in [custom_split_train_test(ens, 0.66)␣
↪for i in range(0,100)]]
```

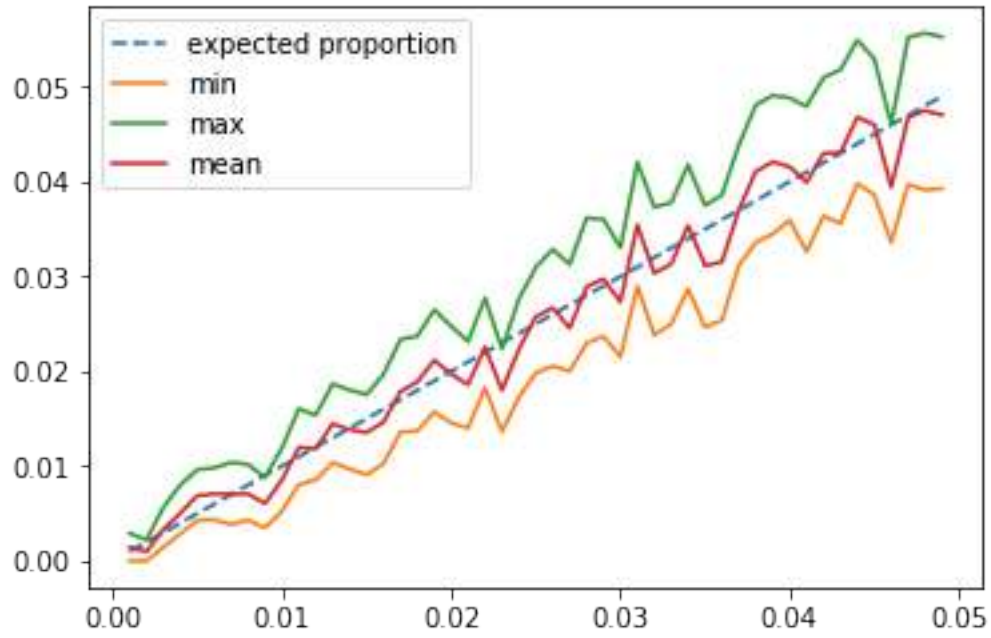
```
plt.plot(tirages, "o")
plt.ylabel("proportion classe 1")
plt.xlabel("tirages");
```



On considère maintenant la moyenne, les valeurs extrêmes de la proportion en faisant varier p.

```
ps = [0.001 * i for i in range(1, 50)]
tmin, tmax, tmean = [], [], []
for p in ps:
    ens = generate_dataset(4000, p)
    tirages = [sum(test)/len(test) for train, test in [custom_split_train_test(ens, 0.
↪66) for i in range(0,200)]]
    tirages.sort()
    tmin.append(tirages[int(len(tirages)*0.05)])
    tmax.append(tirages[-int(len(tirages)*0.05)])
    tmean.append(sum(tirages) / len(tirages))
```

```
fig, ax = plt.subplots(1, 1)
ax.plot(ps, ps, "--", label="expected proportion")
ax.plot(ps, tmin, label="min")
ax.plot(ps, tmax, label="max")
ax.plot(ps, tmean, label="mean")
ax.legend();
```

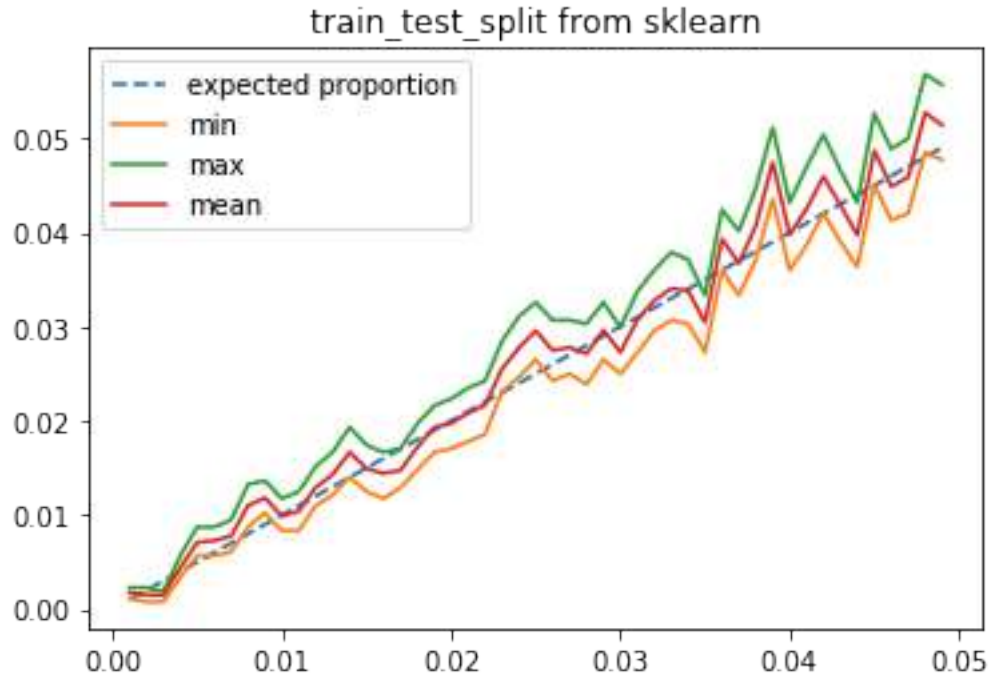


Et train_test_split [150]...

```
from sklearn.model_selection import train_test_split
import pandas

ps = [0.001 * i for i in range(1, 50)]
tmin, tmax, tmean = [], [], []
for p in ps:
    ens = pandas.Series(generate_dataset(4000, p))
    tirages = [sum(test)/len(test) for train, test in [train_test_split(ens, test_
↪size=0.66) for i in range(0,200)]]
    tirages.sort()
    tmin.append(tirages[int(len(tirages)*0.05)])
    tmax.append(tirages[-int(len(tirages)*0.05)])
    tmean.append(sum(tirages) / len(tirages))
```

150. http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
fig, ax = plt.subplots(1, 1)
ax.plot(ps, ps, "--", label="expected proportion")
ax.plot(ps, tmin, label="min")
ax.plot(ps, tmax, label="max")
ax.plot(ps, tmean, label="mean")
ax.set_title("train_test_split from sklearn")
ax.legend();
```



L'écart entre les extrêmes paraît plus petit. Le générateur pseudo aléatoire utilisé par scikit-learn [151] paraît de meilleur qualité. Nous y reviendront peut-être un jour.

9.1.2 Répartition stratifiée

Nous utilisons maintenant le paramètre *stratify* qui permet de s'assurer que les deux classes sont équitablement réparties entre les deux ensembles *train* et *test*.

```
from sklearn.model_selection import train_test_split
import pandas

ps = [0.001 * i for i in range(1, 50)]
tmin, tmax, tmean = [], [], []
for p in ps:
    ens = pandas.Series(generate_dataset(4000, p))

    traintest = []
    excs = []
    for i in range(0, 200):
        try:
            train, test = train_test_split(ens, test_size=0.66, stratify=ens)
            traintest.append((train,test))
```

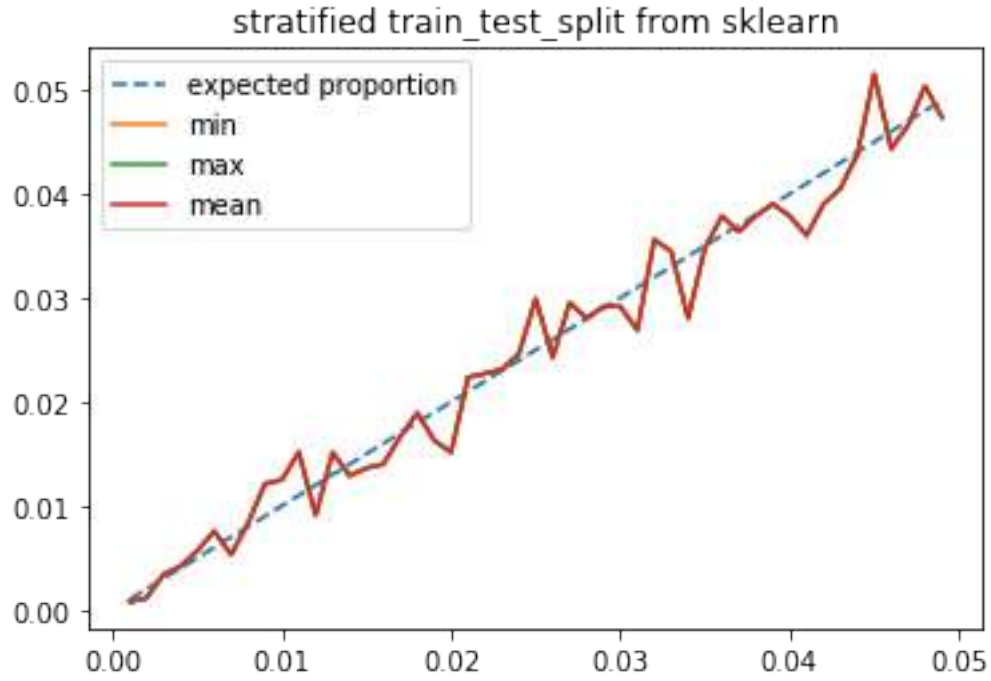
(suite sur la page suivante)

151. http://scikit-learn.org/

(suite de la page précédente)

```
        except ValueError as e:
            print("Skipping: small class too small in one side", e)
            excs.append(e)
    if len(traintest) < 100:
        ex = excs[0] if len(excs) > 0 else "no exception"
        raise Exception("Too few, you should check the implementation is ok.\n{0}".
↪format(ex))
    tirages = [sum(test)/len(test) for train, test in traintest]
    tirages.sort()
    tmin.append(tirages[int(len(tirages)*0.05)])
    tmax.append(tirages[-int(len(tirages)*0.05)])
    tmean.append(sum(tirages) / len(tirages))
```

```
fig, ax = plt.subplots(1, 1)
ax.plot(ps, ps, "--", label="expected proportion")
ax.plot(ps, tmin, label="min")
ax.plot(ps, tmax, label="max")
ax.plot(ps, tmean, label="mean")
ax.set_title("stratified train_test_split from sklearn")
ax.legend();
```



La proportion initiale est bien respectée. Comment faire cela en pratique ? Le plus simple est sans doute de :

— De trier les observations qui appartiennent à chaque classe.
— De les permuter de façon aléatoire.
— Puis de prendre les premiers éléments pour la base d'apprentissage dans chaque classe et les derniers pour la base de test.

```
def custom_statitied_split_train_test(ens, p, stratify):
    strat = set(stratify)
    train = []
    test = []
    for st in strat:
```

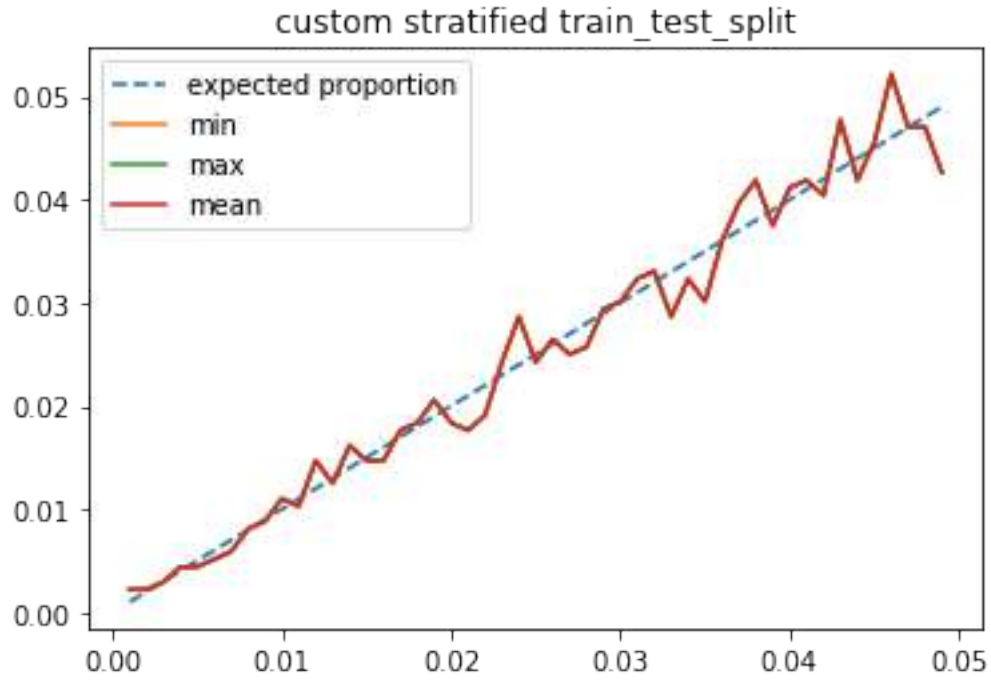
(suite sur la page suivante)

(suite de la page précédente)

```
        cl = [e for e, s in zip(ens, stratify) if s == st]
        random.shuffle(cl)
        i = int(len(cl) * p)
        train.extend(cl[:i])
        test.extend(cl[i:])
    return train, test
```

```
ps = [0.001 * i for i in range(1, 50)]
tmin, tmax, tmean = [], [], []
for p in ps:
    ens = generate_dataset(4000, p)
    tirages = [sum(test)/len(test) for train, test in [custom_statitied_split_train_
↪test(ens,
                          p=0.66, stratify=ens) for i in range(0,200)]]
    tirages.sort()
    tmin.append(tirages[int(len(tirages)*0.05)])
    tmax.append(tirages[-int(len(tirages)*0.05)])
    tmean.append(sum(tirages) / len(tirages))
```

```
fig, ax = plt.subplots(1, 1)
ax.plot(ps, ps, "--", label="expected proportion")
ax.plot(ps, tmin, label="min")
ax.plot(ps, tmax, label="max")
ax.plot(ps, tmean, label="mean")
ax.set_title("custom stratified train_test_split")
ax.legend();
```



La méthode est simple mais plus coûteuse puisqu'elle nécessite un tri.

```
import time

ns = []
```

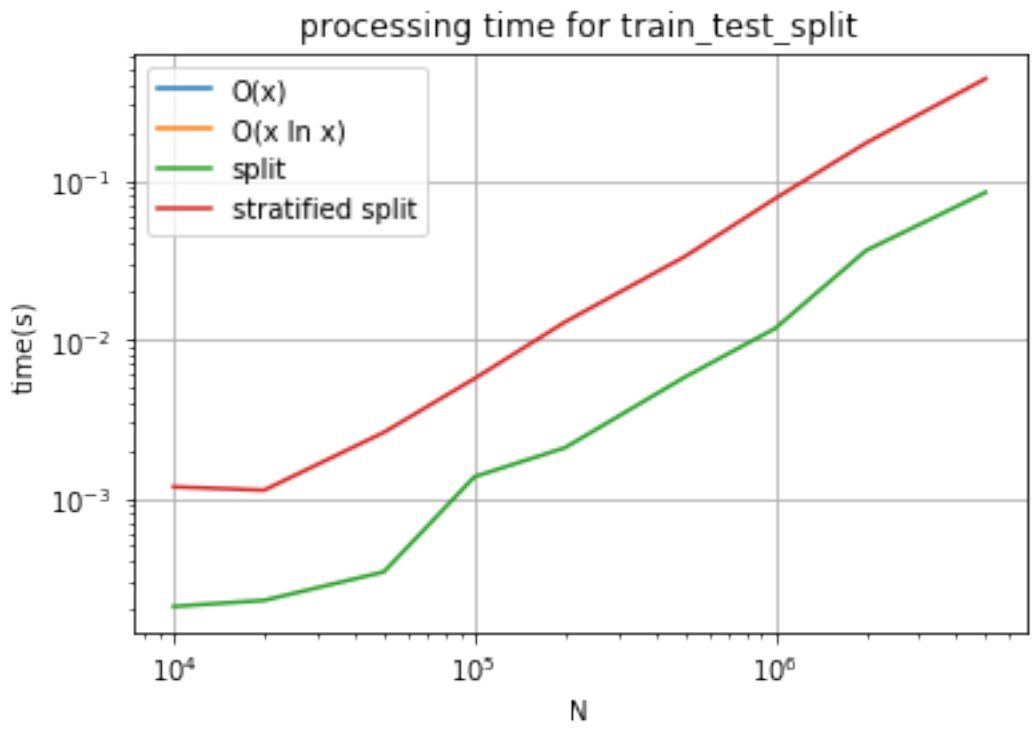
(suite sur la page suivante)

(suite de la page précédente)

```
tn = []
ts = []

ii = 5
for N in [10000, 20000, 50000, 100000, 200000, 500000, int(1e6),
          int(2e6), int(5e6)]:
    print(N)
    ens = pandas.Series(generate_dataset(N, 0.05))
    ns.append(N)

    tt = []
    for i in range(0,ii):
        t = time.perf_counter()
        train_test_split(ens, test_size=0.66)
        d = 1.0 * (time.perf_counter()-t) / ii
        tt.append(d)
    tt.sort()
    tn.append(tt[len(tt)//2])

    tt = []
    for i in range(0,ii):
        t = time.perf_counter()
        train_test_split(ens, test_size=0.66, stratify=ens)
        d = 1.0 * (time.perf_counter()-t) / ii
        tt.append(d)
    tt.sort()
    ts.append(tt[len(tt)//2])
```

```
10000
20000
50000
100000
200000
500000
1000000
2000000
5000000
```

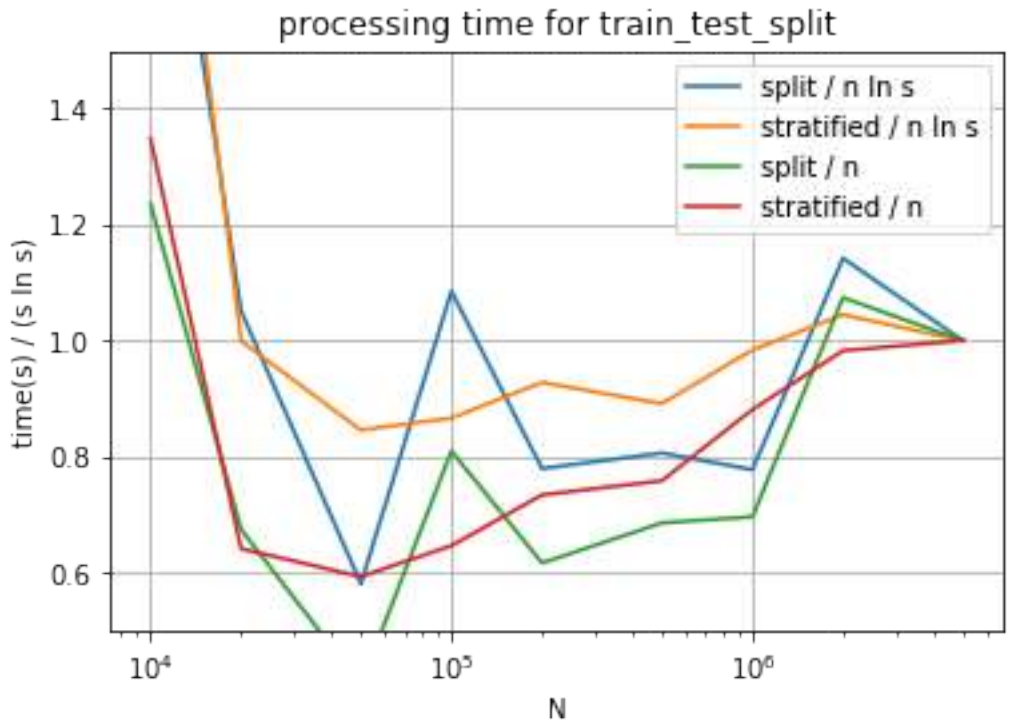
```
import math
fig, ax = plt.subplots(1, 1)
dd = tn[-1] - (ts[-1] - tn[-1])*1.3
ax.plot(ns, [x * dd / ns[-1] for x in ns], label="O(x)")
ax.plot(ns, [x * math.log(x) * ns[0] * dd / ns[-1] / (ns[0] * math.log(ns[0])) for x␣
↪in ns], label="O(x ln x)")
ax.plot(ns, tn, label="split")
ax.plot(ns, ts, label="stratified split")
ax.set_title("processing time for train_test_split")
ax.grid(True)
ax.set_xscale("log", nonposx='clip')
ax.set_yscale("log", nonposy='clip')
ax.set_xlabel("N")
ax.set_ylabel("time(s)")
ax.legend();
```



Le coût de la fonction train_test_split [152] semble être entre $O(n)$ et $O(n \ln n)$. Regardons.

```
import math
fig, ax = plt.subplots(1, 1)
ax.plot(ns, [(x / tn[-1]) / (n * math.log(n) / (ns[-1] * math.log(ns[-1]))) for x, n␣
↪in zip(tn, ns)],
        label="split / n ln s")
ax.plot(ns, [(x / ts[-1]) / (n * math.log(n) / (ns[-1] * math.log(ns[-1]))) for x, n␣
↪in zip(ts, ns)],
        label="stratified / n ln s")
ax.plot(ns, [(x / tn[-1]) / (n / ns[-1]) for x, n in zip(tn, ns)], label="split / n")
ax.plot(ns, [(x / ts[-1]) / (n / ns[-1]) for x, n in zip(ts, ns)], label="stratified /
↪ n")

ax.set_title("processing time for train_test_split")
ax.grid(True)
ax.set_xscale("log", nonposx='clip')
ax.set_xlabel("N")
ax.set_ylabel("time(s) / (s ln s)")
ax.set_ylim([0.5,1.5])
ax.legend();
```

152. http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html



C'est difficile à voir sur ce schéma. Il faudrait tirer plus d'exemple, regader les quantiles plutôt que la seule médiane. Le code de scikit-learn [153] est plus explicite, une permutation précède la répartition en train / test.

9.1.3 Streaming splitting

Streaming veut dire qu'on traite les données sous la forme d'un flux et qu'on ne sait pas combien il y en. Concrètement, il faut commencer la répartition train / test au moment sans savoir quand elle s'arrêtera. Par conséquent, il faut qu'à tout instant, on soit capable d'interrompre la répartition et celle-ci doit être valide.

Le premier algorithme qui consiste à tirer un nombre aléatoire et à le répartir en train ou test selon le nombre tiré. Chaque observation est traitée indépendamment. A tout moment, la répartition peut être interrompue. En pratique, on implémente ce type de processus sous la forme d'itérateur ou de mapper. C'est une fonction qui accepte un itérateur sur un ensemble et qui retourne un itérateur sur les valeurs transformées. Dans notre cas, on retourne l'observation, suivi de 0 si elle est classée en *train* et 1 en *test*.

```
def streaming_split_train_test(stream, p):
    for obs in stream:
        x = random.random()
        yield obs, 0 if x <= p else 1
```

```
def iterate_data(n, t):
    while n > 0:
        yield 1 if random.random() < t else 0
        n -= 1

for obs, s in streaming_split_train_test(iterate_data(10, 0.05), 0.66):
    print("obs={0} train/test={1}".format(obs, s))
```

153. https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/model_selection/_split.py#L1048

```
obs=0 train/test=1
obs=0 train/test=0
obs=0 train/test=0
obs=0 train/test=1
obs=0 train/test=0
obs=1 train/test=0
obs=0 train/test=0
obs=0 train/test=0
obs=0 train/test=0
obs=0 train/test=0
```

La répartition stratifiée repose sur une permutation aléatoire et cela implique d'avoir accès à l'intégralité des données au préalable. En *streaming*, ce n'est pas possible. Il faut donc penser à autre chose pour obtenir une version stratifiée de la version *streaming*. Rien n'empêche en version *streaming* de garder les dernières observations en mémoire pour faire une mini-permutation. Nous allons introduire quelques changements :

1. Le *stream* est maintenant un flux sur deux valeurs, l'observation et la classe à laquelle il appartient.
2. La fonction va conserver les dernières valeurs pour chaque classe.
3. La fonction va produire des observations de temps en temps quand elle sera sûre que les observations seront stratifiées.
4. Nuos allons compter les observations distribuées dans chaque base.

```
def streaming_stratified_split_train_test(stream, p):
    n = max(1/p, 1/(1-p))
    if n > 10000:
        raise Exception("Cette répartition train / test est vraiment déséquilibrée.")
    memory = {}
    for obs, strat in stream:
        if strat not in memory:
            memory[strat] = []
        memory[strat].append(obs)

        for k in memory:
            v = memory[k]
            if len(v) >= n:
                # on permute aléatoirement
                random.shuffle(v)
                i = int(len(v) * p + 0.5)
                for j in range(0,i):
                    yield v[j], 0  # apprentissage
                for j in range(i,len(v)):
                    yield v[j], 1  # test
                # on efface de la mémoire les informations produites
                memory[k] = []

    # lorsqu'on a fini, il faut tout de même répartir les
    # observations stockées
    for k in memory:
        v = memory[k]
        # on permute aléatoirement
        random.shuffle(v)
        i = int(len(v) * p)
        for j in range(0,i):
            yield v[j], 0  # apprentissage
        for j in range(i,len(v)):
            yield v[j], 1  # test
```

```
iter = streaming_stratified_split_train_test( ((i,i) for i in iterate_data(10000, 0.
↪05)), 0.66)
df = pandas.DataFrame(iter)
df.columns = ["obs", "train_test"]
print(df.shape)
df.head()
```

```
(10000, 2)
```

```
df2 = df.copy()
df2["un"] = 1
piv = df2.groupby(["obs", "train_test"], as_index=False).count().pivot("train_test",
↪"obs", "un")
piv
```

Il y a juste un petit problème avec cette implémentation. On multiplie la taille du buffer par un réel. Je suggère d'enlever le nombre 0.5 dans le code pour voir ce qu'il se passe. La somme des arrondis est loin d'être un arrondi même si N choisi tel que $N = \max(\frac{1}{p}, \frac{1}{1-p})$.

```
piv["sum"] = piv[0] + piv[1]
piv["ratio"] = piv[1] / piv["sum"]
piv
```

Il faut corriger ces erreurs d'arrondi. On s'inspire de l'algorithme de Bresenham [154] et mémoriser les éléments répartis.

```
def streaming_stratified_split_train_test2(stream, p):
    n = max(1/p, 1/(1-p))
    if n > 10000:
        raise Exception("Cette répartition train / test est vraiment déséquilibrée.")
    counts = {}
    memory = {}
    for obs, strat in stream:
        if strat not in memory:
            memory[strat] = []
        memory[strat].append(obs)

        for k in memory:
            v = memory[k]
            if len(v) >= n:
                # on permute aléatoirement
                random.shuffle(v)
                if (0,k) in counts:
                    tt = counts[1,k] + counts[0,k]
                    delta = - int(counts[0,k] - tt*p + 0.5)
                else:
                    delta = 0
                i = int(len(v) * p + 0.5)
                i += delta
                i = max(0, min(len(v), i))
                for j in range(0,i):
                    yield v[j], 0  # apprentissage
                for j in range(i,len(v)):
                    yield v[j], 1  # test
                if (0,k) not in counts:
                    counts[0,k] = i
```

(suite sur la page suivante)

154. https://fr.wikipedia.org/wiki/Algorithme_de_trac%C3%A9_de_segment_de_Bresenham

(suite de la page précédente)

```
                    counts[1,k] = len(v)-i
                else:
                    counts[0,k] += i
                    counts[1,k] += len(v)-i
                # on efface de la mémoire les informations produites
                memory[k] = []

    # lorsqu'on a fini, il faut tout de même répartir les
    # observations stockées
    for k in memory:
        v = memory[k]
        # on permute aléatoirement
        random.shuffle(v)
        if (0,k) in counts:
            tt = counts[1,k] + counts[0,k]
            delta = - int(counts[0,k] - tt*p + 0.5)
        else:
            delta = 0
        i = int(len(v) * p + 0.5)
        i += delta
        i = max(0, min(len(v), i))
        for j in range(0,i):
            yield v[j], 0  # apprentissage
        for j in range(i,len(v)):
            yield v[j], 1  # test
        if (0,k) not in counts:
            counts[0,k] = i
            counts[1,k] = len(v)-i
        else:
            counts[0,k] += i
            counts[1,k] += len(v)-i
```

```
iter = streaming_stratified_split_train_test2( ((i,i) for i in iterate_data(10000, 0.
↪05)), 0.66)
df = pandas.DataFrame(iter)
df.columns = ["obs", "train_test"]
df2 = df.copy()
df2["un"] = 1
piv = df2.groupby(["obs", "train_test"], as_index=False).count().pivot("train_test",
↪"obs", "un")
piv["sum"] = piv[0] + piv[1]
piv["ratio"] = piv[1] / piv["sum"]
print("ratio de classe 1 dans l'échantillon entier %1.5f" %
      ((piv.iloc[1,1] + piv.iloc[0,1]) / (piv.iloc[0,1] + piv.iloc[0,0] + piv.iloc[1,
↪1] + piv.iloc[1,0]) ))
piv
```

```
ratio de classe 1 dans l'échantillon entier 0.04980
```

Pas trop mal. Le dernier inconvénient est la taille de la fenêtre. Dans l'exemple qui suit, elle sera de 3. L'algorithme va donc grouper les éléments par trois, les permuter aléatoirement et les laisser filer. Nous ne pourrons jamais avoir trois éléments de suite du même côté *train* ou *test*. On peut bidouiller comme suit (ligne marquées `# changement`). La fonction qui suit ne permet toujours pas d'avoir de grandes séquences répartie du même côté mais ce sera l'inconvénient de ce type d'algorithme. La taille du buffer limite cette possibilité.

```
def streaming_stratified_split_train_test3(stream, p):
    n = 2 * max(1/p, 1/(1-p)) # changement
    if n > 10000:
        raise Exception("Cette répartition train / test est vraiment déséquilibrée.")
    counts = {}
    memory = {}
    for obs, strat in stream:
        if strat not in memory:
            memory[strat] = []
        memory[strat].append(obs)

        for k in memory:
            v = memory[k]
            if len(v) >= n + random.randint(0, 10):  # changement
                # on permute aléatoirement
                random.shuffle(v)
                if (0,k) in counts:
                    tt = counts[1,k] + counts[0,k]
                    delta = - int(counts[0,k] - tt*p + 0.5)
                else:
                    delta = 0
                i = int(len(v) * p + 0.5)
                i += delta
                i = max(0, min(len(v), i))
                for j in range(0,i):
                    yield v[j], 0  # apprentissage
                for j in range(i,len(v)):
                    yield v[j], 1  # test
                if (0,k) not in counts:
                    counts[0,k] = i
                    counts[1,k] = len(v)-i
                else:
                    counts[0,k] += i
                    counts[1,k] += len(v)-i
                # on efface de la mémoire les informations produites
                memory[k] = []

    # lorsqu'on a fini, il faut tout de même répartir les
    # observations stockées
    for k in memory:
        v = memory[k]
        # on permute aléatoirement
        random.shuffle(v)
        if (0,k) in counts:
            tt = counts[1,k] + counts[0,k]
            delta = - int(counts[0,k] - tt*p + 0.5)
        else:
            delta = 0
        i = int(len(v) * p + 0.5)
        i += delta
        i = max(0, min(len(v), i))
        for j in range(0,i):
            yield v[j], 0  # apprentissage
        for j in range(i,len(v)):
            yield v[j], 1  # test
        if (0,k) not in counts:
            counts[0,k] = i
```

(suite sur la page suivante)

(suite de la page précédente)

```
            counts[1,k] = len(v)-i
        else:
            counts[0,k] += i
            counts[1,k] += len(v)-i
```

```
iter = streaming_stratified_split_train_test3( ((i,i) for i in iterate_data(10000, 0.
↪05)), 0.66)
df = pandas.DataFrame(iter)
df.columns = ["obs", "train_test"]
df2 = df.copy()
df2["un"] = 1
piv = df2.groupby(["obs", "train_test"], as_index=False).count().pivot("train_test",
↪"obs", "un")
piv["sum"] = piv[0] + piv[1]
piv["ratio"] = piv[1] / piv["sum"]
print("ratio de classe 1 dans l'échantillon entier %1.5f" %
      ((piv.iloc[1,1] + piv.iloc[0,1]) / (piv.iloc[0,1] + piv.iloc[0,0] + piv.iloc[1,
↪1] + piv.iloc[1,0]) ))
piv
```

```
ratio de classe 1 dans l'échantillon entier 0.05000
```

9.1.4 Streaming distribué

C'est possible mais c'est un peu plus compliqué parce que le hasard en distribué, c'est compliqué. On n'est jamais sûr que des séries pseudo-aléatoires soient tout-à-fait indépendantes lorsqu'elles sont générées en parallèles.

9.2 Corrélations non linéaires

Les corrélations indiquent si deux variables sont linéairement équivalentes. Comment étendre cette notion à des variables liées mais pas de façon linéaire.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

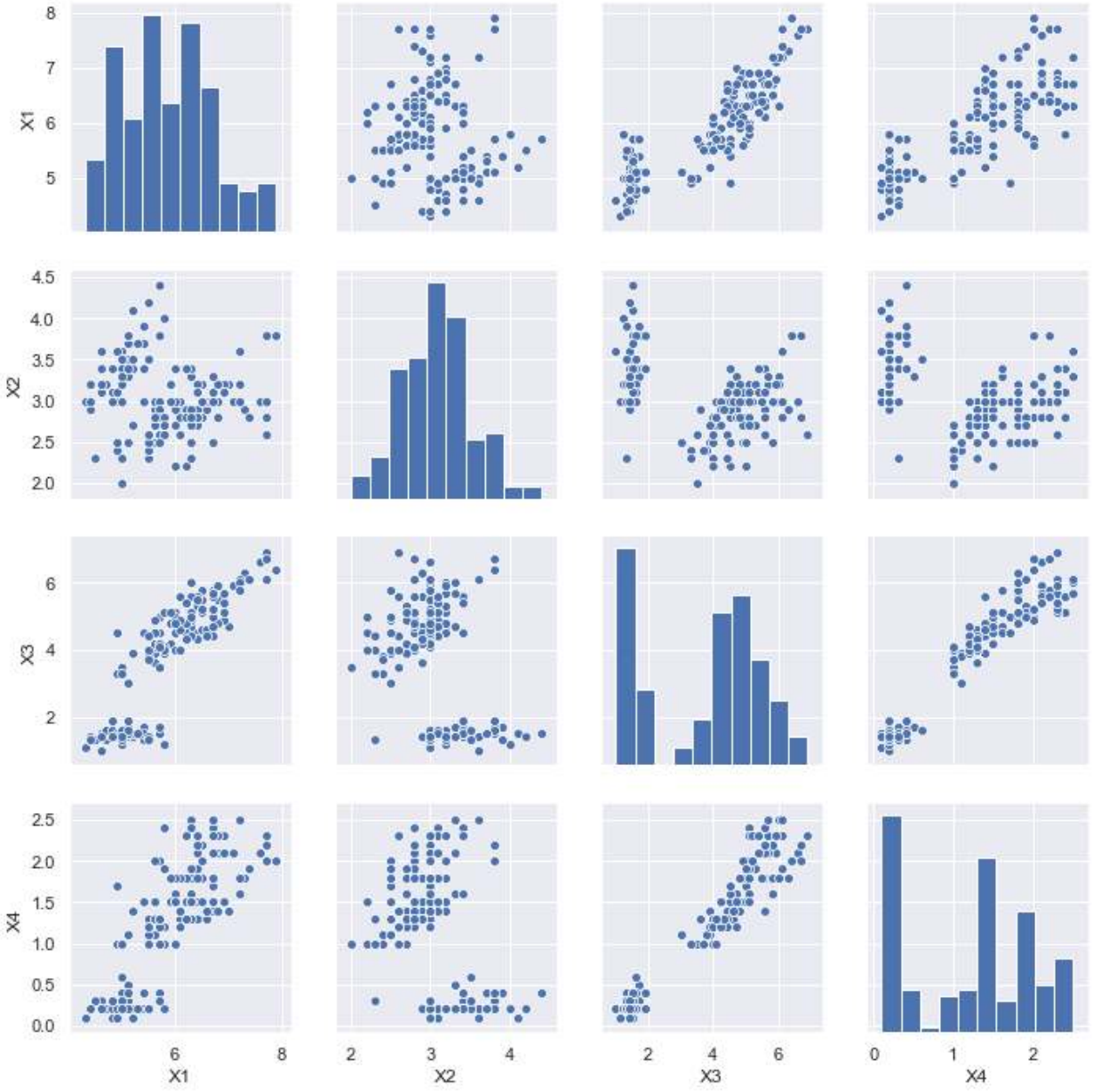


```
%matplotlib inline
```

9.2.1 Un exemple

```
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data
Y = iris.target
import pandas
df = pandas.DataFrame(X)
df.columns = ["X1", "X2", "X3", "X4"]
df.head()
```

```
import seaborn as sns
sns.set()
sns.pairplot(df);
```



Et les corrélations :

```
df.corr()
```

9.2.2 Un peu de théorie

Le coefficient de corrélation [155] de Pearson est calculé comme suit :

$$cor(X_i, X_j) = \frac{cov(X_i, Y_i)}{\sigma(X_i)\sigma(X_j)}$$

Lorsque les variables sont centrées $\mathbb{E}X_i = \mathbb{E}X_j = 0$, cette formule devient :

$$cor(X_i, X_j) = \frac{\mathbb{E}(X_i X_j)}{\sqrt{\mathbb{E}X_i^2 \mathbb{E}X_j^2}}$$

Lorsque les variables sont réduites $\mathbb{E}X_i^2 = \mathbb{E}X_j^2 = 1$, cette formule devient $cor(X_i, X_j) = \mathbb{E}(X_i X_j)$. Admettons maintenant que l'on cherche à trouver le coefficient α_{ij} qui minimise la variance du bruit ϵ_{ij} :

$$X_j = \alpha_{ij} X_i + \epsilon_{ij}$$

Le coefficient α_{ij} est le résultat d'une régression linéaire qui minimise $\mathbb{E}(X_j - \alpha_{ij} X_i)^2$. Si les variables X_i, X_j sont centrées et réduites : $\alpha_{ij}^* = \mathbb{E}(X_i X_j) = cor(X_i, X_j)$. On étend cette définition dans le cas d'une fonction paramétrable $f : f(\omega, X) \rightarrow \mathbb{R}$ et d'une régression non linéaire. On suppose que les paramètres ω^* minimisent la quantité $\min_\omega (X_j - f(\omega, X_i))^2$. On écrit alors $X_j = \alpha_{ij} \frac{f(\omega^*, X_i)}{\alpha_{ij}} + \epsilon_{ij}$ et on choisit α_{ij} de telle sorte que $\mathbb{E}\left(\frac{f(\omega^*, X_i)^2}{\alpha_{ij}^2}\right) = 1$. On définit la corrélation non linéaire au sens de f :

$$cor^f(X_i, X_j) = \sqrt{\mathbb{E}(f(\omega, X_i)^2)}$$

On vérifie que ce coefficient est compris entre [0, 1]. Il est positif de manière évidente. Il est également inférieur à 1, si cela n'était pas le cas, nous pourrions construire une fonction $f(\omega^*, X) + c$ qui est une meilleur solution pour le programme de minimisation. Ce nombre ressemble à une corrélation à ceci près qu'elle ne peut être négative.

9.2.3 Vérifications

Tout d'abord le cas linéaire :

```
from sklearn.preprocessing import scale
import numpy

def correlation_etendue(df, model, **params):
    cor = df.corr()
    df = scale(df)
    for i in range(cor.shape[0]):
        xi = df[:, i:i+1]
        for j in range(cor.shape[1]):
            mod = model(**params)
            xj = df[:, j]
            mod.fit(xi, xj)
            v = mod.predict(xi)
            c = numpy.std(v)
            cor.iloc[i,j] = c
    return cor
```

(suite sur la page suivante)

155. https://fr.wikipedia.org/wiki/Corr%C3%A9lation_(statistiques)

(suite de la page précédente)

```
from sklearn.linear_model import LinearRegression
cor = correlation_etendue(df, LinearRegression, fit_intercept=False)
cor
```

On affiche à nouveau les corrélations qui sont identiques au signe près.

```
df.corr()
```

Et le cas non linéaire :

```
from sklearn.tree import DecisionTreeRegressor
cor = correlation_etendue(df, DecisionTreeRegressor)
cor
```

```
from sklearn.ensemble import RandomForestRegressor
cor = correlation_etendue(df, RandomForestRegressor, n_estimators=10)
cor
```

9.2.4 Overfitting

Ces chiffres sont beaucoup trop optimistes. Les modèles de machine learning peuvent tout à fait faire de l'overfitting. Il faut améliorer la fonction en divisant en apprentissage et test plusieurs fois. Il faut également tenir compte de l'erreur de prédiction. On rappelle que :

$$X_j = \alpha_{ij} \frac{f(\omega^*, X_i)}{\alpha_{ij}} + \epsilon_{ij} = cor^f(X_i, X_j) \frac{f(\omega^*, X_i)}{\sqrt{\mathbb{E}(f(\omega, X_i)^2)}} + \epsilon_{ij}$$

Or $\mathbb{E}(X_j^2) = 1$ et on suppose que les bruits ne sont pas corrélées linéairement aux $f(\omega^*, X_i)$. On en déduit que $cor^f(X_i, X_j) = \sqrt{1 - \mathbb{E}\epsilon_{ij}^2}$.

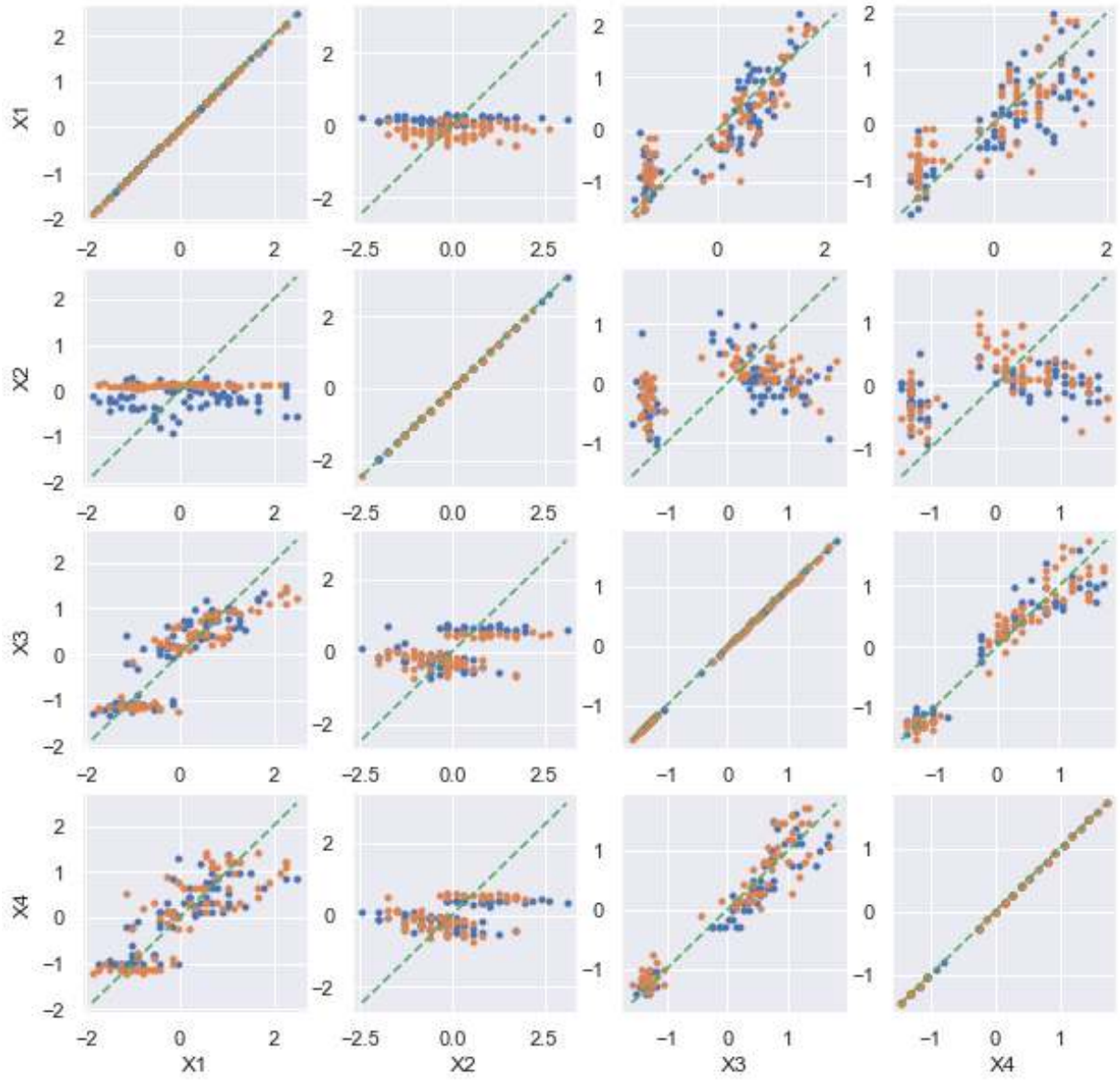
```
from sklearn.model_selection import train_test_split
import numpy

def correlation_cross_val(df, model, draws=5, **params):
    cor = df.corr()
    df = scale(df)
    for i in range(cor.shape[0]):
        xi = df[:, i:i+1]
        for j in range(cor.shape[1]):
            xj = df[:, j]
            mem = []
            for k in range(0, draws):
                xi_train, xi_test, xj_train, xj_test = train_test_split(xi, xj, test_
↪size=0.5)
                mod = model(**params)
                mod.fit(xi_train, xj_train)
                v = mod.predict(xi_test)
                c = (1 - numpy.var(v - xj_test))
                mem.append(max(c, 0) **0.5)
            cor.iloc[i,j] = sum(mem) / len(mem)
    return cor

cor = correlation_cross_val(df, LinearRegression, fit_intercept=False, draws=20)
cor
```

```
cor = correlation_cross_val(df, DecisionTreeRegressor)
cor
```

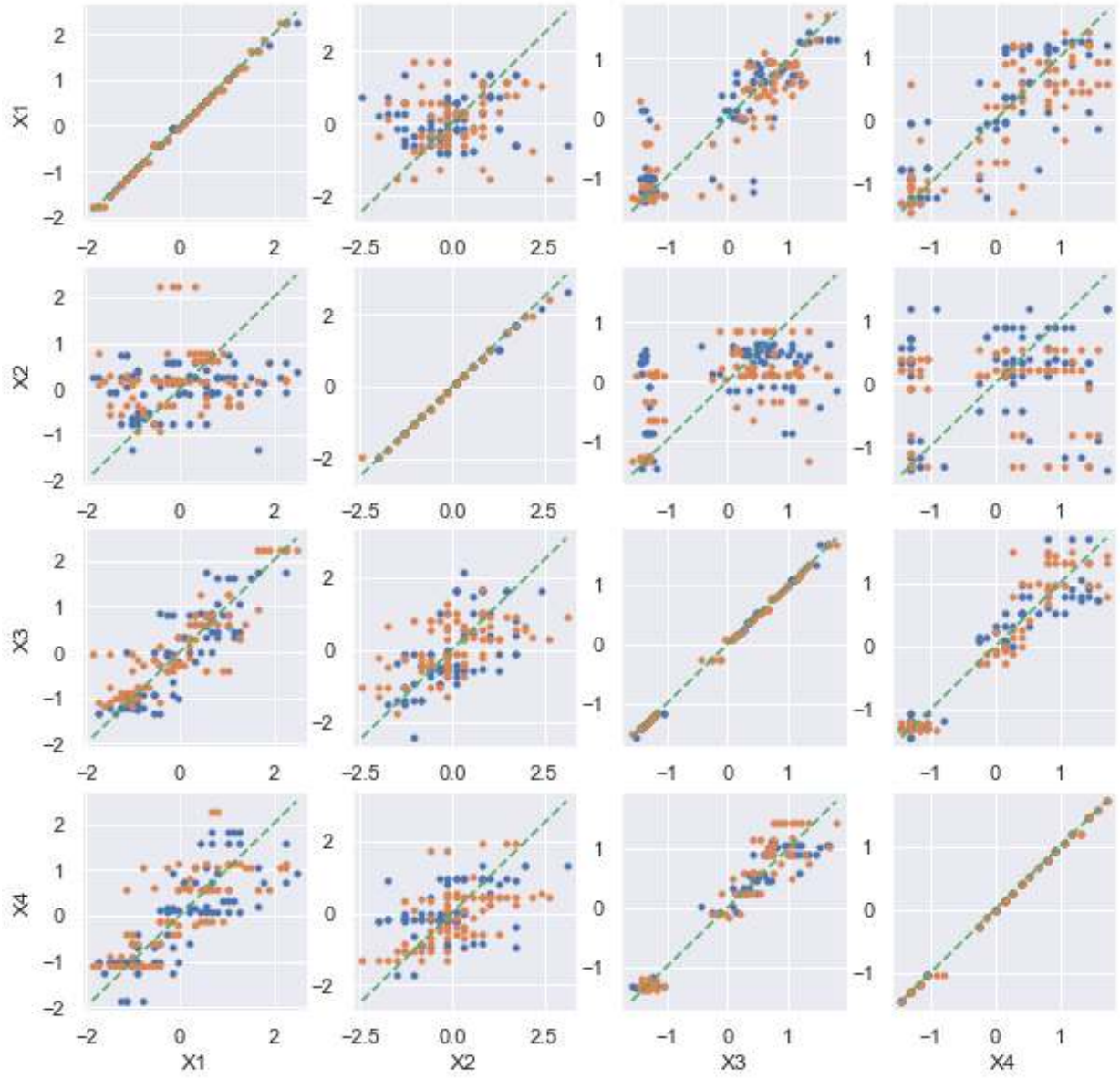
```
cor = correlation_cross_val(df, RandomForestRegressor, n_estimators=10)
cor
```

Les résultats sont assez fluctuants lorsque les données sont mal corrélées. On remarque également que la matrice n'est plus nécessairement symmétrique.

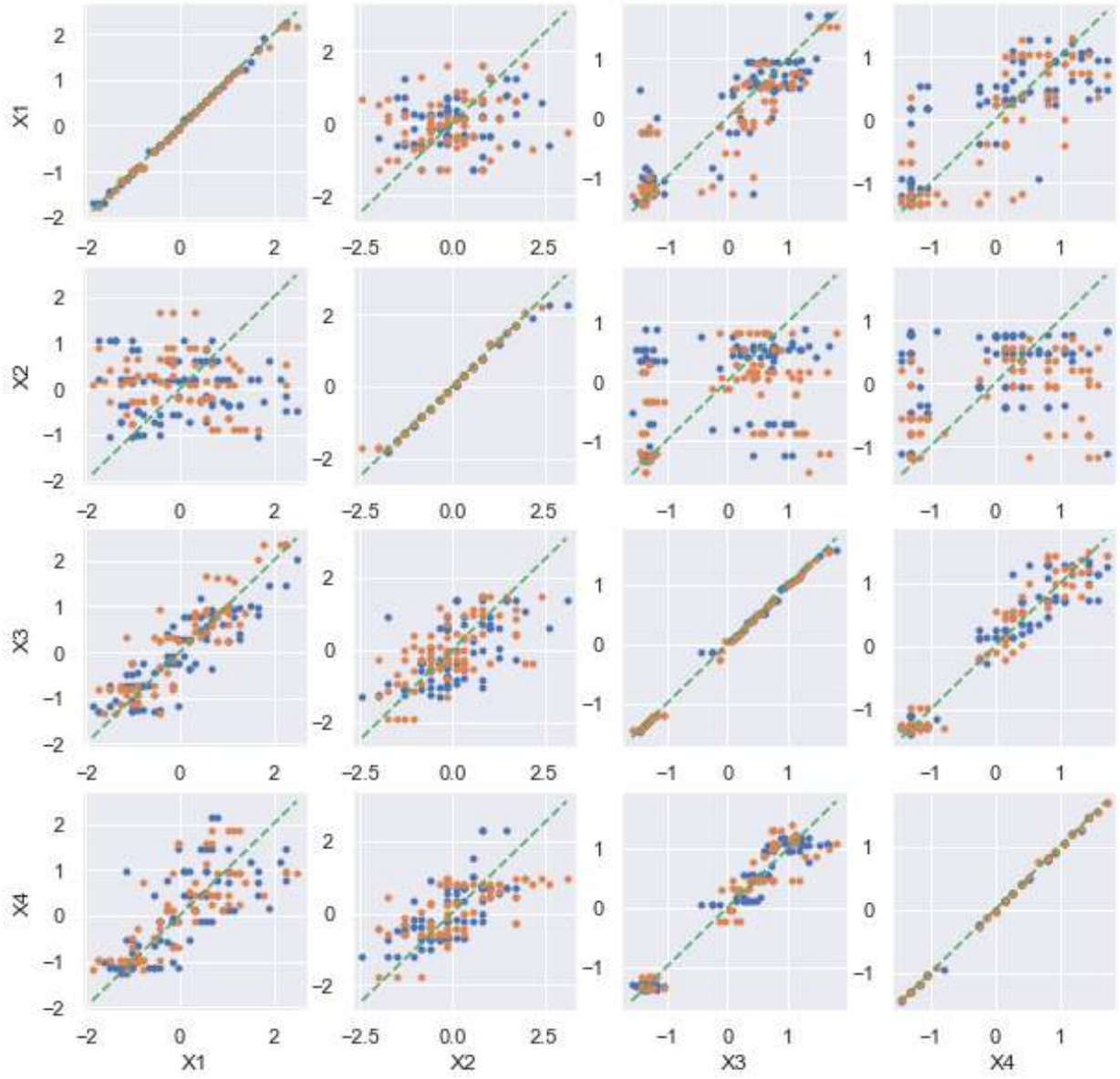
```
import matplotlib.pyplot as plt

def pairplot_cross_val(data, model=None, ax=None, **params):
    if ax is None:
        fig, ax = plt.subplots(data.shape[1], data.shape[1], figsize=params.get(
↪'figsize', (10,10)))
    if 'figsize' in params:
        del params["figsize"]
    if model is None:
        from sklearn.linear_model import LinearRegression
        model = LinearRegression

    df = scale(data)
    cor = numpy.corrcoef(df.T)
    for i in range(cor.shape[0]):
        xi = df[:, i:i+1]
        for j in range(cor.shape[1]):
            xj = df[:, j]
            mem = []
            xi_train, xi_test, xj_train, xj_test = train_test_split(xi, xj, test_
↪size=0.5)
            mod = model(**params)
            mod.fit(xi_train, xj_train)
            v = mod.predict(xi_test)
            mod = model(**params)
            mod.fit(xi_test, xj_test)
            v2 = mod.predict(xi_train)
            ax[i,j].plot(xj_test, v, ".")
            ax[i,j].plot(xj_train, v2, ".")
            if j == 0:
                ax[i,j].set_ylabel(data.columns[i])
            if i == data.shape[1]-1:
                ax[i,j].set_xlabel(data.columns[j])
            mi = min(min(xj_test), min(v), min(xj_train), min(v2))
            ma = max(max(xj_test), max(v), max(xj_train), max(v2))
            ax[i,j].plot([mi, ma], [mi, ma], "--")
    return ax

ax = pairplot_cross_val(df)
ax;
```



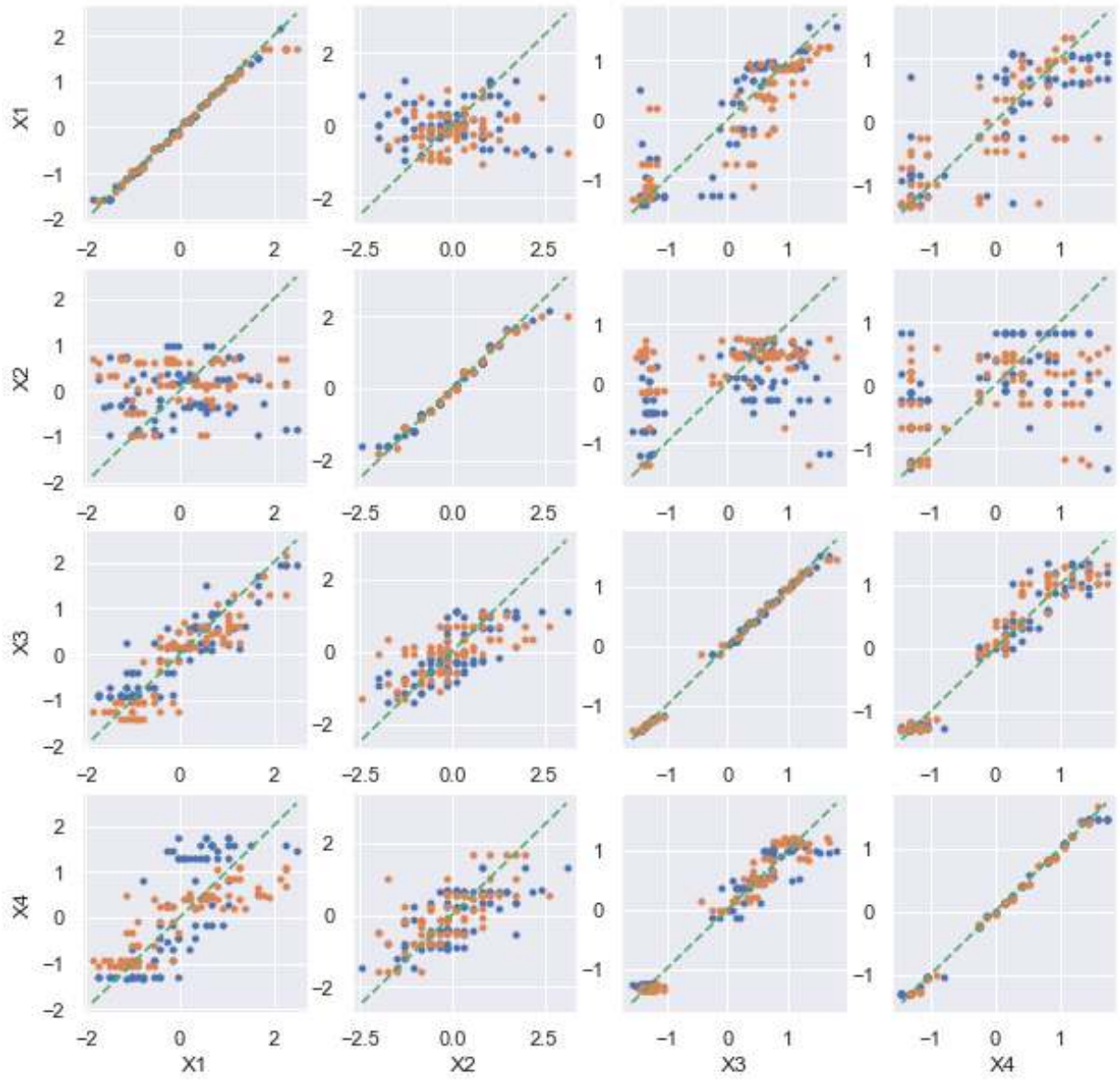

```
ax = pairplot_cross_val(df, model=DecisionTreeRegressor)
ax;
```




```
ax = pairplot_cross_val(df, model=RandomForestRegressor, n_estimators=10)
ax;
```




```
from sklearn.neighbors import KNeighborsRegressor
ax = pairplot_cross_val(df, model=KNeighborsRegressor)
ax;
```



9.2.5 Corrélations de variables catégorielles

C'est le problème épineux si on se restreint au linéaire. Cela n'a pas trop de sens d'affecter une valeur à chaque catégorie et la corrélation de deux variables binaires (des modalités) est toujours étrange car il n'y a que deux valeurs possibles.

$$cov(X,Y) = \mathbb{E}\left[(X - \mathbb{E}X)(Y - \mathbb{E}Y)\right] = \mathbb{E}(XY) - \mathbb{E}X\mathbb{E}Y = \mathbb{P}(X = 1 \, et \, Y = 1) - \mathbb{E}X\mathbb{E}Y$$

Dans le cas de variables binaires générées de modalités de la même variables catégorielles, le premier terme est toujours nul puisque les modalités sont exclusives et la corrélation est toujours négative.

```
import random
ex = numpy.zeros((100, 2))
for i in range(0, ex.shape[0]):
```

(suite sur la page suivante)

(suite de la page précédente)

```
    h = random.randint(0, ex.shape[1]-1)
    ex[i, h] = 1
ex[:5]
```

```
array([[0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

```
numpy.corrcoef(ex.T)
```

```
array([[ 1., -1.],
       [-1.,  1.]])
```

```
import random
ex = numpy.zeros((100, 3))
for i in range(0, ex.shape[0]):
    h = random.randint(0, ex.shape[1]-1)
    ex[i, h] = 1
ex[:5]
numpy.corrcoef(ex.T)
```

```
array([[ 1.        , -0.55708601, -0.40824829],
       [-0.55708601,  1.        , -0.53066863],
       [-0.40824829, -0.53066863,  1.        ]])
```

Supposons maintenant que nous avons deux variables catégorielles très proches :

— X_1 est une couleur rouge, bleu, gris.
— X_2 est une nuance rose, orange, cyan, magenta, blanc noir.

```
c1 = ["rouge", "bleu", "gris"]
c2 = ["rose" ,"orange" ,"cyan" ,"magenta", "blanc", "noir"]
ind = [random.randint(0, 2) for i in range(0, 100)]
x1 = [c1[i] for i in ind]
x2 = [c2[i*2 + random.randint(0,1)] for i in ind]
df = pandas.DataFrame(dict(X1=x1, X2=x2))
df.head()
```

On peut évidemment transformer en entier.

```
dummies = pandas.get_dummies(df)
dummies.head()
```

```
dummies.corr()
```

Ca ne dit pas grand-chose.

```
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
df["X1e"] = enc.fit_transform(df["X1"])
df["X2e"] = enc.fit_transform(df["X2"])
df.head()
```

```
df.corr()
```

Ca ne veut toujours pas dire grand-chose. Et si on change la première colonne en permutant les lables :

```
df["X1e"] = df["X1e"].apply(lambda i: (i+1)%3)
df.head()
```

```
df.corr()
```

La corrélation linéaire sur des variables catégorielles n'a pas de sens. Essayons avec un arbre de décision. C'est le modèle adéquat pour ce type de valeur discrètes :

```
cor = correlation_cross_val(df[["X1e", "X2e"]], DecisionTreeRegressor)
cor
```

```
c:python370_x64libsite-packagesipykernel_launcher.py:6: DataConversionWarning:␣
↪Data with input dtype int32, int64 were all converted to float64 by the␣
↪scale function.
```

Et si on permute le premier label :

```
df["X1e"] = df["X1e"].apply(lambda i: (i+1)%3)
correlation_cross_val(df[["X1e", "X2e"]], DecisionTreeRegressor)
```

```
c:python370_x64libsite-packagesipykernel_launcher.py:6: DataConversionWarning:␣
↪Data with input dtype int32, int64 were all converted to float64 by the␣
↪scale function.
```

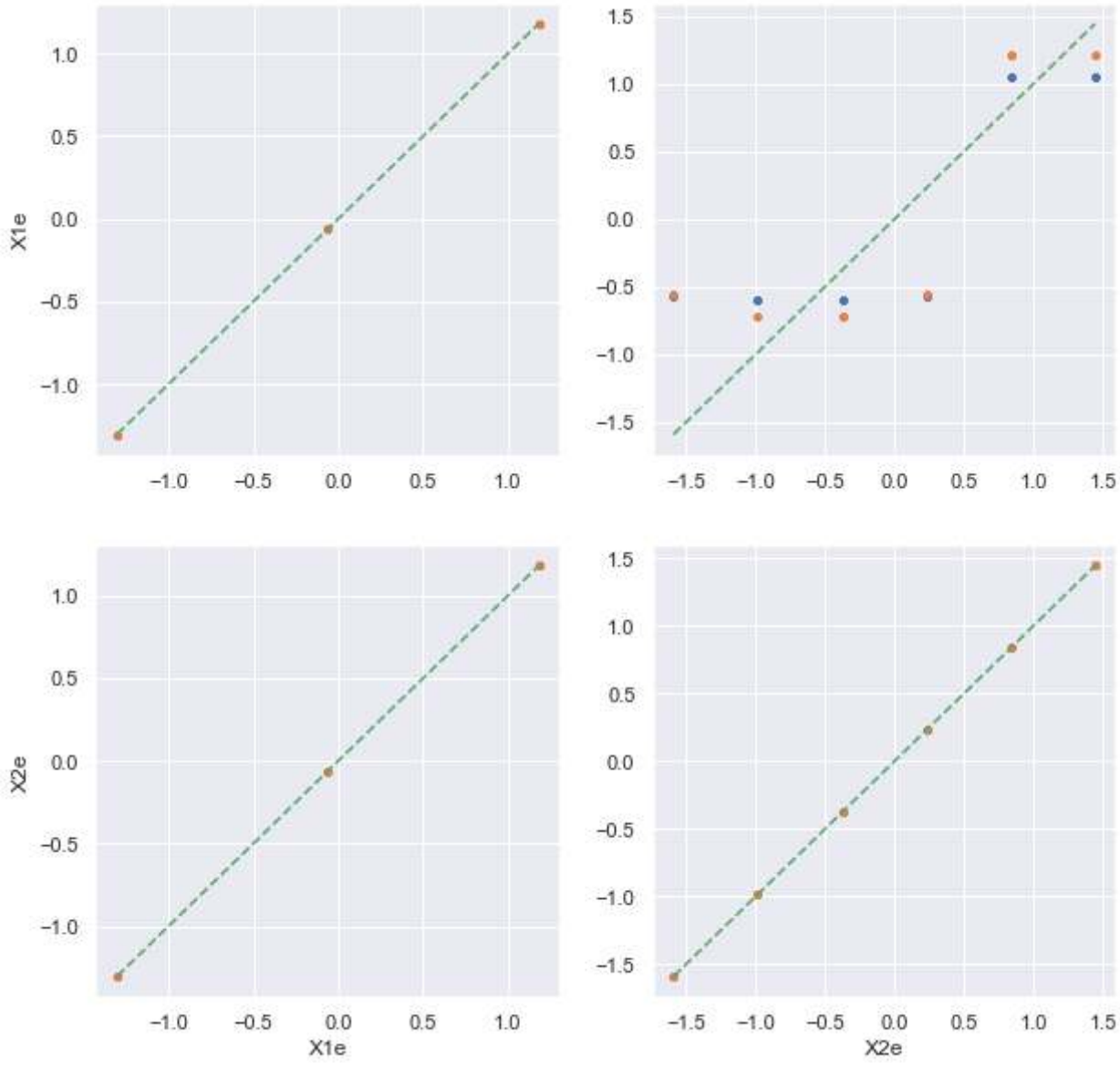
Même résultat qui s'interprète de la sorte :

— La variable *X1e* se déduit de *X2e* (car *cor(X2e, X1e) = 1*).
— La variable *X2e* et fortement lié à *X2e*.

La valeur numérique choisie pour représente la variable catégorielle n'a pas d'impact sur les résultats.

```
ax = pairplot_cross_val(df[["X1e", "X2e"]], model=DecisionTreeRegressor)
ax;
```

```
c:python370_x64libsite-packagesipykernel_launcher.py:12:␣
↪DataConversionWarning: Data with input dtype int32, int64 were all␣
↪converted to float64 by the scale function.
  if sys.path[0] == '':
```



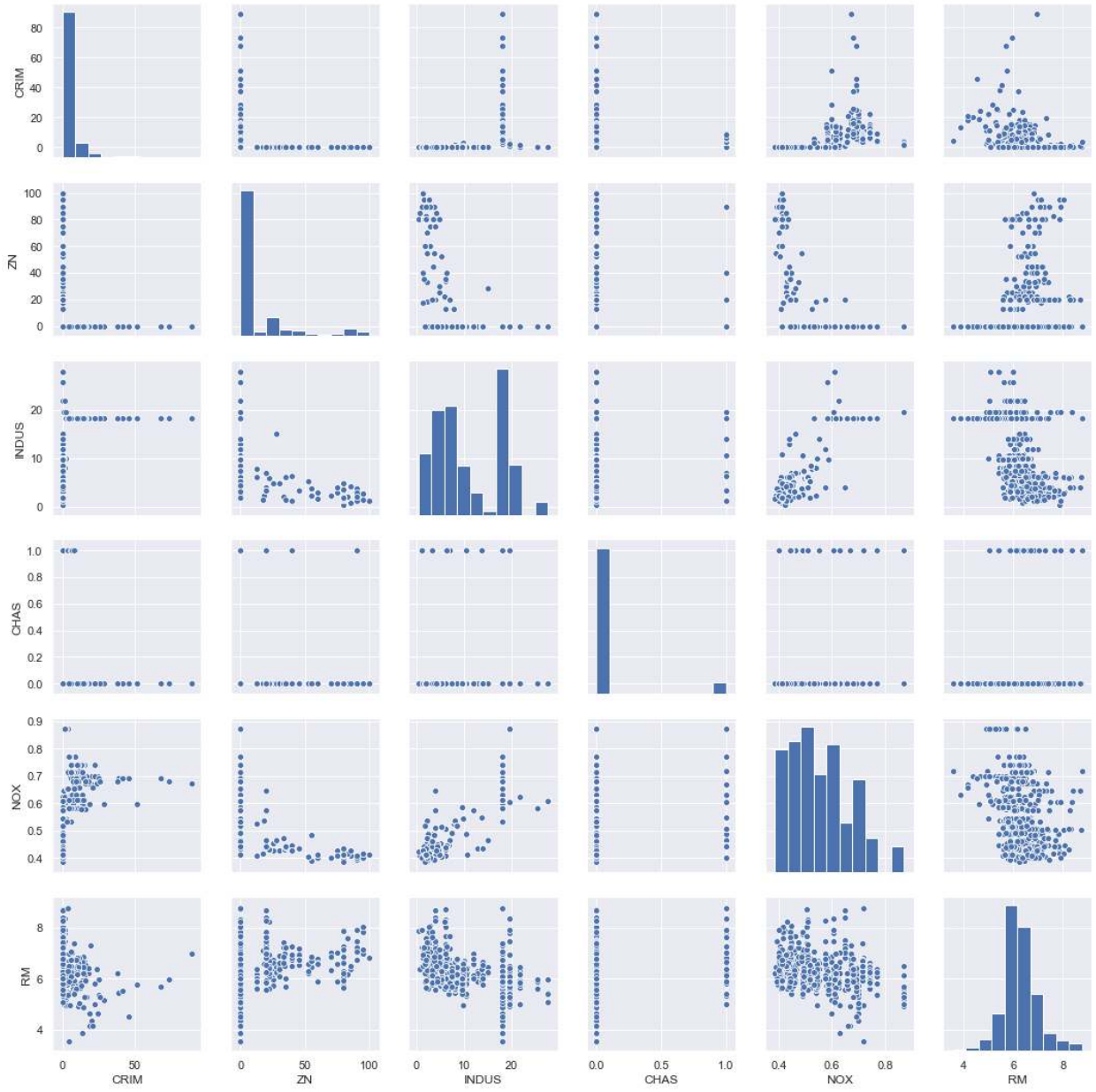
Et sur un jeu de données plus complet.

```
from sklearn.datasets import load_boston
df = load_boston()
df = pandas.DataFrame(df.data, columns=df.feature_names)
df.head()
```

```
df.corr()
```

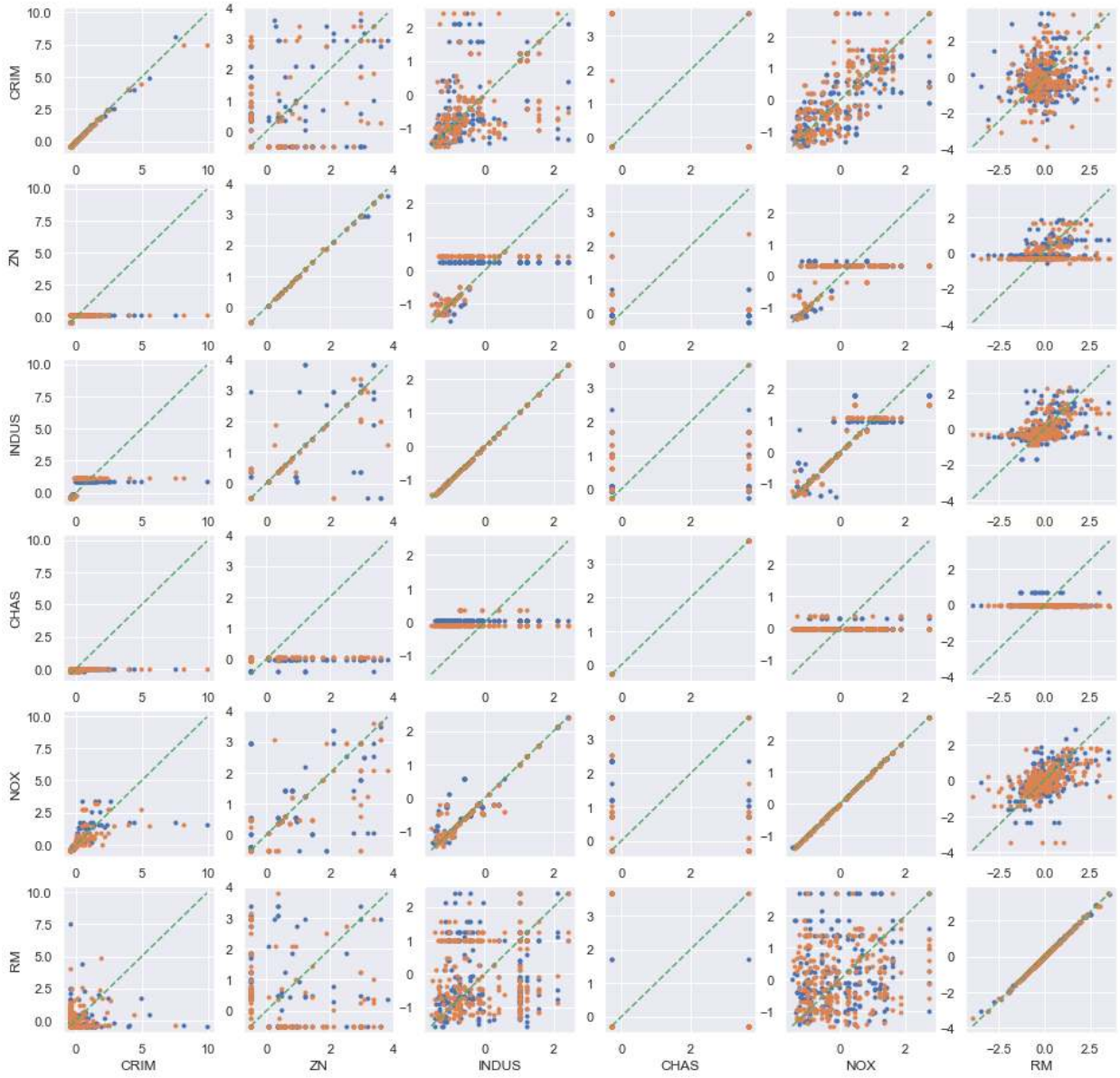
On dessine les 5 premières variables. On voit que la variable CHAS est binaire.

```
sns.pairplot(df[df.columns[:6]]);
```



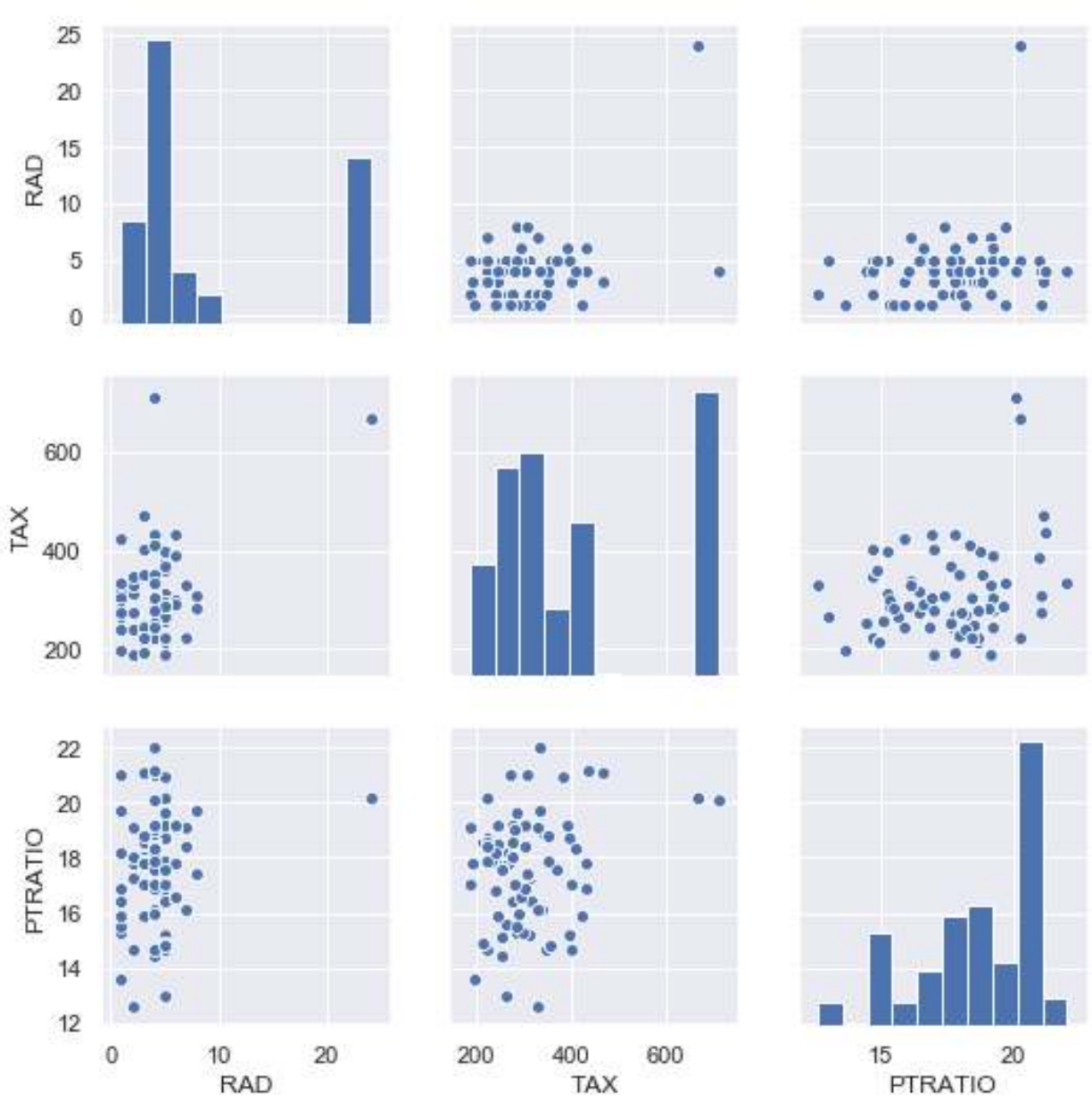

```
correlation_cross_val(df, DecisionTreeRegressor)
```

```
pairplot_cross_val(df[df.columns[:6]], model=DecisionTreeRegressor, figsize=(16,16));
```



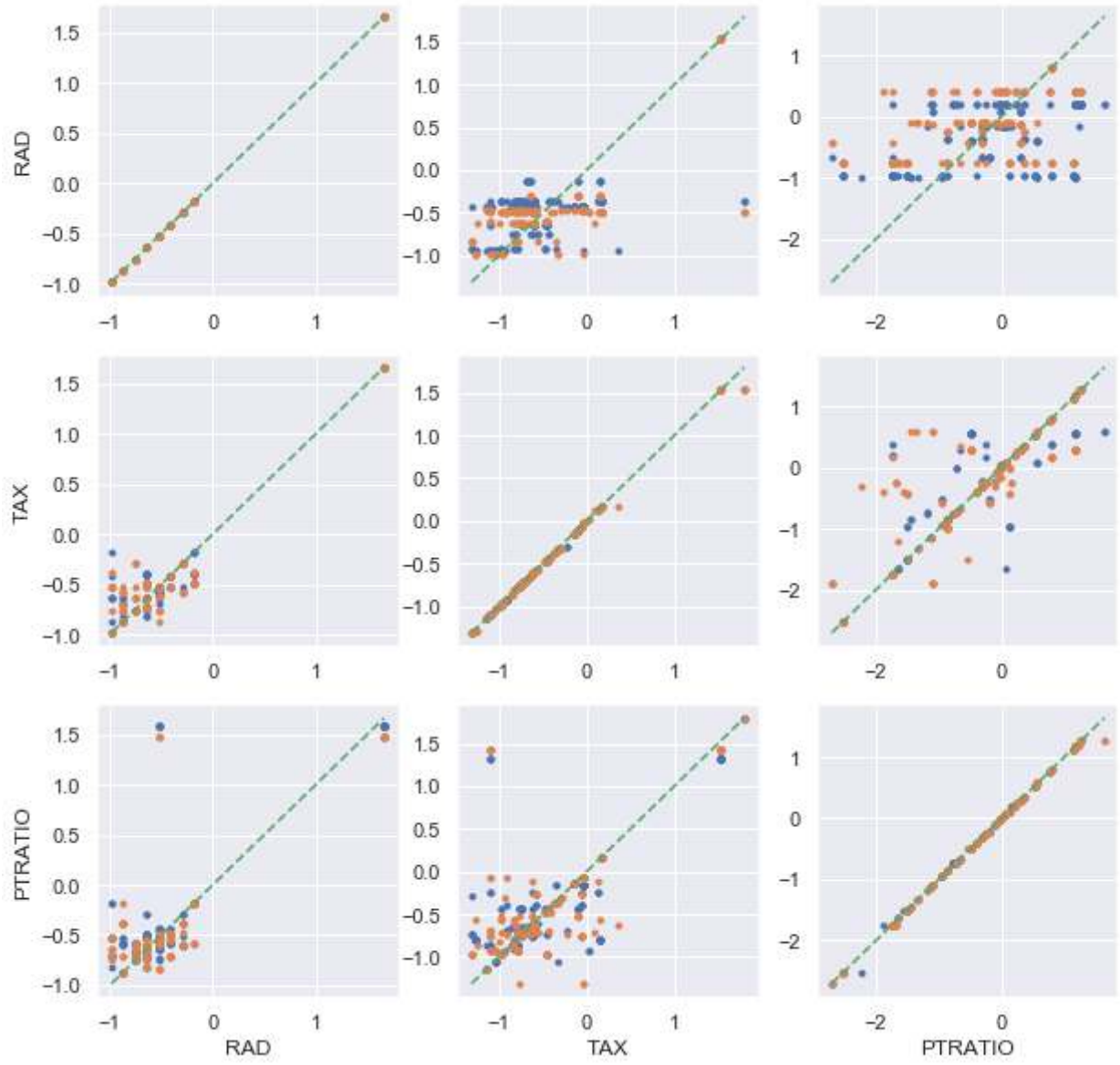
On regarde en pariculier les variables TAX, RAD, PTRATIO.

```
sns.pairplot(df[["RAD", "TAX", "PTRATIO"]]);
```




```
df[["RAD", "TAX", "PTRATIO"]].corr()
```

```
pairplot_cross_val(df[["RAD", "TAX", "PTRATIO"]], model=DecisionTreeRegressor);
```




```
correlation_cross_val(df[["RAD", "TAX", "PTRATIO"]], DecisionTreeRegressor)
```

Les variables sont toutes trois liées de façon non linéaire.

9.2.6 Maximal information coefficient

Cette approche est plutôt pragmatique mais peut se révéler coûteuse en terme de calculs. Elle permet aussi de comprendre qu'un coefficient de corrélation dépend des hypothèses qu'on choisi pour les données. On peut toujours construire un coefficient de corrélation qui soit égal à 1 mais il correspond à toujours à un phénomène qu'on souhaite étudier. La corrélation linéaire recherche des relations linéaires. On peut chercher une relation polynomiale. Les arbres de décision recherche une corrélation construite à partir de fonction en escalier. Plus la relation a de degré de liberté, plus le coefficient a de chance de tendre vers 1, moins il a de chance d'être aussi élevé sur de nouvelles données.

Cela permet néanmoins de mieux comprendre les avantages et les inconvénients de métriques du type MIC [156] ou

156. https://en.wikipedia.org/wiki/Maximal_information_coefficient

Maximal information coefficient. Plus de détails sont disponibles dans cet article : Equitability, mutual information, and the maximal information coefficient [157]. Le module minepy [158] implémente cette métrique ainsi que d'autres qui poursuivent le même objectif. L'information mutuelle est définie comme ceci pour deux variables discrètes :

$$MI(X,Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \ln_2 \frac{p(x,y)}{p(x)p(y)}$$

La fonction $p(x,y)$ définit la distribution conjointe des deux variables, $p(x)$, $p(y)$ les deux probabilités marginales. Il existe une extension pour les variables continues :

$$MIC(X,Y) = \int_{x \in \mathcal{X}} \in_{y \in \mathcal{Y}} p(x,y) \ln_2 \frac{p(x,y)}{p(x)p(y)} dxdy$$

Une façon de calculer une approximation du coefficient $MIC(x,y)$ est de discrétiser les deux variables X et Y ce qu'on fait en appliquant un algorithme similaire à celui utilisé pour construire un arbre de décision à ceci près que qu'il n'y a qu'une seule variable et que la variable à prédire est elle-même.

L'information mutuelle est inspiré de la distance de Kullback-Leiber [159] qui est une distance entre deux probabilités qui sont ici la disribution du couple (X,Y) et la distribution que ce couple aurait si les deux variables étaient indépendantes, c'est à dire le produit de leur distribution.

Je reproduis ici le code de l'exemple proposé par la librairie minepy [160] et j'y ajoute la corrélation proposée dans ce notebook `DT` pour *Decision Tree*.

```
%matplotlib inline
```

```
import numpy as np
import matplotlib.pyplot as plt
from minepy import MINE


rs = np.random.RandomState(seed=0)

def mysubplot(x, y, numRows, numCols, plotNum,
              xlim=(-4, 4), ylim=(-4, 4)):

    r = np.around(np.corrcoef(x, y)[0, 1], 1)
    mine = MINE(alpha=0.6, c=15, est="mic_approx")
    mine.compute_score(x, y)
    mic = np.around(mine.mic(), 1)

    # début ajout
    df = pandas.DataFrame(dict(x=x, y=y))
    cor = correlation_cross_val(df, DecisionTreeRegressor)
    dt = max(cor.iloc[1,0], cor.iloc[0,1])

    ax = plt.subplot(numRows, numCols, plotNum,
                     xlim=xlim, ylim=ylim)
    ax.set_title('Pearson r=%.1f\nMIC=%.1f\nDT=%.1f' % (r, mic, dt),fontsize=10)
    ax.set_frame_on(False)
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)
    ax.plot(x, y, ',')
    ax.set_xticks([])
```

(suite sur la page suivante)

157. https://arxiv.org/abs/1301.7745v1
158. http://minepy.readthedocs.io/en/latest/python.html
159. https://fr.wikipedia.org/wiki/Divergence_de_Kullback-Leibler
160. http://minepy.readthedocs.io/en/latest/python.html#second-example

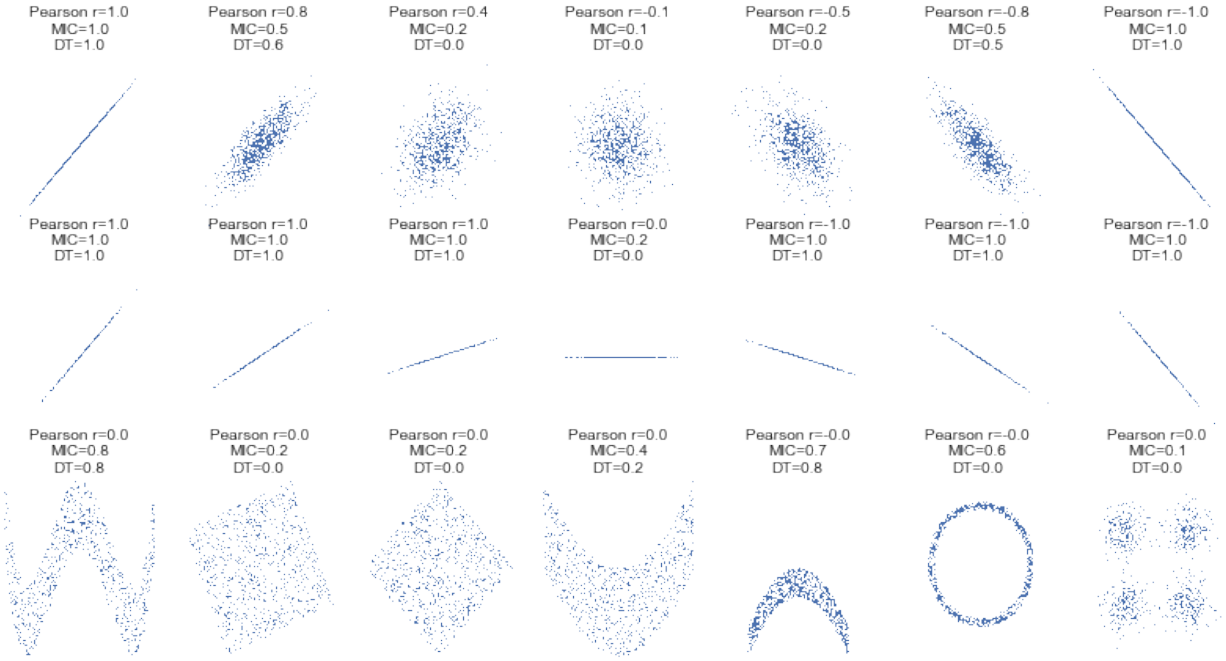
(suite de la page précédente)

```
    ax.set_yticks([])
    return ax

def rotation(xy, t):
    return np.dot(xy, [[np.cos(t), -np.sin(t)], [np.sin(t), np.cos(t)]])

def mvnormal(n=1000):
    cors = [1.0, 0.8, 0.4, 0.0, -0.4, -0.8, -1.0]
    for i, cor in enumerate(cors):
        cov = [[1, cor],[cor, 1]]
        xy = rs.multivariate_normal([0, 0], cov, n)
        mysubplot(xy[:, 0], xy[:, 1], 3, 7, i+1)

def rotnormal(n=1000):
    ts = [0, np.pi/12, np.pi/6, np.pi/4, np.pi/2-np.pi/6,
          np.pi/2-np.pi/12, np.pi/2]
    cov = [[1, 1],[1, 1]]
    xy = rs.multivariate_normal([0, 0], cov, n)
    for i, t in enumerate(ts):
        xy_r = rotation(xy, t)
        mysubplot(xy_r[:, 0], xy_r[:, 1], 3, 7, i+8)

def others(n=1000):
    x = rs.uniform(-1, 1, n)
    y = 4*(x**2-0.5)**2 + rs.uniform(-1, 1, n)/3
    mysubplot(x, y, 3, 7, 15, (-1, 1), (-1/3, 1+1/3))

    y = rs.uniform(-1, 1, n)
    xy = np.concatenate((x.reshape(-1, 1), y.reshape(-1, 1)), axis=1)
    xy = rotation(xy, -np.pi/8)
    lim = np.sqrt(2+np.sqrt(2)) / np.sqrt(2)
    mysubplot(xy[:, 0], xy[:, 1], 3, 7, 16, (-lim, lim), (-lim, lim))

    xy = rotation(xy, -np.pi/8)
    lim = np.sqrt(2)
    mysubplot(xy[:, 0], xy[:, 1], 3, 7, 17, (-lim, lim), (-lim, lim))

    y = 2*x**2 + rs.uniform(-1, 1, n)
    mysubplot(x, y, 3, 7, 18, (-1, 1), (-1, 3))

    y = (x**2 + rs.uniform(0, 0.5, n)) * \
        np.array([-1, 1])[rs.randint(0, 1, size=n)]
    mysubplot(x, y, 3, 7, 19, (-1.5, 1.5), (-1.5, 1.5))

    y = np.cos(x * np.pi) + rs.uniform(0, 1/8, n)
    x = np.sin(x * np.pi) + rs.uniform(0, 1/8, n)
    mysubplot(x, y, 3, 7, 20, (-1.5, 1.5), (-1.5, 1.5))

    xy1 = np.random.multivariate_normal([3, 3], [[1, 0], [0, 1]], int(n/4))
    xy2 = np.random.multivariate_normal([-3, 3], [[1, 0], [0, 1]], int(n/4))
    xy3 = np.random.multivariate_normal([-3, -3], [[1, 0], [0, 1]], int(n/4))
    xy4 = np.random.multivariate_normal([3, -3], [[1, 0], [0, 1]], int(n/4))
    xy = np.concatenate((xy1, xy2, xy3, xy4), axis=0)
    mysubplot(xy[:, 0], xy[:, 1], 3, 7, 21, (-7, 7), (-7, 7))

plt.figure(figsize=(14,7))
mvnormal(n=800)
```

(suite sur la page suivante)

(suite de la page précédente)

```
rotnormal(n=200)
others(n=800)
# plt.tight_layout()
# plt.show()
```



9.3 File d'attente, un petit exemple

Psychokinèse, les ampoules grillent à distance

9.3.1 Petite histoire

Cet énoncé s'inspire du livre Devenez sorciers, devenez savants [161] de Georges Charpak et Henri Broch dont est tiré l'extrait suivant.

> Le présentateur se tourne vers la caméra principale, et d'un air très sérieux et enjôleur, regarde le téléspectateur droit dans les yeux en déclarant : *Allez-y ! Allumez cinq ou six lampes à côté de vous.* Puis il se tourne vers le médium et demande : *Vous pensez réellement pouvoir le faire ?* Après quelques moments d'hésitation, le mage se prononce : *J'espère avoir suffisamment de concentration ce soir, mais je ne suis*

161. https://fr.wikipedia.org/wiki/Devenez_sorciers,_devenez_savants

pas dans les conditions vraiment idéales ; pour produire ce genre de phénomène à distance, d'habitude, je me retire pendant plusieurs jours dans une solitude totale et une profonde obscurité, après un jeûne strict.

Si jamais il échoue, le public le mettra au compte des circonstances et non pas de ces compétences. Et, pourtant, le médium n'échoue pas. Des ampoules grillent chez les téléspectateurs qui regardent cette émission. Ils font part au standard téléphonique de la chaîne qui diffuse en direct cet extraordinaire moment de culture. Le médium a donc bien réussi - comme il le prétendait -, en concentrant sa puissance spirituelle sur la matière, à griller des ampoules électriques à distance.

Supposons que cette émission soit suivie par environ un million de téléspectateurs et qu'elle dure une heure ou plus. Cela signifie qu'environ cinq à six millions d'ampoules ont été allumées pendant une heure ou plus. Supposons que ce nombre soit de deux millions.

La durée de vie moyenne d'une ampoule à incandescence est de mille heures. Ce qui signifie que, pendant la durée de l'émission, il y aura environ deux milles lampes grillées.

9.3.2 Un peu de théorie

Ce problème ressemble à un problème de files d'attente. Ce dernier consiste à déterminer un nombre adéquat de guichets en fonction de la vitesse de remplissage d'une file d'attente afin de limiter le temps d'attente. On désigne souvent cette problématique par un sigle du type $M/M/S$. Le premier M signifie qu'on suppose que la probabilité que n personnes arrivent pendant une durée t suit une loi de Poisson de paramètre λt. Le second M signifie qu'on suppose que le temps de traitement d'une personne par un guichetier suit une loi exponentielle de paramètre μ. S désigne le nombre de guichets. Pour de tels problèmes, on cherche à déterminer la probabilité que le système (file d'attente + guichets) contienne n personnes. De cette probabilité, on peut déduire par exemple le temps d'attente moyen pour une personne qui entre dans la file d'attente. On suppose que le système est stationnaire, les probabilités ne dépendent pas du temps.

Définition D1 : loi de Poisson et loi exponentielle

Si une variable X suit une loi de Poisson de paramète λt, elle a pour densité :

$$\mathbb{P}(X = n) \quad = \quad \frac{(\lambda t)^n}{n!} \, e^{-\lambda t} \tag{9.1}$$

Si une variable X suit une loi exponentielle de paramètre μ, elle a pour densité :

$$f(t) \quad = \quad \mu \, e^{-\mu t} \text{ et } \mathbb{P}(X \leqslant t) = \int_0^t \mu \, e^{-\mu x} dx = 1 - e^{-\mu t} \tag{9.2}$$

File d'attente, cas M/M/1

On s'intéresse d'abord à un système $M/M/1$. Il n'y a donc qu'un seul guichet. λ est le nombre moyen d'arrivée par unité de temps tandis que μ est le nombre moyen de clients traités par unité de temps. On suppose également que $\lambda < \mu$. Dans le cas contraire, la file d'attente devient infinie. On désigne par $p_n(t)$ la probabilité que la file d'attente contiennent n personne. Que vaut cette probabilité à l'instant $t + dt$?

On considère que pendant la durée $t + dt$, au plus une personne peut s'ajouter à la file d'attente et au plus une personne peut être traitée par un guichetier. Les autres cas sont négligeables. On désigne par $B(x, t, dt)$ le fait qu'une personne quitte un guichet pendant les instants t et $t + dt$ sachant qu'elle est arrivée au guichet à l'instant x. On cherche à déterminer la probabilité $\mathbb{P}(B(x, t, dt))$. On peut dire aussi que $\mathbb{P}(B(x, t, dt)$ est la probabilité que le temps de traitement d'une personne est inférieur à $t + dt - x$ sachant qu'il est supérieur à $t - x$. Si D est une variable de durée

suivant une loi exponentielle, alors :

$$\begin{aligned}
\mathbb{P}\left(B(x,t,dt)\right) &= \mathbb{P}\left(D \leqslant t+dt-x | D > t-x\right) \\
&= \frac{\mathbb{P}\left(t+dt-x \geqslant D > t-x\right)}{\mathbb{P}\left(D > t-x\right)} \\
&= \frac{\int_{t-x}^{t+dt-x} \mu e^{-\mu u} du}{\int_{t-x}^{\infty} \mu e^{-\mu u} du} = \frac{e^{-\mu(t-x)} - e^{-\mu(t-x+dt)}}{e^{-\mu(t-x)}} \\
&= 1 - e^{-\mu dt} \\
\mathbb{P}\left(B(x,t,dt)\right) &= -\mu dt + o(dt)
\end{aligned}$$

Cette probabilité ne dépend ni de x, ni de t. En ce qui concerne les arrivées, la probabilité qu'une personne arrive pendant les instants t et $t+dt$ ne dépend pas du passé et suit une loi de Poisson de paramètre λ. Cette constation et l'équation *(1)* (page **??**) nous permettent d'écrire que :

$$\mathbb{P}\left(\text{une personne arrive pendant } dt\right) = \lambda dt\, e^{-\lambda dt} \sim \lambda dt + o(dt) \tag{9.3}$$

$$\mathbb{P}\left(\text{une personne part pendant } dt\right) = 1 - e^{-\mu dt} \sim \mu dt + o(dt) \tag{9.4}$$

De plus, pendant la durée dt, quatre cas sont possibles :

— Une personne peut arriver sans qu'aucune ne parte d'un guichet.
— Une personne peut partir d'un guichet sans qu'aucune autre n'arrive.
— Une personne peut arriver et une autre partir d'un guichet.
— Aucune personne n'arrive et aucune ne part d'un guichet.

Ces quatre cas permettent d'écrire la relation suivante :

$$\begin{aligned}
p_n(t+dt) &= p_{n-1}(t)\ \lambda dt + p_{n+1}(t)\ \mu dt + \\
&\quad p_n(t)\left(\mu dt \lambda dt\right) + p_n(t)\left(1-\mu dt\right)\left(1-\lambda dt\right)
\end{aligned}$$

On néglige les termes du second degré en $(dt)^2$:

$$\begin{aligned}
p_n(t+dt) &= p_{n-1}(t)\ \lambda dt + p_{n+1}(t)\ \mu dt + p_n(t)\left(1-\mu dt-\lambda dt\right) \\
\iff \frac{p_n(t+dt)-p_n(t)}{dt} &= \lambda p_{n-1}(t) + \mu p_{n+1}(t) - \left(\mu+\lambda\right) p_n(t)
\end{aligned}$$

Cette relation n'est vraie que lorsque $n > 0$, lorsque $n = 0$, aucune personne déjà présente ne peut être traitée par un guichetier, on a donc :

$$\frac{p_0(t+dt)-p_0(t)}{dt} = \mu p_1(t) - \lambda p_0(t)$$

Comme le système est stationnaire, toutes les dérivées sont nulles. Les probabilités ne dépendent pas du temps. Ceci donne les relations suivantes :

$$\begin{aligned}
&\left\{\begin{array}{lcl} \lambda p_{n-1} + \mu p_{n+1} - \left(\mu+\lambda\right) p_n &=& 0 \\ \mu p_1 - \lambda p_0 &=& 0 \end{array}\right. \\
\iff\quad &\left\{\begin{array}{lcl} \mu p_{n+1} &=& \left(\mu+\lambda\right) p_n - \lambda\, p_{n-1} \\ \mu p_1 &=& \lambda p_0 \end{array}\right.
\end{aligned}$$

On vérifie par récurrence que :

$$p_n = \left(\frac{\lambda}{\mu}\right)^n p_0$$

Il reste à déterminer p_0. Or, comme $\sum_0^\infty p_n = 1 = p_0\ \sum_0^\infty \left(\frac{\lambda}{\mu}\right)^n$, on obtient que *frac{p_0}{ 1 - frac{lambda}{mu}} = 1 Longleftrightarrow p_0 = 1 - frac{lambda}{mu}*.

Exemple :

On suppose qu'un médecin traite en moyenne quatre patients par heure tandis qu'il accepte trois rendez-vous par heure. Donc $\lambda = 3$ et $\mu = 4$. Le nombre moyen $\overline{n}$ de patients dans sa salle d'attente est donné par :

$$\overline{n} = \sum_0^\infty np_n = \left(1 - \frac{\lambda}{\mu}\right) \sum_0^\infty n \left(\frac{\lambda}{\mu}\right)^n = \frac{\frac{\lambda}{\mu}}{1 - \frac{\lambda}{\mu}} = \frac{\lambda}{\mu - \lambda}$$

Il y a donc en moyenne trois personnes dans la salle d'attente de ce médecin. Comme le temps moyen de traitement de chacun est $\frac{1}{\mu}$, le temps moyen d'attente d'un patient arrivant dans la salle d'attente est $\frac{\lambda\mu}{\mu - \lambda}$. Ce temps est égal à trois quarts d'heure pour cet exemple.

File d'attente, cas M/M/S

Le système contient dorénavant S guichets, on suppose que la vitesse μ de traitement des clients est commune à tous les guichets. On cherche tout d'abord la probabilité qu'une personne s'en aille parmi les k qui occupent un guichet. On désigne par $(D_1, \ldots, D_k)$ k variables indépendantes suivant une loi exponentielle de paramètre μ, pendant un temps dt, la probabilité qu'une personne parmi k quitte un guichet est :

$$\begin{aligned}
\mathbb{P}\left(\min\left\{D_1, \ldots, D_k\right\} \leqslant dt\right) &= 1 - \mathbb{P}\left(\min\left\{D_1, \ldots, D_k\right\} > dt\right) \\
&= 1 - \left[\prod_{n=1}^k \mathbb{P}\left(D_n > dt\right)\right] \\
&= 1 - \left[\prod_{n=1}^k 1 - \mathbb{P}\left(D_n \leqslant dt\right)\right] \\
&= 1 - \left[\prod_{n=1}^k e^{-\mu dt}\right] \\
&= 1 - e^{-k\mu dt} \sim k\mu dt + o(dt)
\end{aligned}$$

Pour déterminer les probabilités $(p_n)_n$, on applique le même raisonnement que pour un système $M/M/1$ en distinguant les cas $n \leqslant S$ et $n > S$. On adapte la récurrence donnée par le système d'équations *(4)* (page **??**) au cas $M/M/S$:

$$\left\{\begin{array}{lcl}
\mu p_1 - \lambda p_0 & = & 0 \\
\lambda p_{n-1} + (n+1)\,\mu p_{n+1} - (n\mu + \lambda)\,p_n & = & 0 \text{ si } 1 \leqslant n < S \\
\lambda p_{n-1} + S\mu p_{n+1} - (S\mu + \lambda)\,p_n & = & 0 \text{ si } n \geqslant S
\end{array}\right.$$

On suppose que $\frac{\lambda}{S\mu} < 1$ afin que la file d'attente ne devienne infinie. Comme pour un système $M/M/1$, ces formules de récurrences et le fait que $\sum_0^\infty p_n = 1$ permet de déduire que :

$$\left\{\begin{array}{lcl}
p_0 & = & \dfrac{1}{\frac{\left(\frac{\lambda}{\mu}\right)^S}{S!\left(1 - \frac{\lambda}{S\mu}\right)} + \sum_{k=1}^{S-1} \frac{\left(\frac{\lambda}{\mu}\right)^n}{n!}} \\
p_n & = & \dfrac{1}{n!} \left(\dfrac{\lambda}{\mu}\right)^n p_0 \quad \text{si } n < S \\
p_n & = & \dfrac{1}{S^{n-S} S!} \left(\dfrac{\lambda}{\mu}\right)^n p_0 \quad \text{si } n \geqslant S
\end{array}\right.$$

Ces calculs sont utilisés pour optimiser le nombre de guichets. Chaque guichetier a un coût qui doit être comparé avec le coût associé au temps d'attente des clients. Ces résultats sont extraits du livre *[Faure2000]* (page **??**).

La théorie des files d'attente remonte aux premiers travaux de K. Erlang (1917), sur le calcul du nombre d'organes de chaque type à installer dans un central téléphonique automatique. Développée aussi par Engset (1918), cette théorie s'est amplifiée sous l'impulsion de nombreux chercheurs (E. Borel, D. Kendall, A. Kolmogorov, Khintchine, LC. Palm, F. Pollaczek, L. Feller, ...). Les informaticiens l'utilisent notamment pour l'évaluation de performances, à titre prévisionnel, des systèmes ou des réseaux informatiques.

Retour aux ampoules

La durée de traitement d'un client fait penser à la durée de vie d'une ampoule. Les lampes font office de guichets tandis que le rôle des clients est joué par des lumières. Toutefois, ce n'est pas le temps d'attente moyen ni la longueur moyenne de la file d'attente qui nous intéresse mais, en quelque sorte, le nombre de clients qui sont traités pendant une durée T. En fait, plus exactement, c'est le nombre de guichets qui auront traités au moins un client pendant une durée T qui nous intéresse. Il correspond exactement au nombre d'ampoules qui vont griller pendant cette même période T. Il reste à définir ce qu'est une file d'attente d'ampoules et surtout son paramètre λ.

Lorsqu'une ampoule grille, elle est a priori changée dans les plus brefs délais, comme si la file d'attente des ampoules était infinie. Ceci signifie que $\lambda >> \mu$, configuration qui sort du champ d'application des files d'attente $M/M/S$. Le paramètre λ n'a a priori aucun rôle à jouer. On peut néanmoins s'inspirer de la méthode développée dans les paragraphes précédants pour aborder le problème des ampoules.

On suppose que la durée de vie d'une ampoule suit toujours une loi exponentielle de paramètre μ et qu'il y en a exactement S qui brillent en même temps. On note $p_n(t)$ la probabilité que n ampoules aient grillées à l'instant t. Si $N(t)$ est le nombre d'ampoules grillées à l'instant t : $p_n(t) = \mathbb{P}(N(t) = n)$. Cette fonction n'est plus stationnaire et décroît avec le temps à partir d'un certain moment. Plus on avance dans le temps, plus le nombre d'ampoules grillées augmente. Par conséquent, la probabilité qu'il y ait n ampoules grillées augmente tout d'abord puis, à partir d'un moment t, elle diminue.}. On utilise un raisonnement similaire à celui qui a permis d'écrire les équations *(2)* (page **??**), *(3)* (page **??**) pour obtenir :

$$\begin{aligned} p_n(t+dt) &= (1 - S\mu dt)\, p_n(t) + S\mu p_{n-1}(t)dt \\ \iff \frac{p_n(t+dt) - p_{n-1}(t)}{dt} &= -S\mu p_n(t) + S\mu p_{n-1}(t) \\ \iff p'_n(t) &= -S\mu p_n(t) + S\mu p_{n-1}(t) \end{aligned}$$

On connaît la fonction $p_0(t)$ puisqu'elle correspond à la probabilité qu'aucune des S ampoules n'ait grillé depuis l'instant~0 jusqu'à l'instant t, par conséquent :

$$\begin{aligned} p_0(t) &= \mathbb{P}(\text{ durée de vie des } S \text{ ampoules soient toutes supérieures à } t) \\ \implies p_0(t) &= \left[\int_t^\infty \mu e^{-\mu u} du\right]^S \\ \implies p_0(t) &= e^{-S\mu t} \end{aligned}$$

L'équation *(6)* (page **??**) permet de définir une suite d'équations différentielles du premier degré :

$$\begin{aligned} p_0(t) &= e^{-S\mu t} \\ p'_1(t) &= -S\mu p_1(t) + S\mu e^{-S\mu t} \\ p'_2(t) &= -S\mu p_2(t) + p_1(t) \\ &\dots \\ p'_n(t) &= -S\mu p_n(t) + S\mu p_{n-1}(t) \end{aligned}$$

On peut donc calculer par récurrence la suite de fonction $(p_n(t))_n$. La solution de l'équation différentielle homogène est $e^{-S\mu t}$. On utilise la méthode de la variation de la constante en posant $p_n(t) = C_n(t)e^{-S\mu t}$. On aboutit à l'équation :

$$\begin{aligned} p'_n(t) &= -S\mu p_n(t) + S\mu p_{n-1}(t) = -S\mu p_n(t) + C'_n(t)e^{-S\mu t} \\ \implies C'_n(t)e^{-S\mu t} &= S\mu p_{n-1}(t) \\ \implies C'_n(t) &= S\mu p_{n-1}(t)e^{S\mu t} \end{aligned}$$

Pour $n = 1$, on obtient $C_1'(t) = S\mu \Longrightarrow C_1'(t) = S\mu t + A_1$. Par conséquent, $p_1(t) = (S\mu t + A_1)\, e^{-S\mu t}$. On sait que $\forall t,\ \sum_0^\infty p_n(t) = 1$ mais cela ne permet pas de déterminer la constante A_1. Néanmoins, en faisant tendre $t \longrightarrow 0$, nécessairement $p_1(t) \longrightarrow 0$. Par conséquent : $A_1 = 0$ et $p_1(t) = S\mu t\ e^{-S\mu t}$. On pose maintenant $p_2(t) = C_2(t)e^{-S\mu t}$. La résolution de l'équation différentielle *(7)* (page **??**) pour $n = 2$ aboutit à :

$$\begin{aligned} C_2'(t) &= S\mu p_1(t)e^{S\mu t} = (S\mu t)^2 \\ \Longrightarrow C_2(t) &= \frac{1}{2}S^2\mu^2t^2 + A_2 \\ \Longrightarrow p_2(t) &= \left(\frac{1}{2}S^2\mu^2t^2 + A_2\right) e^{-S\mu t} \end{aligned}$$

De même, en faisant tendre $t \longrightarrow 0$, on démontre que $A_2 = 0$. En poursuivant ce raisonnement, par récurrence, on démontre que :

$$p_n(t) = \frac{(S\mu t)^n}{n!}\ e^{-S\mu t} \tag{9.5}$$

$p_n(t) = \mathbb{P}\left(N(t) = n\right)$ et d'après l'équation *(8)* (page **??**), la variable aléatoire $N(t)$ suit une loi de Poisson de paramètre $S\mu t$. N est aussi appelé processus de Poisson [162]. L'espérance de N est égale à $\mathbb{E}N(t) = S\mu t$. Pendant une durée T, le nombre moyen d'ampoules grillées est :

$$\mathbb{E}N(t) - N(t-T) = \mathbb{E}N(T) - \mathbb{E}N(t-T) = S\mu T$$

Ce nombre est indépendant du temps t.

Application numérique

Pour des ampoules d'une durée de 1000 heures, le paramètre $\mu = \frac{1}{1000}$, $T = 1$. S'il y a deux millions d'ampoules, le nombre moyen d'ampoules grillées par heure est $S\mu T = 2000$. On retrouve le résultat énoncé.

9.3.3 Programme informatique

La durée de vie de chaque ampoule suit une loi exponentielle de paramètre μ. Il faut donc simuler une telle variable dont la fonction de répartition est $F_\mu(x) = 1 - e^{\mu x}$. On utilise pour cela une propriété qui découle de la fonction de répartition. On note $F_\mu^{-1}(x) = -\frac{\ln(1-x)}{\mu}$. Cette fonction vérifie $F_\mu^{-1}\left(F_\mu(x)\right) = 1$. Or si U est une variable aléatoire uniforme sur $[0,1]$, alors la variable $V = F_\mu^{-1}(U)$ suit la loi exponentielle avec μ pour paramètre. Effectivement, $\mathbb{P}\left(V \leqslant t\right) = \mathbb{P}\left(F_\mu^{-1}(U) \leqslant t\right) = \mathbb{P}\left(U \leqslant F_\mu(t)\right) = F_\mu(x)$. La fonction de répartition de la variable V est F_μ, V est donc une loi exponentielle de paramètre μ. La première fonction simule une variable exponentielle de paramètre μ :

<<<

```
import math
import random


def generate_expo(mu):
    x = 0
    while x == 0:
        x = random.random()
    y = - math.log(1-x) / mu
    return y


print(generate_expo(2))
```

162. https://fr.wikipedia.org/wiki/Processus_de_Poisson

>>>

```
    0.14989077700030382
```

Le module random [163] propose aussi une fonction qui simule automatiquement une variable exponentielle.

<<<

```
import random


def generate_expo(mu):
    return random.expovariate(mu)


print(generate_expo(2))
```

>>>

```
    0.03443117946291442
```

Pour réaliser cette simulation, on suppose qu'on a un tableau de S ampoules. Chaque case de ce tableau contient la durée de vie restante d'une ampoule, simulée selon une loi exponentielle. Au départ, toutes les durées de vie sont nulles. On considère qu'à chaque itération, une heure passe. Lors de chaque itération, pour chaque ampoule, on vérifie sa durée de vie restante. Si celle-ci est nulle, on la remplace par une autre dont on choisit aléatoirement la durée de vie (arrondie à l'entier le plus proche). Si la durée de vie n'est pas nulle, on la diminue d'une heure. A chaque itération, on compte le nombre d'ampoules grillées pour en faire la moyenne au bout de n itérations. Pour effectuer cette simulation, les valeurs choisies sont :

$$S = 10000, \mu = \frac{1}{100}, n = 500$$

Le programme suivant réalise cette simulation. On calcule la moyenne du nombre d'ampoules grillées par heure sur les 500 itérations excepté la première pour laquelle toutes les ampoules sont grillées - configuration aberrante ou tout du moins très peu probable -. La valeur obtenue est proche de $S\mu = 100$.

9.3.4 Bibliographie

9.4 Optimisation avec données aléatoires

9.4.1 Un problème simple

Un supermarché pourrait vendre en moyenne 80 poulets par semaine s'il pouvait savoir à l'avance combien de poulets à acheter pour satisfaire la demainde. En réalité, le magasin se réapprovisionne une fois par semaine et lorsque la fin de la semaine arrive, tous les poulets invendus sont soldés et supposés vendus. Le gérant du supermarché voudrait savoir quel est le nombre optimal de poulets à commander chaque semaine. On suppose que le prix d'un poulet à l'achat est p, son prix à la vente est $q > p$, son prix soldé est s. Admettons que le supermarché achète X poulets, en vende au

163. https://docs.python.org/3/library/random.html

mieux N non soldés et $X - N$ soldés s'il en reste. Pour calculer son bénéfice B, il faut tenir compte de deux cas et du fait que le supermarché ne peut pas vendre plus de poulets qu'il n'en a acheté :

$$\begin{array}{ll} B = X(q-p) & \text{si } N \geqslant X \\ B = N(q-p) + (X-N)(s-p) = X(s-p) + N(q-s) & \text{si } N < X \end{array}$$

On peut réduire ces deux expressions à une seule en utilisant la fonction indicatrice :

$$B = f(N, X, p, q, s) = X(q-p)\mathbb{1}_{\{N \geqslant X\}} + [X(s-p) + N(q-s)]\,\mathbb{1}_{\{N<X\}}$$

Si N était connu avec certitude, il suffirait de choisir $X = N$, ce serait la réponse optimale mais le nombre de poulets N vendus est inconnu car il varie chaque semaine. Pour avoir une idée plus précise, le gérant du supermarché a délibérément acheté trop de poulets pendant plusieurs semaines. Il s'est aperçu que la variable aléatoire N suit une loi de Poisson [164] de paramètre $\lambda = 80$. On connaît seulement la probabilité que N soit égale à une valeur fixée. La figure suivante montre l'allure de cette distribution.

$$\mathbb{P}(X = i) = e^{-\lambda}\frac{\lambda^i}{i!}$$

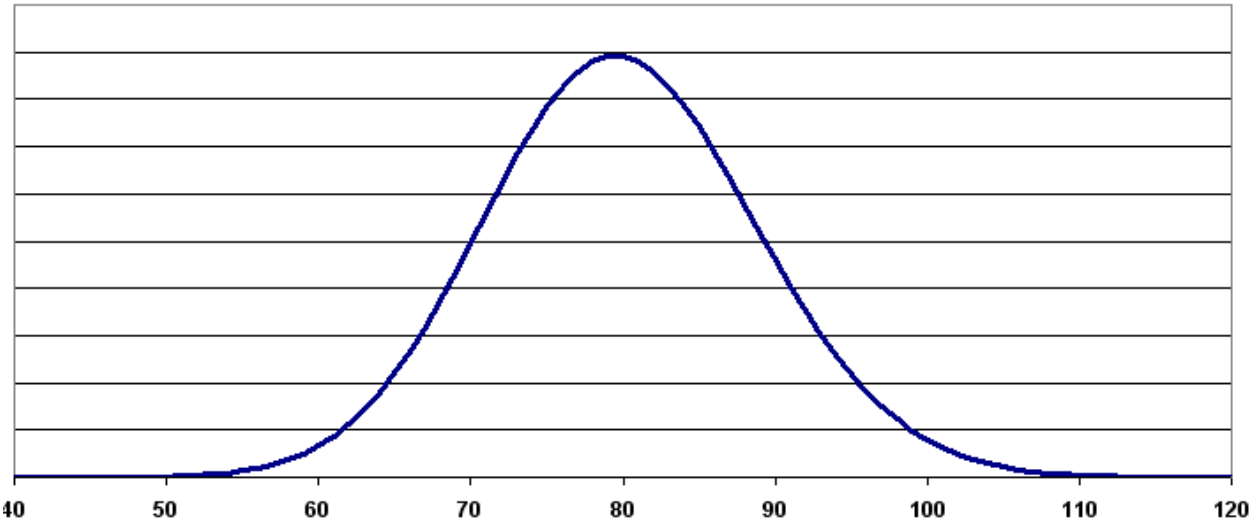


Fig. 1 – Ce graphe répresente la fonction de densité d'une loi de Poisson de paramètre 80. On observe que le pic est obtenu pour une valeur proche de 80, c'est la valeur la plus probable. Ceci signifie que le nombre de poulets achetés le plus probable est 80.

Comme le nombre de poulets achetés varie d'une semaine à l'autre, le bénéfice du supermarché varie aussi d'une semaine à l'autre. Ce que le gérant veut optimiser, c'est la somme de ses profits sur une année ce qui est équivalent à maximiser la moyenne de ses profits chaque semaine. Il faut donc chercher à maximiser l'espérence de la variable aléatoire B à p, q, s constant puis à obtenir la valeur X ayant mené à ce maximum.

$$\max_X \mathbb{E}B = \max_X \mathbb{E}f(N, X, p, q, s) = \max_X \left\{ \sum_{i=0}^{\infty} f(N, X, p, q, s)\mathbb{P}(N = i) \right\}$$

Etant donné la forme de la fonction f, il n'est pas évident de construire une expression exacte de X^* défini par $\max_X \mathbb{E}f(N, X, p, q, s) = f(N, X^*, p, q, s)$. Comme $l = 80$, d'après la figure précédente, on cherche X^* dans l'ensemble $\{0, ..., 2l = 180\}$, aller au delà de 180 est inutile tant la probabilité est faible. Il suffit de calculer f pour

164. https://fr.wikipedia.org/wiki/Loi_de_Poisson

chacune de ces valeurs et de prendre celle qui permet d'obtenir le maximum. Ces calculs longs et répétitifs vont être effectués par un programme informatique qui sera découpé en fonctions comme ceci :

fonction	objectif
`factorielle(x)`	calcule $x!$
`profit(N,X,p,q,s)`	calcule la fonction f
`proba_poisson(l, i)`	calcule la probabilité de Poisson connaissant λ et i
`esperance(X,p,q,s,l)`	calcule l'espérance (sa moyenne) de la fonction f
`maximum(p,q,s,l)`	construit une liste de toutes les valeurs de f
`find_maximum (res)`	cherche le maximum dans la liste retournée par la fonction `maximum`

Le programme obtenu ressemble à `poulet.py`, les dernières lignes servent à tracer la courbe présentée par la figure qui suit.

<<<

```
from mlstatpy.garden.poulet import maximum
res = maximum(2, 5, 1, 80)
# res est la courbe affichée plus bas
print(res[:4])
```

>>>

```
    [(0, 0.0), (1, 2.9999999999999942), (2, 5.9999999999999885), (3, 8.
↪999999999999975)]
```

9.4.2 Modélisation de la demande

La représentation de la demande est essentielle, c'est elle qui détermine le résultat. Il est possible de l'affiner comme par exemple supposer que certaines personnes achètent deux ou trois poulets et que la somme des poulets achetés peut être décomposée comme $N = N_1 + 2N_2 + 3N_3$ où N_i est le nombre de personnes achetant i poulets. Dans ce cas, ce n'est plus N qui suit une loi de Poisson mais N_1, N_2, N_3 qui suivent chacune des lois de Poisson de paramètres différents dont il faudra estimer les paramètres.

Cette modification implique l'écriture d'une fonction `proba_poisson_melange` au lieu de `proba_poisson`. La demande n'est plus une loi connue mais un mélange de lois connues dont la densité n'a pas d'expression connue : il faut la tabuler. Pour cela, on utilise deux propriétés sur les lois exponentielles.

Théorème T1 : simulation d'une loi quelconque

Soit $F = \int f$ une fonction de répartition de densité f vérifiant $f > 0$, soit U une variable aléatoire uniformément distribuée sur $[0, 1]$ alors $F^{-1}(U)$ est variable aléatoire de densité f.

La démonstration est courte. Soit X une variable aléatoire de densité f, par définition, $\mathbb{P}(X \leqslant x) = F(x)$. Soit U une variable aléatoire uniformément distribué sur $[0, 1]$, alors :

$$\begin{aligned} \forall u \in [0,1], \mathbb{P}(U \leqslant u) &= u \\ \iff \mathbb{P}\left(F^{-1}(U) \leqslant F^{-1}(u)\right) &= u \\ \iff \mathbb{P}\left(F^{-1}(U) \leqslant F^{-1}(F(t))\right) &= F(t) \\ \iff \mathbb{P}\left(F^{-1}(U) \leqslant t\right) &= F(t) \end{aligned}$$

Si la fonction F n'est pas strictement croissante, on pourra prendre $F^{-1}(t) = \inf\{u | F(u) \geqslant t\}$. Ce théorème sera appliqué à une loi exponentielle de paramètre λ. La densité d'une telle loi est $f(x) = \lambda \exp -\lambda x$, $F(x) = \int_0^x f(t)dt = 1 - \exp^{-\lambda x}$. On en déduit que $F^{-1}(t) = -\frac{\ln(1-t)}{\lambda}$, par conséquent : $-\frac{\ln(1-U)}{\lambda}$ suit une loi exponentielle de paramètre λ si U est une loi uniforme sur $[0, 1]$.

Théorème T2 : simulation d'une loi de Poisson

On définit une suite infinie $(X_i)_i > 0$ de loi exponentielle de paramètre λ. On définit ensuite la série de variables aléatoires $S_i = \sum_{k=1}^{i} X_k$ et enfin $N(t) = \inf\{i | S_i > t\}$. Alors la variable aléatoire $N(t)$ suit une loi de Poisson de paramètre λt.

La loi exponentielle est souvent utilisée pour modéliser le temps d'attente d'un événement comme le temps d'attente d'un métro une fois sur le quai. On l'utilise aussi pour modéliser la durée de vie d'un outil, d'une ampoule par exemple. La loi de Poisson peut par exemple modéliser le nombre d'ampoules nécessaire pour éclairer une pièce sur une certaine durée. Avant de démontrer le théorème, il faut définir d'abord la loi Gamma [165]. On pose au préalable $\Gamma(\alpha) = \int_0^\infty u^{\alpha-1}e^{-u}du$. Une variable aléatoire de loi Gamma de paramètres (α, λ) a pour densité : $f(x) = \frac{\lambda^\alpha}{\Gamma(\alpha)}t^{\alpha-1}e^{-\lambda t}$. La fonction Γ vérifie une propriété utile par la suite : $\forall n \in \mathbb{N}^*$, $\Gamma(n) = (n-1)!$.

Théorème T3 : somme de loi exponentielle iid

Soit $X_1, ..., X_n$ n variables aléatoires indépendantes et identiquement distribuées de loi $Exp(\lambda)$ alors la somme $\sum_{k=1}^{n} X_k$ suit une loi $Gamma(n, \lambda)$.

La démonstration utilise l'unicité de la fonction caractéristique $\mathbb{E}e^{iX}$. Il suffit de démonstrer que la fonction caractéristique de la somme est celle d'une loi Gamma. On suppose que $X_1, ..., X_n$ suivent des lois exponentielles de paramètre λ et Y suit une loi $Gamma(n, \lambda)$.

$$\begin{aligned}
\mathbb{E}\exp\left(i\sum_{k=1}^{n} X_k\right) &= \prod_{k=1}^{n} \mathbb{E}e^{iX_k} \\
&= \left[\int_0^\infty \lambda e^{ix}e^{-\lambda x}dx\right]^n = \lambda^n\left[\int_0^\infty e^{(i-\lambda)x}dx\right]^n \\
&= \lambda^n\left[-\frac{1}{(i-\lambda)}\right]^n = \left[\frac{\lambda}{\lambda - i}\right]^n \\
\mathbb{E}e^{iY} &= \int_0^\infty \frac{\lambda^n}{\Gamma(n)}t^{n-1}e^{-\lambda t}e^{it}dt = \int_0^\infty \frac{\lambda^n}{\Gamma(n)}t^{n-1}e^{(i-\lambda)t}dt \\
&= \frac{\lambda^n}{\Gamma(n)}\frac{\Gamma(n)}{(i-\lambda)^n} = \left[\frac{\lambda}{\lambda - i}\right]^n
\end{aligned}$$

Ces lignes démontrent le théorème. On démontre maintenant *simulation d'une loi de Poisson* (page 219). La démonstration repose sur le fait que $\mathbb{P}(N(t) \geqslant n) \Longleftrightarrow \mathbb{P}(S_n \leqslant t)$. On en déduit que :

$$\mathbb{P}(N(t) = n) = \mathbb{P}(N(t) \geqslant n) - \mathbb{P}(N(t) \geqslant n+1) = \mathbb{P}(S_n \leqslant t) - \mathbb{P}(S_{n+1} \leqslant t)$$

Or d'après le théorème *somme de loi exponentielle iid* (page 219), S_n suit une loi $Gamma(n, \lambda)$.

$$\begin{aligned}
\mathbb{P}(N(t) = n) &= \int_0^t \frac{\lambda^n}{\Gamma(n)}u^{n-1}e^{-\lambda u}du - \int_0^t \frac{\lambda^{n+1}}{\Gamma(n+1)}u^n e^{-\lambda u}du \\
&= \int_0^t \left[\frac{\lambda^n}{(n-1)!}u^{n-1}e^{-\lambda u} - \frac{\lambda^{n+1}}{n!}u^n e^{-\lambda u}\right]du \\
&= \left[\frac{\lambda^n}{n!}u^n e^{-\lambda u}\right]_0^t = e^{-\lambda t}\frac{(\lambda t)^n}{n!}
\end{aligned}$$

Il suffit d'utiliser ce théorème pour simuler une loi de Poisson de paramètre λ, ce que fait la fonction `poisson` suivante :

<<<

165. https://fr.wikipedia.org/wiki/Loi_Gamma

```
import random
import math


def exponentielle(l):
    u = random.random()
    return -1.0 / l * math.log(1.0 - u)


def poisson(l):
    s = 0
    i = 0
    while s <= 1:
        s += exponentielle(l)
        i += 1
    return i-1


print(poisson(2))
```

>>>

```
    1
```

On vérifie que cette méthode de simulation permet de retrouver les résultats théoriques. Pour cela, on effectue 1000 tirages d'une variable suivant une loi de Poisson avec $\lambda = 10$ puis on compte le nombre de fois qu'on obtient chaque entier compris entre 0 et 40. La figure qui suit permet de comparer les résultats obtenus.

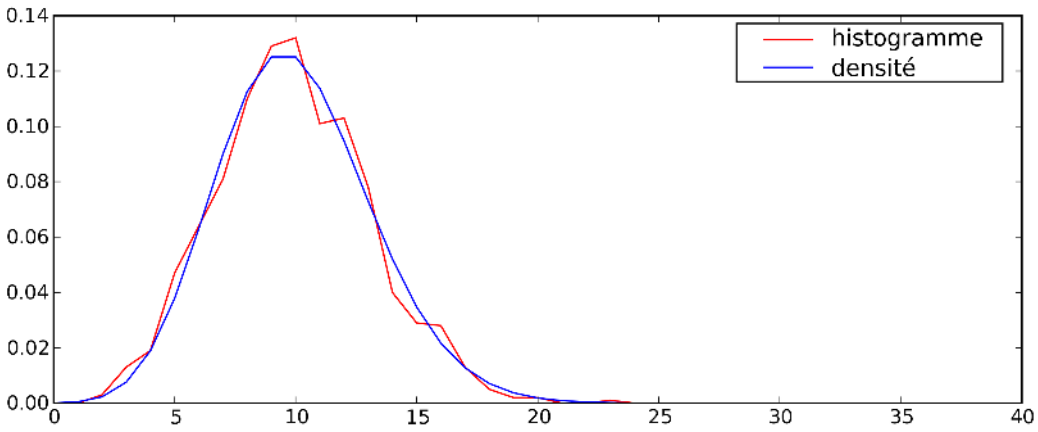


Fig. 2 – Comparaison entre une fonction de densité estimée empiriquement pour la loi de Poisson de paramètre $\lambda = 10$ et sa densité théorique $f(i) = e^{-\lambda}\frac{\lambda^i}{i!}$.

On cherche maintenant à calculer les probabilités $\mathbb{P}(N = i)$ sachant que $N = N_1 + 2N_2 + 3N_3$ et $N_1 \sim \mathcal{P}(48)$, $N_2 \sim \mathcal{P}(10)$, $N_3 \sim \mathcal{P}(4)$. L'addition de deux lois de Poisson indépendantes est une loi de Poisson. En revanche, si N_1 suit une loi de Poisson, $2N_1$ ne suit pas une loi de Poisson. $2N_1$ est une variable paire, c'est une propriété qui n'est jamais vérifiée par une loi de Poisson. Il n'existe pas d'expression évidente pour la densité du mélange N, il faut donc simuler cette variable. C'est l'objectif de la fonction `poisson_melange`. De la même manière, on estime l'histogramme du mélange avec cette fois-ci un plus grand nombre de tirages (10000) pour aboutir à la figure suivante.

On utilise ces éléments pour modéliser la demande de poulets selon ce mélange de lois Poisson. Le premier programme est modifié pour aboutir au suivant.

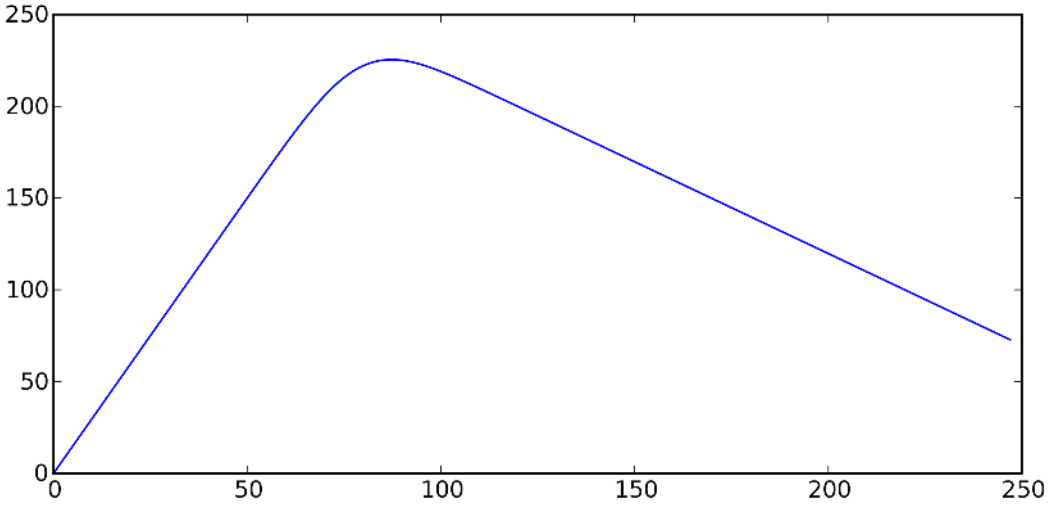


Fig. 3 – Dans le cas du mélange de lois Poisson, le maximum est cette-fois ci obtenu pour 87 poulets et est de 225 euros. Ces résultats sont légèrement différents de ceux obtenus par une simple loi Poisson (80).

9.4.3 Variations saisonnières et prolongations

Les paragraphes précédents supposent que la demande est constante et ne dépend pas des saisons. Cette affirmation est peut-être vraie en ce concerne les poulets mais ce n'est certainement pas le cas des huîtres qui sont traditionnellement consommées en décembre. Appliqué à l'exemple des poulets décrits dans cet énoncé, la loi de Poisson appliquée à la consommation dépend maintenant de la semaine.

Tenir compte de la saisonnalité n'est pas forcément un problème de modélisation mais plutôt d'estimation. Au lieu d'avoir une seule consommation moyenne, il y a en aura maintenant 52. Ceci implique d'avoir des données en nombre suffisant pour estimer les paramètres du modèle : la précision des résultats dépend de celle de l'estimation. Il est possible d'estimer séparément les variations saisonnières et la demande elle-même mais la présentation de ces techniques dépassent le cadre de ce livre, il est préférable de se reporter à *[Gouriéroux1983]* (page **??**) ou *[Saporta2006]* (page **??**).

Les poulets soldés ne sont pas plus mauvais que les poulets non soldés bien que la date de péremption soit certainement plus rapprochée de la date d'achat. On suppose qu'un gérant concurrent de ce supermarché a eu vent de la technique d'optimisation du magasin, il connaît également le prix du poulet et son prix soldé. Il a également accès au prix d'achat puisqu'il se fournit chez les mêmes agriculteurs. Il lui reste à connaître le nombre de poulets commandés et une estimation de la demande pour savoir si les poulets de son concurrents se vendent mieux que les siens. Il se rend dans le supermarché concurrent tous les jours où les poulets sont soldés et les comptent. Il voudrait pouvoir en déduire le nombre de poulets vendus.

9.4.4 Bibliographie

9.5 Régression linéaire et résultats numériques

Ce notebook s'intéresse à la façon d'interpréter les résultats d'une régression linéaire lorsque les variables sont corrélées puis il explore une façon d'associer arbre de décision et régression linéaire pour construire une régression linéaire par morceaux.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

```
%matplotlib inline
```

9.5.1 Un cas simple

Une façon d'interpréter des résultats statistiques est de les calculer dans un cas où la réponse cherchée est connue. On simule un modèle simple $Y = \alpha X_1 + 0.X_2 + \epsilon$ et on cale une régression linéaire. On suppose que X_1, X_2, ϵ sont des variables aléatoires gaussiennes de même variance et moyenne.

```
import numpy.random as npr
eps = npr.normal(1000)
X = npr.normal(size=(1000, 3))
alpha = 2
Y = alpha * X[:,0] + X[:, 2]
X.shape, Y.shape
```

```
((1000, 3), (1000,))
```

```
from numpy import corrcoef
corrcoef(X.T)
```

```
array([[ 1.        , -0.0312982 ,  0.05188551],
       [-0.0312982 ,  1.        , -0.00356494],
       [ 0.05188551, -0.00356494,  1.        ]])
```

```
from statsmodels.regression.linear_model import OLS
```

```
model = OLS(Y,X[:, :2])
results = model.fit()
su = results.summary()
su
```

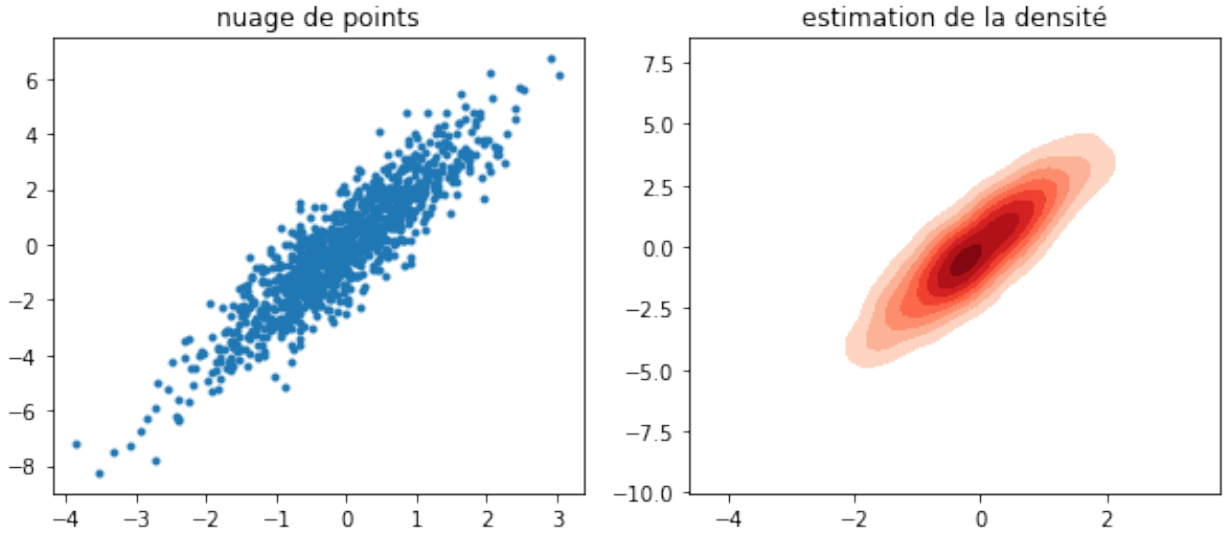
```
results.rsquared, results.rsquared_adj
```

```
(0.8153831029946165, 0.8150131292531227)
```

On vérifie que le coefficient devant X_1 est non nul (P-value nulle, 0 n'est pas l'intervalle de confiance). Le coefficient devant X_2 n'est pas nul mais presque, la P-value est élevée, le coefficient R^2 est élevé. Dessinons.

```
import matplotlib.pyplot as plt
import seaborn
fig, ax = plt.subplots(1, 2, figsize=(10,4))
ax[0].plot(X[:, 0], Y, '.')
seaborn.kdeplot(X[:, 0], Y, cmap="Reds", shade=True, shade_lowest=False, ax=ax[1])
ax[0].set_title("nuage de points")
ax[1].set_title("estimation de la densité");
```

```
c:python370_x64libsite-packagesscipystatsstats.py:1713: FutureWarning: Using␣
↪a non-tuple sequence for multidimensional indexing is deprecated; use␣
↪arr[tuple(seq)] instead of arr[seq]. In the future this will be interpreted␣
↪as an array index, arr[np.array(seq)], which will result either in an error␣
↪or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

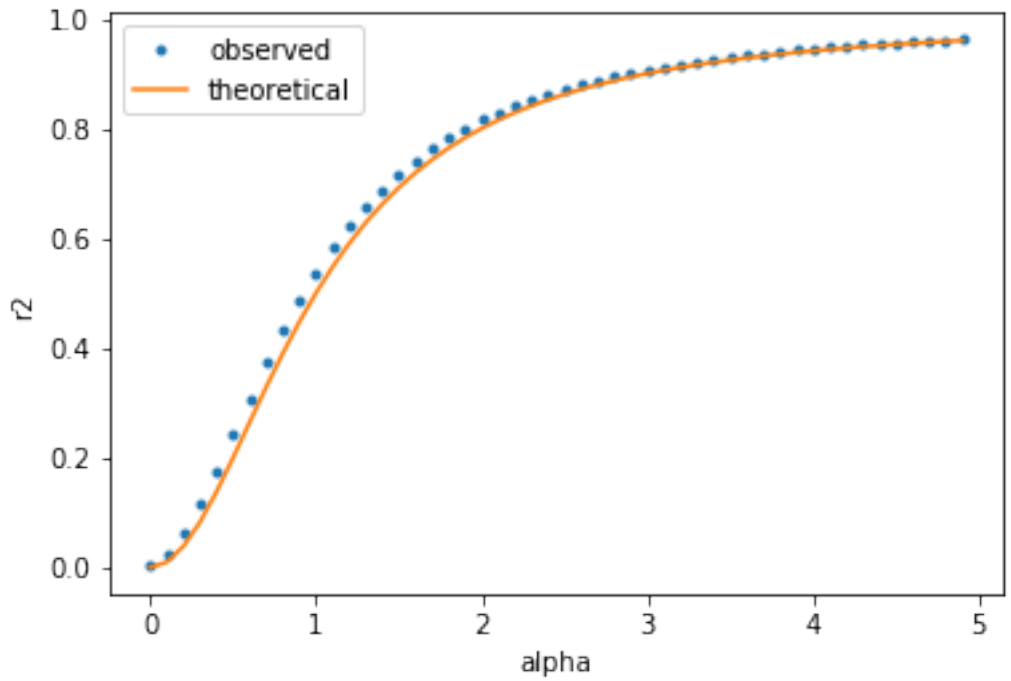


9.5.2 Evolution de R2

Dans la régression précédente, le coefficient R^2 transcrit en quelque sorte la part du bruit ϵ par rapport au terme αX_1. Faisons varier α.

```
alphas = []
r2s = []
for a in [0.1 * i for i in range(0, 50)]:
    Y = a*X[:,0] + X[:, 2]
    model = OLS(Y,X[:, :2])
    results = model.fit()
    alphas.append(a)
    r2s.append(results.rsquared)
```

```
fig, ax = plt.subplots(1, 1)
ax.plot(alphas, r2s, '.', label="observed")
ax.plot(alphas, [a**2/(1+a**2) for a in alphas], label='theoretical')
ax.set_xlabel("alpha")
ax.set_ylabel("r2")
ax.legend();
```



Dans ce cas de régression simple, la valeur à prédire est y_i, la valeur prédite est $\hat{y}_i = \alpha X_{1i}$ et la moyenne $\bar{y} = \alpha\bar{X}_1 + \bar{\epsilon} = 0$.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{\mathbb{V}\epsilon}{\alpha^2\mathbb{V}X_1 + \mathbb{V}\epsilon} = 1 - \frac{1}{1+\alpha^2} = \frac{\alpha^2}{1+\alpha^2}$$

9.5.3 Deux variables corrélées

On ne change pas le modèle mais on fait en sorte que $X_2 = X_1$. Les deux variables sont corrélées.

```
X[:, 1] = X[:, 0]
Y = 2*X[:,0] + X[:, 2]
model = OLS(Y,X[:, :2])
results = model.fit()
results.summary()
```

```
model.rank
```

```
1
```

Les variables corrélées n'ont pas l'air de déranger l'algorithme de résolution car il utilise la méthode SVD [166] pour résoudre le même problème dans un espace de moindre dimension. Le problème survient que les deux variables ne sont pas complétement corrélées. On étudie le modèle $Y \sim X_1 + X_2'$ avec $X_2' = \alpha X_1 + (1-\alpha)X_2$ et on réduit la variance du bruit pour en diminuer les effets.

```
X_ = npr.normal(size=(1000, 3))
```

```
alphas = [0.9 + i * 0.01 for i in range(0,11)]
res = []
```

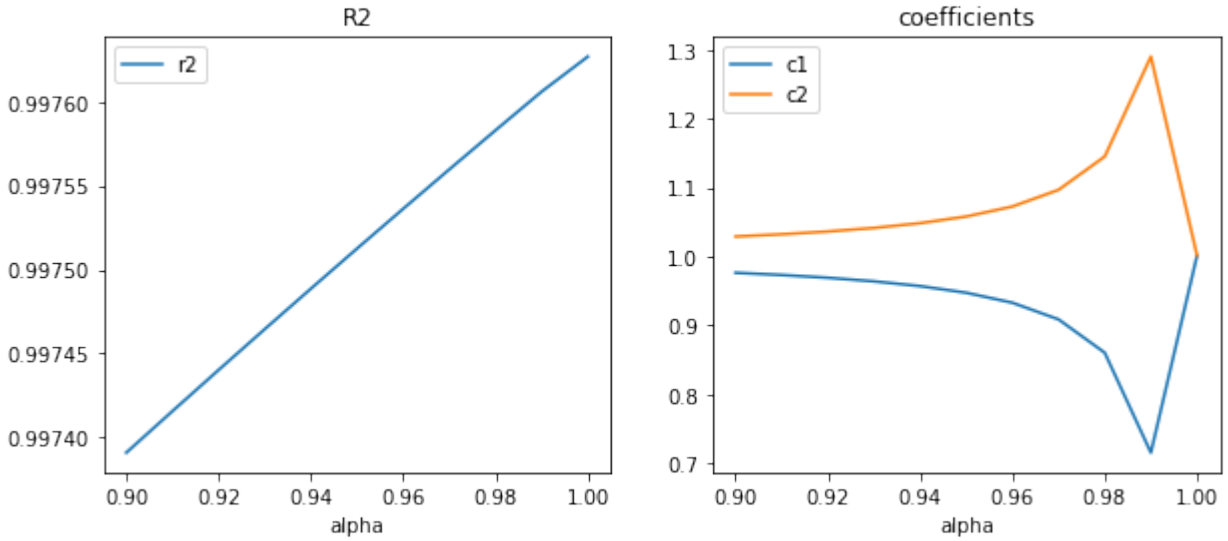
(suite sur la page suivante)

166. https://en.wikipedia.org/wiki/Singular-value_decomposition

(suite de la page précédente)

```
for a in alphas:
    X = X_.copy()
    X[:, 1] = a * X[:, 0] + (1-a) * X[:, 1]
    Y = X[:, 0] + X[:, 1] + 0.1 * X[:, 2]
    model = OLS(Y,X[:, :2])
    results = model.fit()
    res.append(dict(alpha=a, r2=results.rsquared, rank=model.rank, c1=results.
↪params[0], c2=results.params[1]))

import pandas
df = pandas.DataFrame(res)
df = df.set_index('alpha')
df
```

```
fig, ax = plt.subplots(1,2, figsize=(10,4))
df[["r2"]].plot(ax=ax[0])
df[["c1", "c2"]].plot(ax=ax[1])
ax[0].set_title("R2")
ax[1].set_title("coefficients");
```



Le r^2 augmente quand la corrélation augmente mais les coefficients sont moins fiables. Les résultats devraient être sensiblement identiques en théorie mais en pratique, plus le déterminant devient proche de zéro, plus l'ordinateur est limité par sa précision numérique. Pour en savoir plus, vous pouvez lire un examen écrit que j'ai rédigé, en python bien sûr : Examen Programmation ENSAE première année 2006 [167]. Cette précision est aux alentours de 10^{-15} ce qui correspond à la précision numérique des double [168].

```
alphas = [1 - 10**(-i) for i in range(10,18)]
res = []
for a in alphas:
    X = X_.copy()
    X[:, 1] = a * X[:, 0] + (1-a) * X[:, 1]
    Y = X[:, 0] + X[:, 1] + X[:, 2]
    model = OLS(Y,X[:, :2])
    results = model.fit()
```

(suite sur la page suivante)

167. http://www.xavierdupre.fr/site2013/enseignements/tdnote/ecrit_2006.pdf
168. https://en.wikipedia.org/wiki/Double-precision_floating-point_format

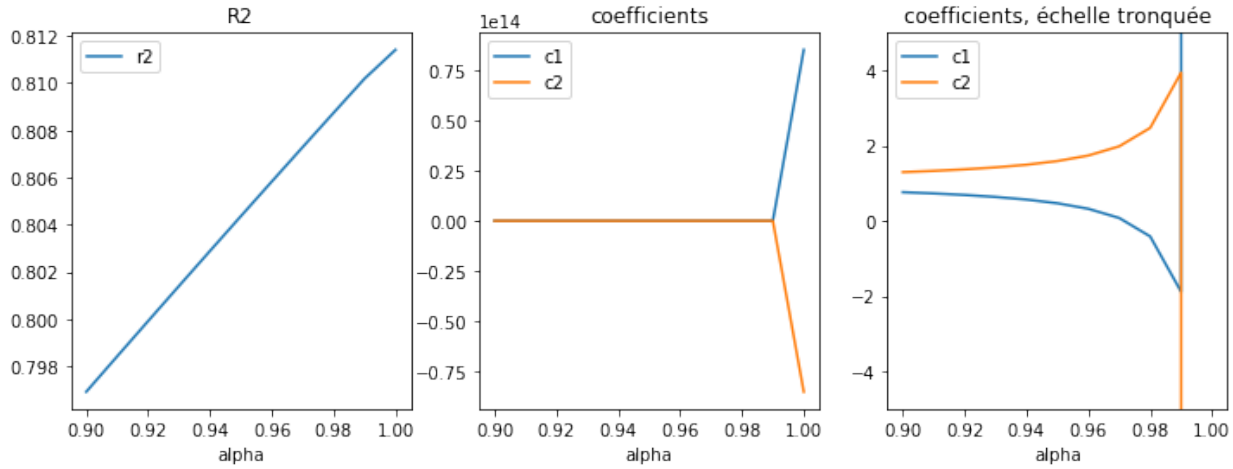
(suite de la page précédente)

```
    res.append(dict(alpha_1=a-1, r2=results.rsquared, rank=model.rank, c1=results.
↪params[0], c2=results.params[1]))

import pandas
df = pandas.DataFrame(res)
df = df.set_index('alpha_1')
df
```

On fait un dernier test avec scikit-learn [169] pour vérifier que l'algorithme de résolution donne des résultats similaires pour un cas où le déterminant est quasi-nul.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

alphas = [0.9 + i * 0.01 for i in range(0,11)]
res = []
for a in alphas:
    X = X_.copy()
    X[:, 1] = a * X[:, 0] + (1-a) * X[:, 1]
    Y = X[:, 0] + X[:, 1] + X[:, 2]
    model = LinearRegression()
    model.fit(X[:, :2], Y)
    r2 = r2_score(Y, model.predict(X[:, :2]))
    res.append(dict(alpha=a, c1=model.coef_[0], c2=model.coef_[1], r2=r2))

import pandas
df = pandas.DataFrame(res)
df = df.set_index('alpha')
df
```

```
fig, ax = plt.subplots(1,3, figsize=(12,4))
df[["c1", "c2"]].plot(ax=ax[1])
df[["c1", "c2"]].plot(ax=ax[2])
df[["r2"]].plot(ax=ax[0])
ax[0].set_title("R2")
ax[1].set_title("coefficients")
ax[2].set_ylim([-5, 5])
ax[2].set_title("coefficients, échelle tronquée");
```



169. http://scikit-learn.org/stable/

Le second graphe est trompeur mais il ne faut pas oublier de regarder l'échelle de l'axe des ordonnées.

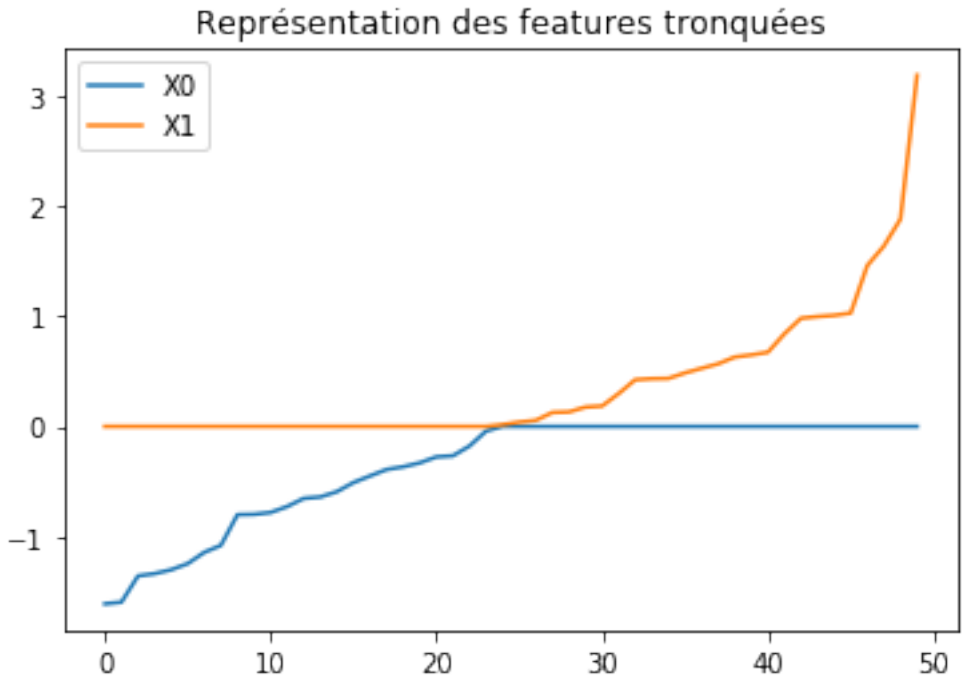
9.5.4 Indicatrices

X_1 est une variable aléatoire gaussienne. On teste maintenant un modèle $Y = X'_1 + X'_2 + \epsilon$ avec $X'_1 = X_1 \mathbb{1}_{X_1<0}$ et $X'_2 = X_1 \mathbb{1}_{X_1 \geqslant 0}$.

```
X = npr.normal(size=(1000, 3))
X[:, 1] = X[:, 0]
X[X[:, 0] >= 0, 0] = 0
X[X[:, 1] < 0, 1] = 0
Y = X[:, 0] + X[:, 1] + X[:, 2]
corrcoef(X.T)
```

```
array([[ 1.        ,  0.47358312, -0.03083914],
       [ 0.47358312,  1.        , -0.01293737],
       [-0.03083914, -0.01293737,  1.        ]])
```

```
from pandas import DataFrame
names = ["X%d" % i for i in range(X.shape[1]-1)]
ax = DataFrame(X[:50,:2], columns=names).sort_values(names).reset_index(drop=True).
↪plot()
ax.set_title("Représentation des features tronquées");
```



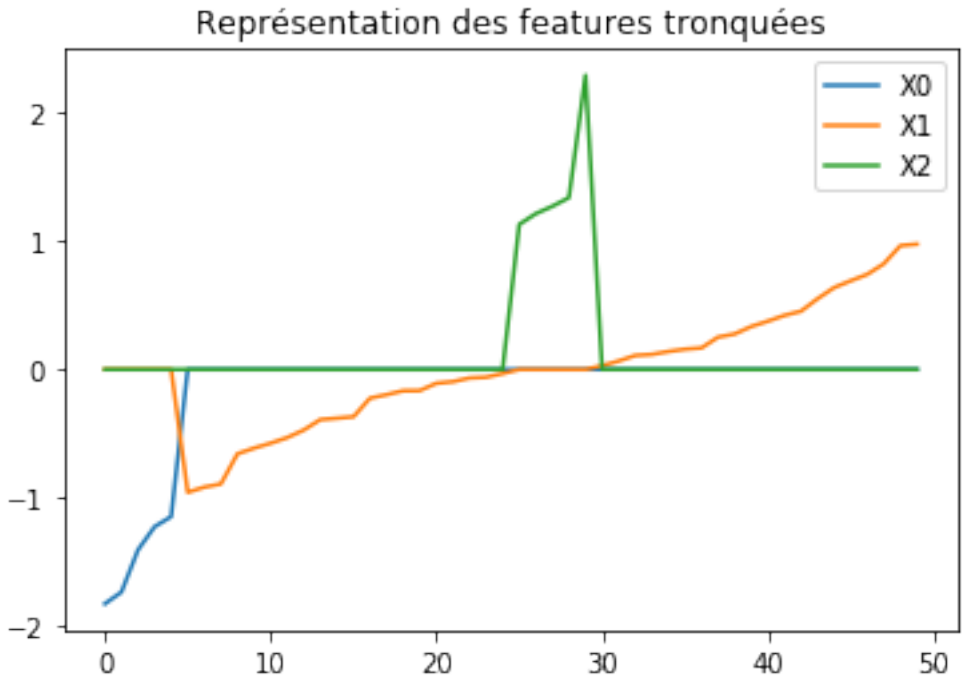

```
model = OLS(Y,X[:, :3])
results = model.fit()
results.summary()
```

On découpe en trois.

```
import numpy
X = npr.normal(size=(1000, 4))
for i in range(0, 3):
    X[:, i] = X_[:, 0]
X[:, 3] = X_[:, 2]
X[X_[:, 0] > -1, 0] = 0
X[(X_[:, 0] < -1) | (X_[:, 0] > 1), 1] = 0
X[X_[:, 0] < 1, 2] = 0
Y = X[:, 0] + X[:, 1] + X[:, 2] + X[:, 3]
corrcoef(X.T)
```

```
array([[ 1.        , -0.00347584,  0.16846101,  0.06722762],
       [-0.00347584,  1.        ,  0.00326437, -0.04707208],
       [ 0.16846101,  0.00326437,  1.        ,  0.08754832],
       [ 0.06722762, -0.04707208,  0.08754832,  1.        ]])
```

```
from pandas import DataFrame
names = ["X%d" % i for i in range(X.shape[1]-1)]
ax = DataFrame(X[:50,:3], columns=names).sort_values(names).reset_index(drop=True).
↪plot()
ax.set_title("Représentation des features tronquées");
```




```
model = OLS(Y,X[:, :4])
results = model.fit()
results.summary()
```

9.5.5 Régression linéaire par morceaux

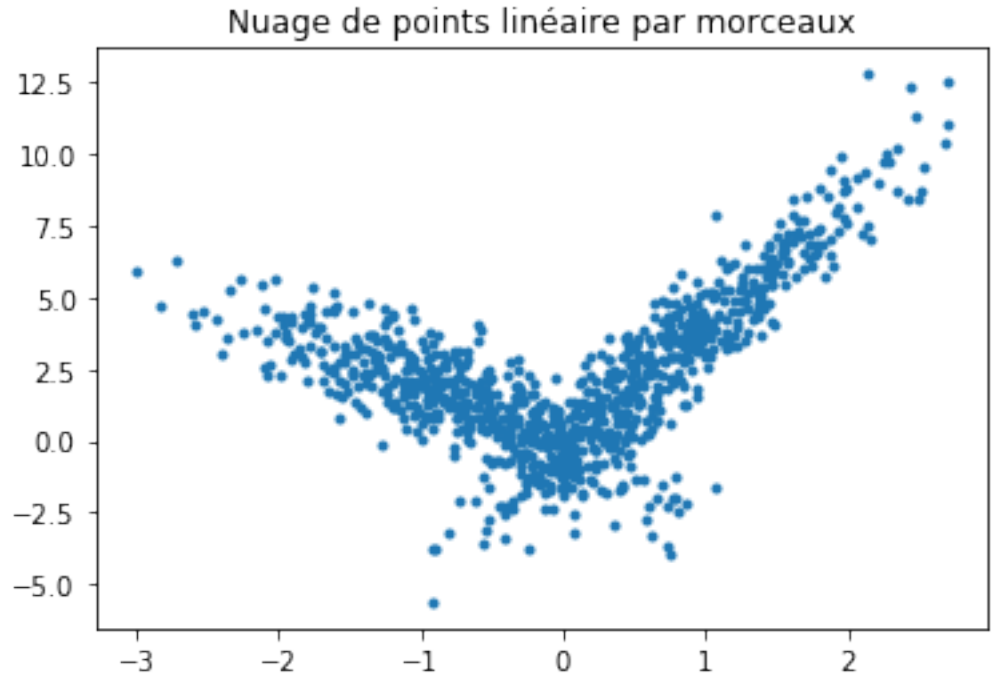
On se place dans un cas particulier où le problème est linéaire par morceaux :

$$Y = -2X_1 \mathbb{1}_{X_1+\epsilon_1<0} + 4X_1 \mathbb{1}_{X+\epsilon_1>0} + \epsilon_2$$

La régression donne de très mauvais résultat sur ce type de problèmes mais on cherche une façon systématique de découper le problème en segments linéaires.

```
X = npr.normal(size=(1000,4))
alpha = [4, -2]
t = (X[:, 0] + X[:, 3] * 0.5) > 0
switch = numpy.zeros(X.shape[0])
switch[t] = 1
Y = alpha[0] * X[:, 0] * t + alpha[1] * X[:, 0] * (1-t) + X[:, 2]
```

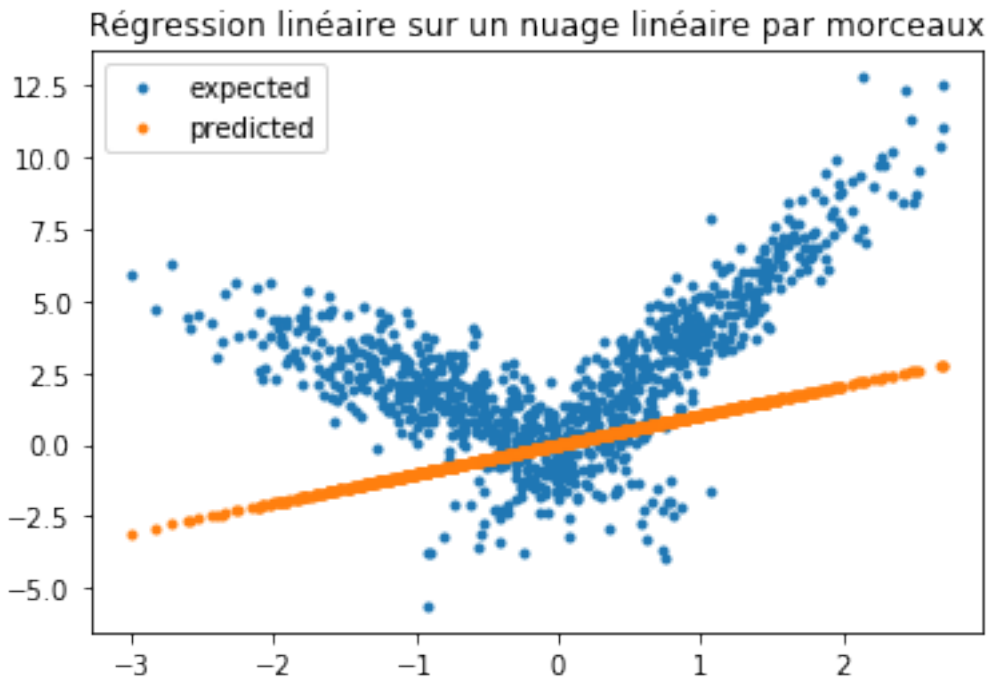
```
fig, ax = plt.subplots(1, 1)
ax.plot(X[:, 0], Y, ".")
ax.set_title("Nuage de points linéaire par morceaux");
```




```
model = OLS(Y,X[:, :1])
results = model.fit()
results.summary()
```

```
yp = results.predict(X[:, :1])
```

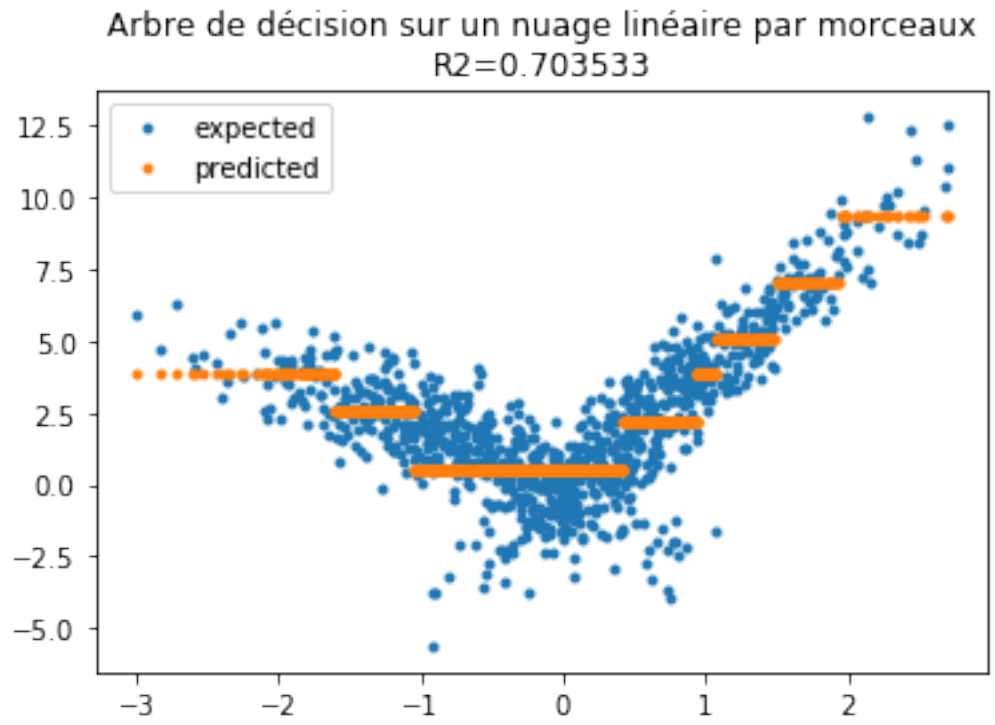
```
fig, ax = plt.subplots(1, 1)
ax.plot(X[:, 0], Y, ".", label="expected")
ax.plot(X[:, 0], yp, ".", label="predicted")
ax.legend()
ax.set_title("Régression linéaire sur un nuage linéaire par morceaux");
```



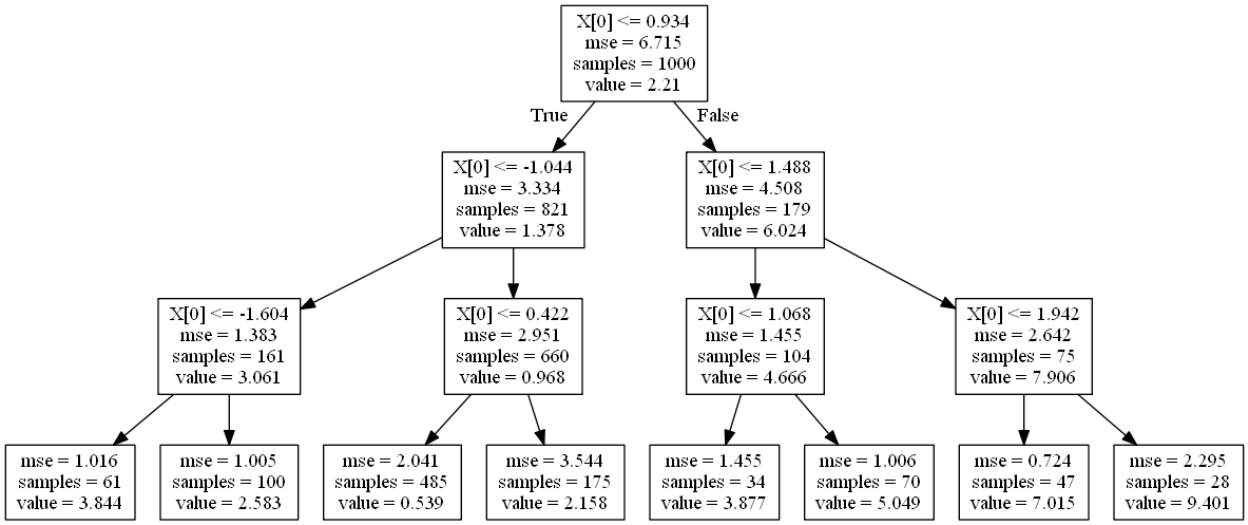
Passons à un arbre de décision qui n'est pas le meilleur modèle mais on va détourner ses résultats pour revenir à un problème de régression par morceaux.

```
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(min_samples_leaf=10, max_depth=3)
model.fit(X[:, :1], Y)
yp = model.predict(X[:, :1])
```

```
fig, ax = plt.subplots(1, 1)
ax.plot(X[:, 0], Y, ".", label="expected")
ax.plot(X[:, 0], yp, ".", label="predicted")
ax.legend()
r2 = r2_score(Y, model.predict(X[:, :1]))
ax.set_title("Arbre de décision sur un nuage linéaire par morceaux\nR2=%f" % r2);
```




```
from sklearn.tree import export_graphviz
export_graphviz(model, out_file="arbre.dot")
from pyensae.graphhelper import run_dot
run_dot("arbre.dot", "arbre.png")
from IPython.display import Image
Image("arbre.png")
```



On extrait tous les seuils de l'arbre et on ajoute les milieux de segments.

```
th = list(sorted(set(model.tree_.threshold)))
th += [(th[i] + th[i-1])/2 for i in range(1,len(th))]
th = list(sorted(th))
th
```

```
[-2.0,
 -1.8018612563610077,
 -1.6037225127220154,
 -1.323736995458603,
 -1.0437514781951904,
 -0.3109976723790169,
 0.4217561334371567,
 0.678125374019146,
 0.9344946146011353,
 1.0011553764343262,
 1.067816138267517,
 1.2776717841625214,
 1.4875274300575256,
 1.7147845923900604,
 1.9420417547225952]
```

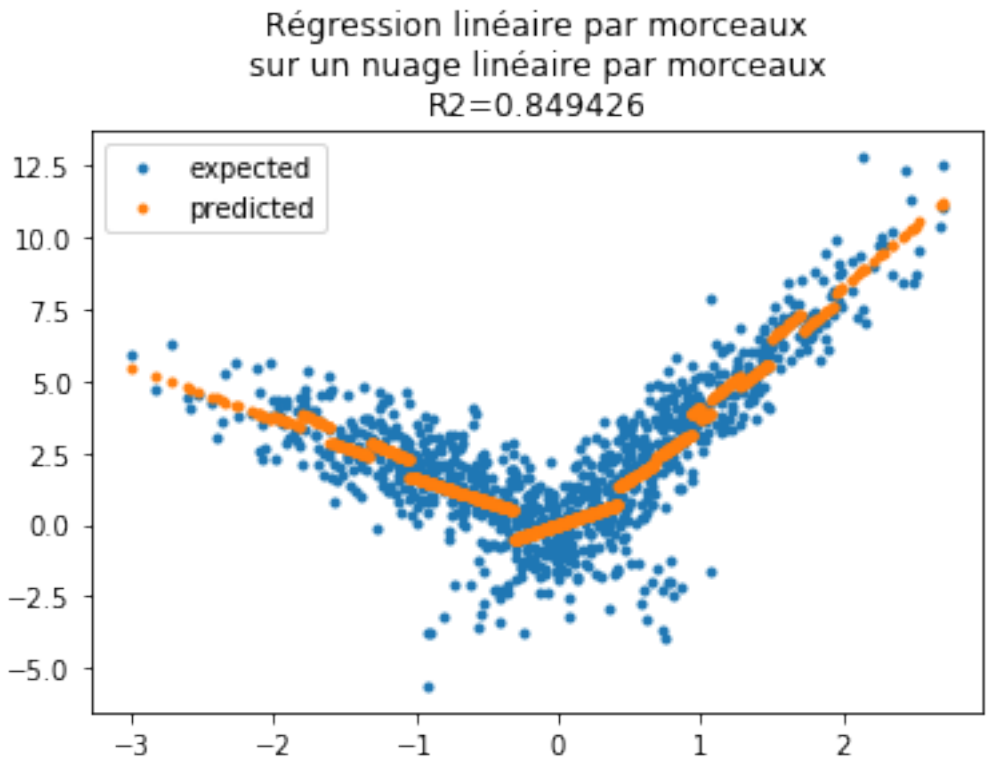
On fait une régression sur les variables $W_{i>0} = X_1 \mathbb{1}_{X_1>t_i}$, $W_0 = X_1$ où les (t_i) sont les seuils.

```
W = numpy.zeros((X.shape[0], len(th)+1))
x = X[:, 0]
W[:, 0] = x
for i in range(len(th)):
    W[x > th[i], i+1] = x[x > th[i]]
```

```
model = OLS(Y,W)
results = model.fit()
results.summary()
```

Dessinons les résultats de la prédictions.

```
yp = results.predict(W)
fig, ax = plt.subplots(1, 1)
ax.plot(X[:, 0], Y, ".", label="expected")
ax.plot(X[:, 0], yp, ".", label="predicted")
ax.legend()
ax.set_title("Régression linéaire par morceaux\nsur un nuage linéaire par␣
↪morceaux\nR2=%f" % results.rsquared);
```



Le modèle nous suggère de ne garder que quelques seuils. En s'appuyant sur les p-values :

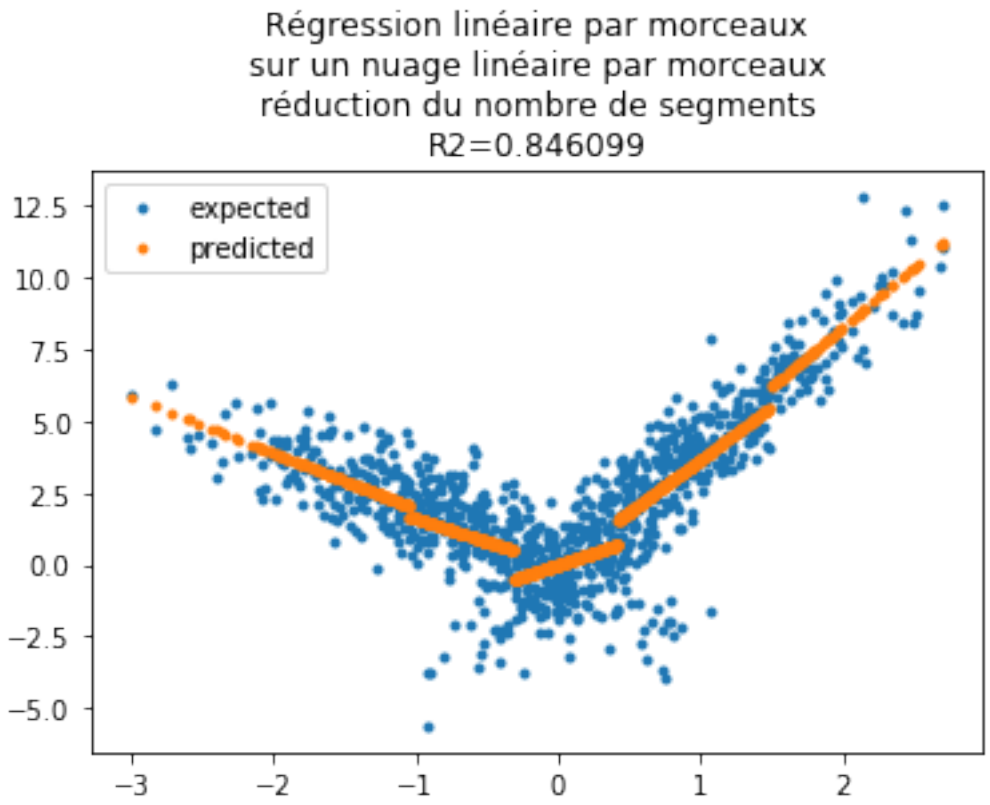
```
keep = numpy.arange(len(results.pvalues))[results.pvalues < 0.05]
keep
```

```
array([ 0,  5,  6,  7, 13])
```

```
W2 = W[:, keep]
```

```
model = OLS(Y,W2)
results = model.fit()
results.summary()
```

```
yp = results.predict(W2)
fig, ax = plt.subplots(1, 1)
ax.plot(X[:, 0], Y, ".", label="expected")
ax.plot(X[:, 0], yp, ".", label="predicted")
ax.legend()
ax.set_title("Régression linéaire par morceaux\nsur un nuage linéaire par morceaux\n"␣
↪+
             "réduction du nombre de segments\nR2=%f" % results.rsquared);
```



Le coefficient R^2 est quasiment identique pour un nombre de segments moindre.

9.6 Le gradient et le discret

Les méthodes d'optimisation à base de gradient s'appuie sur une fonction d'erreur dérivable qu'on devrait appliquer de préférence sur des variables aléatoires réelles. Ce notebook explore quelques idées.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```


9.6.1 Un petit problème simple

On utilise le jeu de données *iris* disponible dans scikit-learn [170].

170. http://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

```
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
Y = iris.target
```

On cale une régression logistique. On ne distingue pas apprentissage et test car ce n'est pas le propos de ce notebook.

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(multi_class="ovr", solver="liblinear")
clf.fit(X, Y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

Puis on calcule la matrice de confusion.

```
from sklearn.metrics import confusion_matrix
pred = clf.predict(X)
confusion_matrix(Y, pred)
```

```
array([[49,  1,  0],
       [ 2, 21, 27],
       [ 1,  4, 45]], dtype=int64)
```

9.6.2 Multiplication des observations

Le paramètre `multi_class='ovr'` stipule que le modèle cache en fait l'estimation de 3 régressions logistiques binaire. Essayons de n'en faire qu'une seule en ajouter le label `Y` aux variables. Soit un couple $(X_i \in \mathbb{R}, Y_i \in \mathbb{N})$ qui correspond à une observation pour un problème multi-classe. Comme il y a C classes, on multiplie cette ligne par le nombre de classes C pour obtenir :

$$\forall c \in [1, ..., C], \left\{ \begin{array}{l} X'_i = (X_{i,1}, ..., X_{i,d}, Y_{i,1}, ..., Y_{i,C}) \\ Y'_i = \mathbb{1}_{Y_i=c} \\ Y_{i,k} = \mathbb{1}_{c=k} \end{array} \right.$$

Voyons ce que cela donne sur un exemple :

```
import numpy
import pandas

def multiplie(X, Y, classes=None):
    if classes is None:
        classes = numpy.unique(Y)
    XS = []
    YS = []
    for i in classes:
        X2 = numpy.zeros((X.shape[0], 3))
        X2[:,i] = 1
        Yb = Y == i
        XS.append(numpy.hstack([X, X2]))
        Yb = Yb.reshape((len(Yb), 1))
        YS.append(Yb)
```

(suite sur la page suivante)

(suite de la page précédente)

```
    Xext = numpy.vstack(XS)
    Yext = numpy.vstack(YS)
    return Xext, Yext

x, y = multiplie(X[:1,:], Y[:1], [0, 1, 2])
df = pandas.DataFrame(numpy.hstack([x, y]))
df.columns = ["X1", "X2", "Y0", "Y1", "Y2", "Y'"]
df
```

Trois colonnes ont été ajoutées côté X, la ligne a été multipliée 3 fois, la dernière colonne est Y qui ne vaut 1 que lorsque le 1 est au bon endroit dans une des colonnes ajoutées. Le problème de classification qui été de prédire la bonne classe devient : est-ce la classe à prédire est k ? On applique cela sur toutes les lignes de la base et cela donne :

```
Xext, Yext = multiplie(X, Y)
numpy.hstack([Xext, Yext])
df = pandas.DataFrame(numpy.hstack([Xext, Yext]))
df.columns = ["X1", "X2", "Y0", "Y1", "Y2", "Y'"]
df.iloc[numpy.random.permutation(df.index), :].head(n=10)
```

```
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier()
clf.fit(Xext, Yext.ravel())
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

```
pred = clf.predict(Xext)
confusion_matrix(Yext, pred)
```

```
array([[278,  22],
       [ 25, 125]], dtype=int64)
```

9.6.3 Introduire du bruit

Un des problèmes de cette méthode est qu'on ajoute une variable binaire pour un problème résolu à l'aide d'une optimisation à base de gradient. C'est moyen. Pas de problème, changeons un peu la donne.

```
def multiplie_bruit(X, Y, classes=None):
    if classes is None:
        classes = numpy.unique(Y)
    XS = []
    YS = []
    for i in classes:
        # X2 = numpy.random.randn((X.shape[0]* 3)).reshape(X.shape[0], 3) * 0.1
        X2 = numpy.random.random((X.shape[0], 3)) * 0.2
```

(suite sur la page suivante)

(suite de la page précédente)

```
        X2[:,i] += 1
        Yb = Y == i
        XS.append(numpy.hstack([X, X2]))
        Yb = Yb.reshape((len(Yb), 1))
        YS.append(Yb)

    Xext = numpy.vstack(XS)
    Yext = numpy.vstack(YS)
    return Xext, Yext

x, y = multiplie_bruit(X[:1,:], Y[:1], [0, 1, 2])
df = pandas.DataFrame(numpy.hstack([x, y]))
df.columns = ["X1", "X2", "Y0", "Y1", "Y2", "Y'"]
df
```

Le problème est le même qu'avant excepté les variables Y_i qui sont maintenant réel. Au lieu d'être nul, on prend une valeur $Y_i < 0.4$.

```
Xextb, Yextb = multiplie_bruit(X, Y)
df = pandas.DataFrame(numpy.hstack([Xextb, Yextb]))
df.columns = ["X1", "X2", "Y0", "Y1", "Y2", "Y'"]
df.iloc[numpy.random.permutation(df.index), :].head(n=10)
```

```
from sklearn.ensemble import GradientBoostingClassifier
clfb = GradientBoostingClassifier()
clfb.fit(Xextb, Yextb.ravel())
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

```
predb = clfb.predict(Xextb)
confusion_matrix(Yextb, predb)
```

```
array([[299,   1],
       [ 10, 140]], dtype=int64)
```

C'est un petit peu mieux.

9.6.4 Comparaisons de plusieurs modèles

On cherche maintenant à comparer le gain en introduisant du bruit pour différents modèles.

```
def error(model, x, y):
    p = model.predict(x)
    cm = confusion_matrix(y, p)
    return (cm[1,0] + cm[0,1]) / cm.sum()
```

(suite sur la page suivante)

(suite de la page précédente)

```
def comparaison(model, X, Y):

    if isinstance(model, tuple):
        clf = model[0](**model[1])
        clfb = model[0](**model[1])
        model = model[0]
    else:
        clf = model()
        clfb = model()

    Xext, Yext = multiplie(X, Y)
    clf.fit(Xext, Yext.ravel())
    err = error(clf, Xext, Yext)

    Xextb, Yextb = multiplie_bruit(X, Y)
    clfb.fit(Xextb, Yextb.ravel())
    errb = error(clfb, Xextb, Yextb)
    return dict(model=model.__name__, err1=err, err2=errb)

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier,␣
↪AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier, RadiusNeighborsClassifier
from xgboost import XGBClassifier

models = [(LogisticRegression, dict(multi_class="ovr", solver="liblinear")),
          GradientBoostingClassifier,
          (RandomForestClassifier, dict(n_estimators=20)),
          DecisionTreeClassifier,
          ExtraTreeClassifier,
          XGBClassifier,
          (ExtraTreesClassifier, dict(n_estimators=20)),
          (MLPClassifier, dict(activation="logistic")),
          GaussianNB, KNeighborsClassifier,
          (AdaBoostClassifier, dict(base_estimator=LogisticRegression(multi_class="ovr
↪", solver="liblinear"),
                                    algorithm="SAMME"))]

res = [comparaison(model, X, Y) for model in models]
df = pandas.DataFrame(res)
df.sort_values("model")
```

err1 correspond à Y_0, Y_1, Y_2 binaire, *err2* aux mêmes variables mais avec un peu de bruit. L'ajout ne semble pas faire décroître la performance et l'améliore dans certains cas. C'est une piste à suivre. Reste à savoir si les modèles n'apprennent pas le bruit.

9.6.5 Avec une ACP

On peut faire varier le nombre de composantes, j'en ai gardé qu'une. L'ACP est appliquée après avoir ajouté les variables binaires ou binaires bruitées. Le résultat est sans équivoque. Aucun modèle ne parvient à apprendre sans l'ajout de bruit.

```
from sklearn.decomposition import PCA

def comparaison_ACP(model, X, Y):

    if isinstance(model, tuple):
        clf = model[0](**model[1])
        clfb = model[0](**model[1])
        model = model[0]
    else:
        clf = model()
        clfb = model()

    axes = 1
    solver = "full"
    Xext, Yext = multiplie(X, Y)
    Xext = PCA(n_components=axes, svd_solver=solver).fit_transform(Xext)
    clf.fit(Xext, Yext.ravel())
    err = error(clf, Xext, Yext)

    Xextb, Yextb = multiplie_bruit(X, Y)
    Xextb = PCA(n_components=axes, svd_solver=solver).fit_transform(Xextb)
    clfb.fit(Xextb, Yextb.ravel())
    errb = error(clfb, Xextb, Yextb)
    return dict(modelACP=model.__name__, errACP1=err, errACP2=errb)

res = [comparaison_ACP(model, X, Y) for model in models]
dfb = pandas.DataFrame(res)
pandas.concat([ df.sort_values("model"), dfb.sort_values("modelACP")], axis=1)
```

9.6.6 Base d'apprentissage et de test

Cette fois-ci, on s'intéresse à la qualité des frontières que les modèles trouvent en vérifiant sur une base de test que l'apprentissage s'est bien passé.

```
from sklearn.model_selection import train_test_split

def comparaison_train_test(models, X, Y, mbruit=multiplie_bruit, acp=None):

    axes = acp
    solver = "full"

    ind = numpy.random.permutation(numpy.arange(X.shape[0]))
    X = X[ind,:]
    Y = Y[ind]
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1./3)

    res = []
    for model in models:

        if isinstance(model, tuple):
            clf = model[0](**model[1])
            clfb = model[0](**model[1])
            model = model[0]
        else:
            clf = model()
            clfb = model()
```

(suite sur la page suivante)

(suite de la page précédente)

```
        Xext_train, Yext_train = multiplie(X_train, Y_train)
        Xext_test, Yext_test = multiplie(X_test, Y_test)
        if acp:
            Xext_train_ = Xext_train
            Xext_test_ = Xext_test
            acp_model = PCA(n_components=axes, svd_solver=solver).fit(Xext_train)
            Xext_train = acp_model.transform(Xext_train)
            Xext_test = acp_model.transform(Xext_test)
        clf.fit(Xext_train, Yext_train.ravel())

        err_train = error(clf, Xext_train, Yext_train)
        err_test = error(clf, Xext_test, Yext_test)

        Xextb_train, Yextb_train = mbruit(X_train, Y_train)
        Xextb_test, Yextb_test = mbruit(X_test, Y_test)
        if acp:
            acp_model = PCA(n_components=axes, svd_solver=solver).fit(Xextb_train)
            Xextb_train = acp_model.transform(Xextb_train)
            Xextb_test = acp_model.transform(Xextb_test)
            Xext_train = acp_model.transform(Xext_train_)
            Xext_test = acp_model.transform(Xext_test_)
        clfb.fit(Xextb_train, Yextb_train.ravel())

        errb_train = error(clfb, Xextb_train, Yextb_train)
        errb_train_clean = error(clfb, Xext_train, Yext_train)
        errb_test = error(clfb, Xextb_test, Yextb_test)
        errb_test_clean = error(clfb, Xext_test, Yext_test)

        res.append(dict(modelTT=model.__name__, err_train=err_train, err2_train=errb_
↪train,
                err_test=err_test, err2_test=errb_test, err2b_test_clean=errb_test_
↪clean,
                err2b_train_clean=errb_train_clean))

    dfb = pandas.DataFrame(res)
    dfb = dfb[["modelTT", "err_train", "err2_train", "err2b_train_clean", "err_test",
↪"err2_test", "err2b_test_clean"]]
    dfb = dfb.sort_values("modelTT")
    return dfb

dfb = comparaison_train_test(models, X, Y)
dfb
```

Les colonnes *err2b_train_clean* et *err2b_test_clean* sont les erreurs obtenues par des modèles appris sur des colonnes bruitées et testées sur des colonnes non bruitées ce qui est le véritable test. On s'aperçoit que les performances sont très dégradées sur la base d'test. Une raison est que le bruit choisi ajouté n'est pas centré. Corrigeons cela.

```
def multiplie_bruit_centree(X, Y, classes=None):
    if classes is None:
        classes = numpy.unique(Y)
    XS = []
    YS = []
    for i in classes:
        # X2 = numpy.random.randn((X.shape[0]* 3)).reshape(X.shape[0], 3) * 0.1
        X2 = numpy.random.random((X.shape[0], 3)) * 0.2 - 0.1
        X2[:,i] += 1
```

(suite sur la page suivante)

(suite de la page précédente)

```
        Yb = Y == i
        XS.append(numpy.hstack([X, X2]))
        Yb = Yb.reshape((len(Yb), 1))
        YS.append(Yb)

    Xext = numpy.vstack(XS)
    Yext = numpy.vstack(YS)
    return Xext, Yext

dfb = comparaison_train_test(models, X, Y, mbruit=multiplie_bruit_centree, acp=None)
dfb
```

C'est mieux mais on en conclut que dans la plupart des cas, la meilleure performance sur la base d'apprentissage avec le bruit ajouté est due au fait que les modèles apprennent par coeur. Sur la base de test, les performances ne sont pas meilleures. Une erreur de 33% signifie que la réponse du classifieur est constante. On multiplie les exemples.

```
def multiplie_bruit_centree_duplique(X, Y, classes=None):
    if classes is None:
        classes = numpy.unique(Y)
    XS = []
    YS = []
    for i in classes:

        for k in range(0,5):
            #X2 = numpy.random.randn((X.shape[0]* 3)).reshape(X.shape[0], 3) * 0.3
            X2 = numpy.random.random((X.shape[0], 3)) * 0.8 - 0.4
            X2[:,i] += 1
            Yb = Y == i
            XS.append(numpy.hstack([X, X2]))
            Yb = Yb.reshape((len(Yb), 1))
            YS.append(Yb)

    Xext = numpy.vstack(XS)
    Yext = numpy.vstack(YS)
    return Xext, Yext

dfb = comparaison_train_test(models, X, Y, mbruit=multiplie_bruit_centree_duplique,␣
↪acp=None)
dfb
```

Cela fonctionne un peu mieux le fait d'ajouter du hasard ne permet pas d'obtenir des gains significatifs à part pour le modèle SVC [171].

```
def multiplie_bruit_centree_duplique_rebalance(X, Y, classes=None):
    if classes is None:
        classes = numpy.unique(Y)
    XS = []
    YS = []
    for i in classes:

        X2 = numpy.random.random((X.shape[0], 3)) * 0.8 - 0.4
        X2[:,i] += 1  # * ((i % 2) * 2 - 1)
        Yb = Y == i
        XS.append(numpy.hstack([X, X2]))
        Yb = Yb.reshape((len(Yb), 1))
```

(suite sur la page suivante)

171. http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

(suite de la page précédente)

```
        YS.append(Yb)


    Xext = numpy.vstack(XS)
    Yext = numpy.vstack(YS)
    return Xext, Yext

dfb = comparaison_train_test(models, X, Y, mbruit=multiplie_bruit_centree_duplique_
↪rebalance)
dfb
```

9.6.7 Petite explication

Dans tout le notebook, le score de la régression logistique est nul. Elle ne parvient pas à apprendre tout simplement parce que le problème choisi n'est pas linéaire séparable. S'il l'était, cela voudrait dire que le problème suivant l'est aussi.

```
M = numpy.zeros((9, 6))
Y = numpy.zeros((9, 1))
for i in range(0, 9):
    M[i, i//3] = 1
    M[i, i%3+3] = 1
    Y[i] = 1 if i//3 == i%3 else 0
M,Y
```

```
(array([[1., 0., 0., 1., 0., 0.],
        [1., 0., 0., 0., 1., 0.],
        [1., 0., 0., 0., 0., 1.],
        [0., 1., 0., 1., 0., 0.],
        [0., 1., 0., 0., 1., 0.],
        [0., 1., 0., 0., 0., 1.],
        [0., 0., 1., 1., 0., 0.],
        [0., 0., 1., 0., 1., 0.],
        [0., 0., 1., 0., 0., 1.]]), array([[1.],
        [0.],
        [0.],
        [0.],
        [1.],
        [0.],
        [0.],
        [0.],
        [1.]]))
```

```
clf = LogisticRegression(multi_class="ovr", solver="liblinear")
clf.fit(M, Y.ravel())
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

```
clf.predict(M)
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

A revisiter.

9.7 Régression quantile

La régression quantile est moins sensible aux points aberrants. Elle peut être définie comme une régression avec une norme *L1* (une valeur absolue).

9.7.1 Médiane et valeur absolue

On considère un ensemble de nombre réels $\{X_1, ..., X_n\}$. La médiane est le nombre *M* qui vérifie :

$$\sum_i \mathbb{1}_{\{X_i<M\}} = \sum_i \mathbb{1}_{\{X_i>M\}}$$

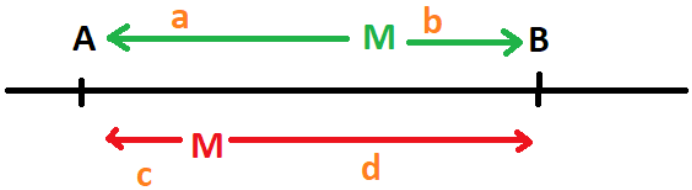
Plus simplement, la médiane est obtenue en triant les éléments $\{X_1, ..., X_n\}$ par ordre croissant. La médiane est alors le nombre au milieu $X_{[\frac{n}{2}]}$.

---

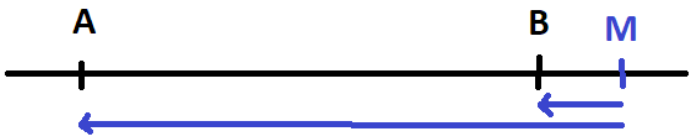
propriété P1 : Médiane et valeur absolue

La médiane *M* de l'ensemble $\{X_1, ..., X_n\}$ minimise la quantité $E = \sum_i |X_i - M|$.

Avant de démontrer la propriété, voyons ce qu'il se passe entre deux réels. La médiane de $\{A, B\}$ peut être n'importe où sur le segment.



De manière évidente, les distances des deux côtés du point *M* sont égales : $a + b = c + d$. Mais si *M* n'est pas sur le segment, on voit de manière évidente que la somme des distances sera plus grande.



N'importe quel point sur le segment *M* minimise $|A-M|+|B-M|$. On revient aux *n* réels triés par ordre croissant $\{X_1,...,X_n\}$ et on considère les paires (X_1,X_n), (X_2,X_{n-1}), ..., $\left(X_{\left[\frac{n}{2}\right]},X_{\left[\frac{n}{2}+1\right]}\right)$. L'intersection de tous ces intervalles est $\left(X_{\left[\frac{n}{2}\right]},X_{\left[\frac{n}{2}+1\right]}\right)$ et on sait d'après la petit exemple avec deux points que n'importe quel point dans cet intervalle minimise $|X_1-M|+|X_n-M|+|X_2-M|+|X_{n-1}-M|+...=E$. La propriété est démontrée.

9.7.2 Régression quantile

Maintenant que la médiane est définie par un problème de minimisation, il est possible de l'appliquer à un problème de régression.

Définition D1 : Régression quantile

On dispose d'un ensemble de *n* couples (X_i,Y_i) avec $X_i \in \mathbb{R}^d$ et $Y_i \in \mathbb{R}$. La régression quantile consiste à trouver α,β tels que la somme $\sum_i |\alpha+\beta X_i-Y_i|$ est minimale.

9.7.3 Résolution d'une régression quantile

La première option consiste à utiliser une méthode de descente de gradient puisque la fonction $E=\sum_i |X_i-M|$ est presque partout dérivable. Une autre option consiste à utiliser l'algorithme Iteratively reweighted least squares [172]. L'implémentation est faite par la classe QuantileLinearRegression [173].

Algorithme A1 : Iteratively reweighted least squares

On souhaite trouver les paramètres Ω qui minimise :

$$E=\sum_i |Y_i-f(X_i,\Omega)|$$

Etape 1

On pose $\forall i,\ w_i^t=1$.

Etape 2

On calcule $\Omega_t=\arg\min E(\Omega)$ avec $E_t(\Omega)=\sum_i w_i^t\left(Y_i-f(X_i,\Omega)\right)^2$.

Etape 3

On met à jour les poids $w_i^{t+1}=\frac{1}{\max\{\delta,|Y_i-f(X_i,\Omega_t)|\}}$. Puis on retourne à l'étape 2.

Le paramètre δ gère le cas où la prédiction est identique à la valeur attendue pour un point X_i donné. Il y a plusieurs choses à démontrer. On suppose que l'algorithme converge, ce qu'on n'a pas encore démontré. Dans ce cas, $\Omega_t=\Omega_{t+1}$ et les coefficients Ω_t optimise la quantité :

$$\sum_i w_i^t\left(Y_i-f(X_i,\Omega)\right)^2=\sum_i \frac{\left(Y_i-f(X_i,\Omega)\right)^2}{\max\delta,|Y_i-f(X_i,\Omega_t)|} \underset{\delta\to 0}{\longrightarrow} \sum_i |Y_i-f(X_i,\Omega)|$$

On remarque également que $E_t(\Omega_t)$ est l'erreur *L1* pour les paramètres Ω. Donc si l'algorithme converge, celui-ci optimise bien l'erreur de la régression quantile. Dans le cas d'une régression linéaire, on sait exprimer la solution :

$$\begin{array}{rcl}\Omega_{t+1} & = & (X'W_tX)^{-1}X'W_ty=g(\Omega_t)\\ \text{avec } W_t & = & diag(\frac{1}{\max\{\delta,|y_i-\Omega_tX_i|\}})\end{array}$$

172. https://en.wikipedia.org/wiki/Iteratively_reweighted_least_squares
173. http://www.xavierdupre.fr/app/mlinsights/helpsphinx/mlinsights/mlmodel/quantile_regression.html#mlinsights.mlmodel.quantile_regression.QuantileLinearRegression

D'après le théorème du point fixe [174], on sait que la suite converge si la fonction *g* est contractante [175].

$$\forall x, y,\ |f(x) - f(y)| \leqslant k \|x - y\| \text{ avec } k < 1$$

9.7.4 Quantile et optimisation

De la même manière que nous avons défini la médiane comme la solution d'un problème d'optimisation, nous pouvons définir n'importe quel quantile comme tel.

---

propriété P2 : Quantile et optimisation

Le quantile Q_p de l'ensemble $\{X_1, ..., X_n\}$ est le nombre qui vérifie :

$$\sum_{i=1}^{n} \mathbb{1}_{\{X_i < Q_p\}} = np$$

Ce nombre minimise la quantité :

$$E = \sum_i p \left|X_i - Q_p\right|^+ + (1-p) \left|X_i - Q_p\right|^-$$

Où $|a|^+ = \max\{a, 0\}$ et $|a|^- = \max\{-a, 0\}$.

On vérifie qu'on retrouve bien ce qui était énoncé pour la médiane avec $p = \frac{1}{2}$. Il faut démontrer que la solution de ce programme d'optimisation atterrit dans l'intervalle souhaité.

On choisit un réel *P* à l'intérieur d'un intervale et on calcule : $E(P) = \sum_i p \left|X_i - P\right|^+ + (1-p) \left|X_i - P\right|^-$. On note $a(P) = \sum_{i=1}^{n} \mathbb{1}_{\{X_i < P\}}$ et $b(P) = \sum_{i=1}^{n} \mathbb{1}_{\{X_i > P\}}$. Comme le point *P* est à l'intérieur d'un intervalle, $a + b = n$. Soit *dx* un réel tel que $P + dx$ soit toujours dans l'intervalle :

$$\begin{array}{rcl} E(P+dx) & = & \sum_i p \left|X_i - P - dx\right|^+ + (1-p) \left|X_i - P - dx\right|^- \\ & = & -b(P)pdx + a(P)(1-p)dx = (a(P) - a(P)p - b(P)p)dx = (a(P) - pn)dx \end{array}$$

On voit que si *P* est choisi de telle sorte que $a(P) = np$, la fonction $E(P)$ est constante sur cette intervalle et c'est précisément le cas lorsque $P = Q_p$. Comme la fonction *E* est une somme positive de fonctions convexes, elle l'est aussi. Si on a trouvé un intervalle où la fonction est constante alors celui-ci contient la solution. Sinon, il suffit juste de trouver les intervalles (X_{i-1}, X_i) et (X_i, X_{i+1}) pour lesquelles la fonction *E* est respectivement décroissante et croissante. On cherche donc le point *P* tel que $a(P) < pn$ si $P < X_i$ et $a(P) > pn$ si $P > X_i$ et ce point correspond au quantile Q_p. Ceci conclut la démonstration.

9.7.5 Régression quantile pour un quantile p quelconque

Comme pour la médiane, il est possible de définir la régression quantile pour un quantile autre que la médiane.

Définition D2 : Régression quantile

On dispose d'un ensemble de *n* couples (X_i, Y_i) avec $X_i \in \mathbb{R}^d$ et $Y_i \in \mathbb{R}$. La régression quantile consiste à trouver α, β tels que la somme $\sum_i p \left|\alpha + \beta X_i - Y_i\right|^+ + (1-p) \left|\alpha + \beta X_i - Y_i\right|^-$ est minimale.

---

174. https://fr.wikipedia.org/wiki/Point_fixe
175. https://fr.wikipedia.org/wiki/Application_contractante

9.7.6 Résolution d'une régression quantile pour un quantile p quelconque

La première option consiste encore à utiliser une méthode de descente de gradient puisque la fonction à minimiser est presque partout dérivable. On peut aussi adapter l'algorithme *Iteratively reweighted least squares* (page 244). L'implémentation est faite par la classe QuantileLinearRegression [176].

Algorithme A2 : Iteratively reweighted least squares

On souhaite trouver les paramètres Ω qui minimise :

$$E = \sum_i p \left|Y_i - f(X_i, \Omega)\right|^+ + (1-p) \left|Y_i - f(X_i, \Omega)\right|^-$$

Etape 1

On pose $\forall i,\ w_i^t = 1$.

Etape 2

On calcule $\Omega_t = \arg\min E(\Omega)$ avec $E_t(\Omega) = \sum_i w_i^t \left(Y_i - f(X_i, \Omega)\right)^2$.

Etape 3

On met à jour les poids $w_i^{t+1} = \frac{1}{\max\left\{\delta, \frac{1}{p}\left|\alpha+\beta X_i - Y_i\right|^+ + \frac{1}{1-p}\left|\alpha+\beta X_i - Y_i\right|^-\right\}}$. Puis on retourne à l'étape 2.

On suppose que l'algorithme converge, ce qu'on n'a pas encore démontré. Dans ce cas, $\Omega_t = \Omega_{t+1}$ et les coefficients Ω_t optimise la quantité :

$$\begin{array}{rcl} \sum_i w_i^t \left(Y_i - f(X_i, \Omega)\right)^2 & = & \sum_i \frac{\left(Y_i - f(X_i, \Omega)\right)^2}{\max\left\{\delta, \frac{1}{p}\left|\alpha+\beta X_i - Y_i\right|^+ + \frac{1}{1-p}\left|\alpha+\beta X_i - Y_i\right|^-\right\}} \\ & \underset{\delta \to 0}{\longrightarrow} & p \left|Y_i - f(X_i, \Omega)\right|^+ + (1-p) \left|Y_i - f(X_i, \Omega)\right|^- \end{array}$$

9.7.7 Notebook

Régression quantile illustrée

La régression quantile est moins sensible aux points aberrants. Elle peut être définie comme une régression avec une norme *L1* (une valeur absolue). Ce notebook explore des régressions avec des quantiles différents.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```



```
%matplotlib inline
```

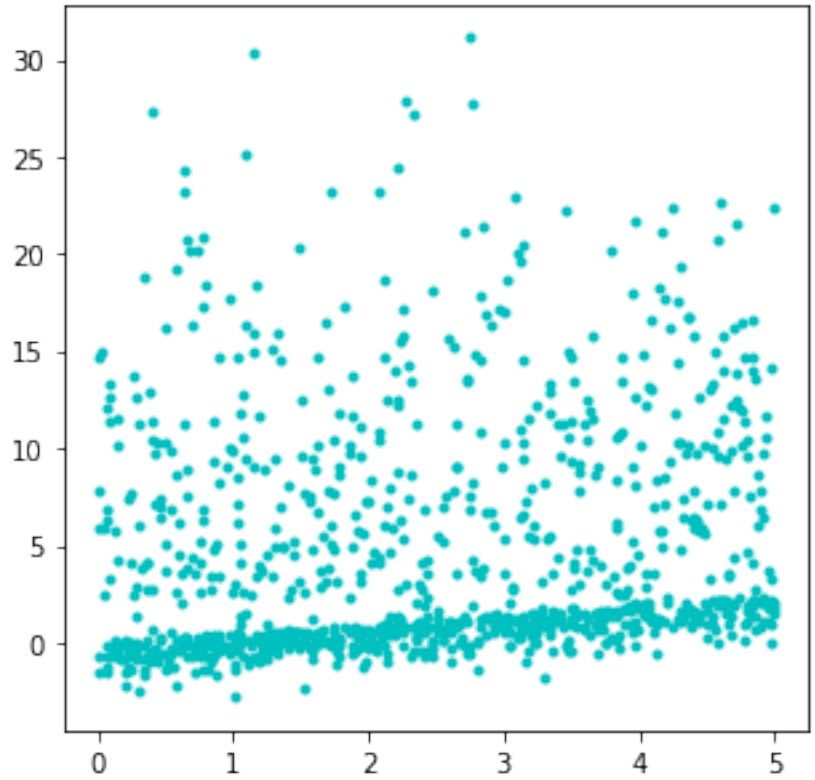
Un jeu de données non symétrique

176. http://www.xavierdupre.fr/app/mlinsights/helpsphinx/mlinsights/mlmodel/quantile_regression.html#mlinsights.mlmodel.quantile_regression.QuantileLinearRegression

```
import numpy.random as npr
import numpy
n = 1000
eps = npr.normal(n)
X = npr.rand(n, 1) * 5
X1 = npr.normal(size=(n, 1)) * 1
X2 = npr.normal(size=(n//2, 1)) * 10
X2 = numpy.vstack([X2, numpy.zeros((n//2, 1))])
eps = - numpy.abs(X1) + numpy.abs(X2)
Y = (0.5 * X + eps).ravel()
X.shape, Y.shape
```

```
((1000, 1), (1000,))
```

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(5,5))
ax.plot(X, Y, 'c.');
```



Régression linéaire et régression quantile

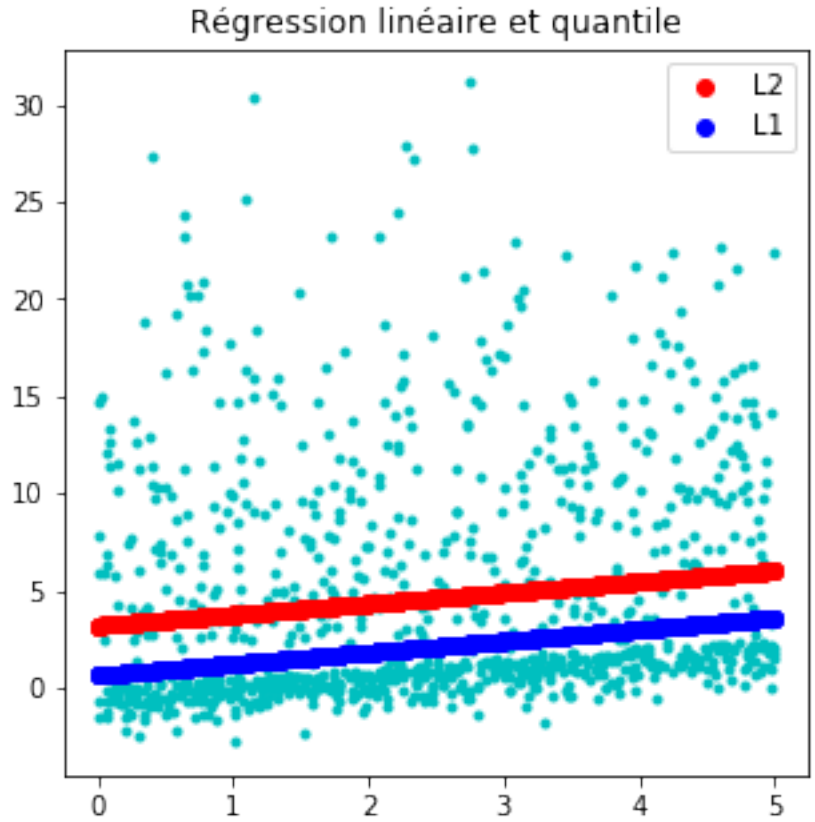
```
from sklearn.linear_model import LinearRegression
clr = LinearRegression()
clr.fit(X, Y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
from mlinsights.mlmodel import QuantileLinearRegression
clq = QuantileLinearRegression()
clq.fit(X, Y)
```

```
QuantileLinearRegression(copy_X=True, delta=0.0001, fit_intercept=True,
             max_iter=10, n_jobs=1, normalize=False, quantile=0.5,
             verbose=False)
```

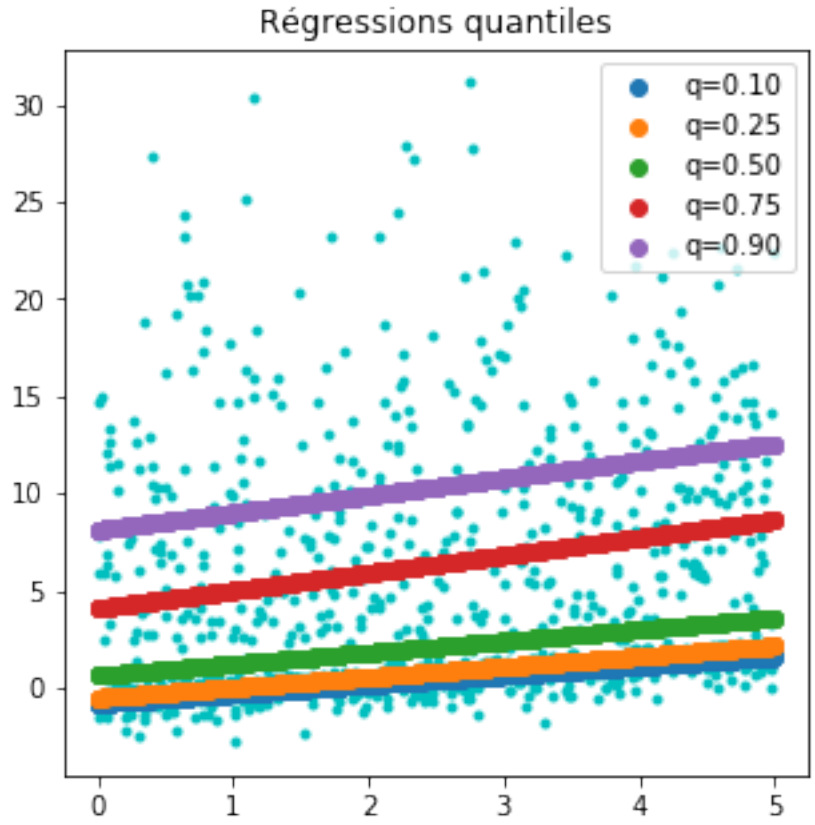
```
fig, ax = plt.subplots(1, 1, figsize=(5,5))
ax.plot(X, Y, 'c.')
lin = clr.predict(X)
ax.plot(X, lin, 'ro', label="L2")
qu = clq.predict(X)
ax.plot(X, qu, 'bo', label="L1")
ax.legend()
ax.set_title("Régression linéaire et quantile");
```



Différents quantiles

```
clqs = {}
for qu in [0.1, 0.25, 0.5, 0.75, 0.9]:
    clq = QuantileLinearRegression(quantile=qu)
    clq.fit(X, Y)
    clqs["q=%1.2f" % qu] = clq
```

```
fig, ax = plt.subplots(1, 1, figsize=(5,5))
ax.plot(X, Y, 'c.')
for k, v in sorted(clqs.items()):
    p = v.predict(X)
    ax.plot(X, p, 'o', label=k)
ax.legend()
ax.set_title("Régressions quantiles");
```



9.7.8 Bilbiographie

CHAPITRE 10

API

10.1 Machine Learning

10.1.1 Métriques

`mlstatpy.ml.MlGridBenchMark` (*self*, *name*, *datasets*, *clog = None*, *fLOG = <function noLOG at 0x7fa874c7b158>*, *path_to_images = ¿.¿*, *cache_file = None*, *progressbar = None*, *graphx = None*, *graphy = None*, *params*)

The class tests a list of model over a list of datasets.

`mlstatpy.ml.ROC` (*self*, *y_true = None*, *y_score = None*, *sample_weight = None*, *df = None*)

Helper to draw a ROC curve

`mlstatpy.ml.voronoi_estimation_from_lr` (*L*, *B*, *C = None*, *D = None*, *cl = 0*, *qr = True*, *max_iter = None*, *verbose = False*)

Determines a Voronoi diagram close to a convex partition defined by a logistic regression in *n* classes. $M \in \mathbb{M}_{nd}$ a row matrix $(L_1, ..., L_n)$. Every border between two classes *i* and *j* is defined by : $\langle L_i, X \rangle + B = \langle L_j, X \rangle + B$.

The function looks for a set of points from which the Voronoi diagram can be inferred. It is done through a linear regression with norm *L1*. See *Régression logistique, diagramme de Voronoï, k-Means* (page 82).

10.2 Traitement du langage naturel

10.2.1 Complétion

`mlstatpy.nlp.CompletionElement` (*self*, *value*, *weight = 1.0*, *disp = None*)

Definition of an element in a completion system, it contains the following members :
— *value* : the completion
— *weight* : a weight or a position, we assume a completion with a lower weight is shown at a lower position
— *disp* : display string (no impact on the algorithm)
— mks0* : value of minimum keystroke
— mks0_* : length of the prefix to obtain *mks0*
— *mks1* : value of dynamic minimum keystroke
— *mks1_* : length of the prefix to obtain *mks1*
— *mks2* : value of modified dynamic minimum keystroke
— *mks2_* : length of the prefix to obtain *mks2*

`empty_prefix` ()

return an instance filled with an empty prefix

`init_metrics` (*self*, *position*, *completions = None*)

initiate the metrics

`str_all_completions` (*self*, *maxn = 10*, *use_precompute = True*)

builds a string with all completions for all prefixes along the paths, this is only available if parameter *completions* was used when calling method `update_metrics`.

`str_mks` (*self*)

return a string with metric information

`str_mks0` (*self*)

return a string with metric information

`update_metrics` (*self*, *prefix*, *position*, *improved*, *delta*, *completions = None*, *iteration = -1*)

update the metrics

`mlstatpy.nlp.CompletionSystem` (*self*, *elements*)

define a completion system

`compare_with_trie` (*self*, *delta = 0.8*, *fLOG = <function noLOG at 0x7fa874c7b158>*)

compare the results with the other implementation

`compute_metrics` (*self*, *ffilter = None*, *delta = 0.8*, *details = False*, *fLOG = <function noLOG at 0x7fa874c7b158>*)

Compute the metric for the completion itself.

`enumerate_test_metric` (*self*, *qset*)

Evaluate the completion set on a set of queries, the function returns a list of `CompletionElement` with the three metrics M, M', $M"$ for these particular queries

`find` (*self*, *value*, *is_sorted = False*)

not very efficient, find an item in a the list

`items` (*self*)

iterate on `(e.value, e)`

`sort_values` (*self*)

sort the elements by value

`sort_weight` (*self*)

sort the elements by value

`test_metric` (*self*, *qset*)

evaluate the completion set on a set of queries, the function returns a dictionary with the aggregated metrics and some statisitcs about them

`to_dict` (*self*)

return a dictionary

`tuples` (*self*)

iterate on `(e.weight, e.value)`

10.2.2 Normalisation

`mlstatpy.data.wikipedia.normalize_wiki_text` (*text*)

Normalizes a text such as a wikipedia title.

`mlstatpy.nlp.remove_diacritics` (*input_str*)

remove diacritics

10.3 Source de données

— *Wikipédia* (page 253)

10.3.1 Wikipédia

`mlstatpy.data.wikipedia.download_dump` (*country*, *name*, *folder = ¿.¿*, *unzip = True*, *timeout = -1*, *overwrite = False*, *fLOG = <function noLOG at 0x7fa874c7b158>*)

Downloads *wikipedia dumps* from `https://dumps.wikimedia.org/frwiki/latest/`.

`mlstatpy.data.wikipedia.download_pageviews` (*dt*, *folder = ¿.¿*, *unzip = True*, *timeout = -1*, *overwrite = False*, *fLOG = <function noLOG at 0x7fa874c7b158>*)

Downloads wikipedia pagacount for a precise date (up to the hours), the url follows the pattern :

```
https://dumps.wikimedia.org/other/pageviews/%Y/%Y-%m/pagecounts-%Y%m%d-
↪%H0000.gz
```

`mlstatpy.data.wikipedia.download_titles` (*country*, *folder = ¿.¿*, *unzip = True*, *timeout = -1*, *overwrite = False*, *fLOG = <function noLOG at 0x7fa874c7b158>*)

Downloads wikipedia titles from `https://dumps.wikimedia.org/frwiki/latest/latest-all-titles-in-ns0.gz`.

`mlstatpy.data.wikipedia.enumerate_titles` (*filename*, *norm = True*, *encoding = ¿utf8¿*)

Enumerates titles from a file.

`mlstatpy.data.wikipedia.download_dump` (*country*, *name*, *folder = ¿.¿*, *unzip = True*, *timeout = -1*, *overwrite = False*, *fLOG = <function noLOG at 0x7fa874c7b158>*)

Downloads *wikipedia dumps* from `https://dumps.wikimedia.org/frwiki/latest/`.

10.4 Graphes

— *Distance* (page 253)

10.4.1 Distance

`mlstatpy.graph.GraphDistance` (*self*, *edge_list*, *vertex_label = None*, *add_loop = False*, *weight_vertex = 1.0*, *weight_edge = 1.0*)

Defines a graph to compute a distance a distance between two graphs.

`distance_matching_graphs_paths` (*self*, *graph2*, *function_mach_vertices = None*, *function_match_edges = None*, *noClean = False*, *store = None*, *use_min = True*, *weight_vertex = 1.0*, *weight_edge = 1.0*)

Computes an alignment between two graphs.

10.5 Image

10.5.1 Conversion

`mlstatpy.image.detection_segment.convert_array2PIL` (*img*, *mode = None*)

Convertit une image donnée sous la forme d'un array au format numpy.array [178].

`mlstatpy.image.detection_segment.convert_PIL2array` (*img*)

Convertit une image donnée sous la forme d'une image Pillow [179] au format numpy.array [180].

10.5.2 Images aléatoires

`mlstatpy.image.detection_segment.random_noise_image` (*size*, *ratio = 0.1*)

Construit une image blanche de taille *size*, noircit aléatoirement *ratio x nb pixels* pixels.

`mlstatpy.image.detection_segment.random_segment_image` (*image*, *lmin = 0.1*, *lmax = 1.0*, *noise = 0.01*, *density = 1.0*)

Ajoute un segment aléatoire à une image. Génère des points le long d'un segment aléatoire.

10.5.3 Segments

`mlstatpy.image.detection_segment.detect_segments` (*image*, *proba_bin = 0.0625*, *cos_angle = 0.9807852804032304*, *seuil_nfa = 1e-05*, *seuil_norme = 2*, *angle = 0.1308996938995747*, *stop = -1*, *verbose = False*)

Détecte les segments dans une image.

`mlstatpy.image.detection_segment.plot_segments` (*image*, *segments*, *outfile = None*, *color = (255, 0, 0)*)

Dessine les segments produits par la fonction `detect_segments`

178. http://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html
179. http://pillow.readthedocs.io/
180. http://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html

CHAPITRE 11

Index

11.1 Index

11.1.1 Modifications

Listes des définitions et théorèmes

Corollaires

Corollaire C1 : Estimateur de l'aire sous la courbe ROC

On dispose des scores $(Y_1, \ldots, Y_n)$ des expériences qui ont réussi et $(X_1, \ldots, X_m)$ les scores des expériences qui ont échoué. On suppose également que tous les scores sont indépendants. Les scores (Y_i) sont identiquement distribués, il en est de même pour les scores (X_i). Un estimateur de l'aire A sous la courbe ROC¿est :

$$\hat{A} = \frac{1}{nm} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\mathbb{1}_{\{Y_j > X_i\}} + \frac{1}{2} \mathbb{1}_{\{Y_j = X_i\}} \right) \tag{11.1}$$

entrée originale (page 134)

Corollaire C1 : approximation d'une fonction créneau

Soit $C \subset \mathbb{R}^p,\ C = \{(y_1, \ldots, y_p) \in \mathbb{R}^p \,|\forall i \in \{1, \cdots, p\},\ 0 \leqslant y_i \leqslant 1\}$, alors :

$$\forall \varepsilon > 0,\ \forall \alpha > 0,\ \exists n \in \mathbb{N}^*,\ \exists\,(x_1, \ldots, x_n) \in (\mathbb{R}^p)^n,\ \exists\,(\gamma_1, \ldots, \gamma_n) \in \mathbb{R}^n \text{ tels que } \forall x \in \mathbb{R}^p,$$

$$\left| \sum_{i=1}^{n} \frac{\gamma_i}{1 + e^{\langle x_i, x \rangle + b_i}} - \mathbb{1}_{\{x \in C\}} \right| \leqslant 1$$

$$\text{et} \quad \inf_{y \in Fr(C)} \|x - y\| > \alpha \Rightarrow \left| \sum_{i=1}^{n} \frac{\gamma_i}{1 + e^{\langle x_i, x \rangle + b_i}} - \mathbb{1}_{\{x \in C\}} \right| \leqslant \varepsilon$$

entrée originale (page 39)

Corollaire C1 : nullité d'un coefficient

Les notations utilisées sont celles du théorème sur *loi asymptotique des coefficients* (page 275). Soit w_k un poids du réseau de neurones d'indice quelconque k. Sa valeur estimée est $\widehat{w_k}$, sa valeur optimale w_k^*. D'après le théorème :

$$N \frac{(\widehat{w_k} - w_k^*)^2}{\widehat{\sigma_N}^2 \left(\widehat{\Sigma_N}^{-1}\right)_{kk}} \overset{T \to +\infty}{\longrightarrow} \chi_1^2$$

entrée originale (page 62)

Corollaire C2 : Variance de l'estimateur AUC

On note $P_X = \mathbb{P}\left(X < \min\{Y_i, Y_j\}\right)$ et $P_Y = \mathbb{P}\left(\max\{X_i, X_j\} < Y\right)$. X_i et X_j sont de même loi que X, Y_i, Y_j sont de même loi que Y. La variance de l'estimateur $\hat{A}$ définie par *(1)* (page **??**) est :

$$\mathbb{V}\hat{A} = \frac{\hat{A}(1 - \hat{A})}{nm} \left[1 + (n-1)\frac{P_Y - \hat{A}^2}{\hat{A}(1 - \hat{A})} + (m-1)\frac{P_X - \hat{A}^2}{\hat{A}(1 - \hat{A})}\right]$$

entrée originale (page 134)

Corollaire C2 : approximation d'une fonction indicatrice

Soit $C \subset \mathbb{R}^p$ compact, alors :

$$\forall \varepsilon > 0,\ \forall \alpha > 0,\ \exists\,(x_1, ..., x_n) \in (\mathbb{R}^p)^n,\ \exists\,(b_1, ..., b_n) \in \mathbb{R}^n \text{ tels que } \forall x \in \mathbb{R}^p,$$

$$\left| \sum_{i=1}^{n} \frac{\gamma_i}{1 + e^{\langle x_i, x \rangle + b_i}} - \mathbb{1}_{\{x \in C\}} \right| \leqslant 1 + 2\varepsilon^2$$

$$\text{et} \quad \inf_{y \in Fr(C)} \|x - y\| > \alpha \Rightarrow \left| \sum_{i=1}^{n} \frac{\gamma_i}{1 + e^{\langle x_i, x \rangle + b_i}} - \mathbb{1}_{\{x \in C\}} \right| \leqslant \varepsilon$$

entrée originale (page 39)

Corollaire C3 : famille libre de fonctions

Soit F_p l'ensemble des fonctions continues de $C \subset \mathbb{R}^p \longrightarrow \mathbb{R}$ avec C compact muni de la norme : $\|f\| = \sup_{x \in C} \|f(x)\|$
Alors l'ensemble E_p des fonctions sigmoïdes :

$$E_p = \left\{ x \longrightarrow 1 - \frac{2}{1 + e^{<y,x>+b}} | y \in \mathbb{R}^p \text{ et } b \in \mathbb{R} \right\}$$

est une base de F_p.

---

entrée originale (page 39)

Définitions

Définition D1 : B+ tree

Soit B_n un Duplicate explicit target name : ńăb+ treeă".

B+ tree [181], soit N un noeud de B_n, il contient un vecteur $V(N) = (x_1, \ldots, x_t)$ avec $0 \leqslant t \leqslant n$ et $x_1 < ... < x_t$. Ce noeud contient aussi exactement $t-1$ noeuds fils notés $(N_1, \ldots, N_{t-1})$. On désigne par $D(N_t)$ l'ensemble des descendants du noeud N_t et $G(N_t) = \{V(M) | M \in D(N_t)\}$. Le noeud N vérifie :

$$\forall x \in G(N_t),\ x_t \leqslant x < x_{t+1}$$
avec par convention $x_0 = -\infty$ et $x_{t+1} = +\infty$

---

entrée originale (page 76)

Définition D1 : Courbe ROC

On suppose que Y est la variable aléatoire des scores des expériences qui ont réussi. X est celle des scores des expériences qui ont échoué. On suppose également que tous les scores sont indépendants. On note F_X et F_Y les fonctions de répartition de ces variables. On définit en fonction d'un seuil $s \in \mathbb{R}$:

— $R(s) = 1 - F_Y(s)$
— $E(s) = 1 - F_X(s)$

La courbe ROC est le graphe $(E(s), R(s))$ lorsque s varie dans $\mathbb{R}$.

---

entrée originale (page 134)

Définition D1 : Dynamic Minimum Keystroke

On définit la façon optimale de saisir une requête sachant un système de complétion S comme étant le minimum obtenu :

$$M'(q,S) \quad = \quad \min_{0 \leqslant k < l(q)} \{M'(q[1..k], S) + \min(K(q,k,S), l(q) - k)\}$$

---

entrée originale (page 118)

Définition D1 : Dynamic Minimum Keystroke arrière

On définit la façon optimale de saisir une requête sachant un système de complétion S comme étant le minimum obtenu :

$$M'_b(q,S) \quad = \quad \min \left\{ \begin{array}{l} \min_{0 \leqslant k < l(q)} \{M'_b(q[1..k], S) + \min(K(q,k,S), l(q) - k)\} \\ \min_{s \succ q} \{M'_b(s,S) + l(s) - l(q)\} \end{array} \right\}$$

181. https://en.wikipedia.org/wiki/B%2B_tree

entrée originale (page 126)

Définition D1 : Minimum Keystroke

On définit la façon optimale de saisir une requête sachant un système de complétion S comme étant le minimum obtenu :

$$M(q,S) = \min_{0 \leqslant k \leqslant l(q)} k + K(q,k,S) \tag{11.2}$$

La quantité $K(q,k,S)$ représente le nombre de touche vers le bas qu'il faut taper pour obtenir la chaîne q avec le système de complétion S et les k premières lettres de q.

entrée originale (page 112)

Définition D1 : Régression quantile

On dispose d'un ensemble de n couples (X_i, Y_i) avec $X_i \in \mathbb{R}^d$ et $Y_i \in \mathbb{R}$. La régression quantile consiste à trouver α, β tels que la somme $\sum_i |\alpha + \beta X_i - Y_i|$ est minimale.

entrée originale (page 246)

Définition D1 : bruit blanc

Une suite de variables aléatoires réelles $(\epsilon_i)_{1 \leqslant i \leqslant N}$ est un bruit blanc :

- $\exists \sigma > 0, \forall i \in \{1, \cdots, N\}\,,\ \epsilon_i \sim \mathcal{N}(0, \sigma)$
- $\forall (i,j) \in \{1, \cdots, N\}^2\,,\ i \neq j \Longrightarrow \epsilon_i \perp\!\!\!\perp \epsilon_j$

entrée originale (page 28)

Définition D1 : loi de Poisson et loi exponentielle

Si une variable X suit une loi de Poisson de paramète λt, elle a pour densité :

$$\mathbb{P}(X = n) \quad = \quad \frac{(\lambda t)^n}{n!}\, e^{-\lambda t} \tag{11.3}$$

Si une variable X suit une loi exponentielle de paramètre μ, elle a pour densité :

$$f(t) \quad = \quad \mu\, e^{-\mu t} \text{ et } \mathbb{P}(X \leqslant t) = \int_0^t \mu\, e^{-\mu x} dx = 1 - e^{-\mu t} \tag{11.4}$$

entrée originale (page 211)

Définition D1 : mot

On note $\mathcal{C}$ l'espace des caractères ou des symboles. Un mot ou une séquence est une suite finie de $\mathcal{C}$. On note $\mathcal{S}_\mathcal{C} = \cup_{k=1}^{\infty} C^k$ l'espace des mots formés de caractères appartenant à $\mathcal{C}$.

entrée originale (page 162)

Définition D1 : mélange de lois normales

Soit X une variable aléatoire d'un espace vectoriel de dimension d, X suit un la loi d'un mélange de N lois gaussiennes de paramètres $(\mu_i, \Sigma_i)_{1 \leqslant i \leqslant N}$, alors la densité f de X est de la forme :

$$f(x) = \sum_{i=1}^{N} p_i \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{\det \Sigma_i}} exp\left(-\frac{1}{2}(x-\mu_i)' \Sigma_i^{-1}(x-\mu_i)\right) \tag{11.5}$$

$$\tag{11.6}$$

Avec : $\sum_{i=1}^{N} p_i = 1$.

---

entrée originale (page 20)

Définition D1 : neurone

Un neurone à p entrées est une fonction $f : \mathbb{R}^{p+1} \times \mathbb{R}^p \longrightarrow \mathbb{R}$ définie par :

- $g : \mathbb{R} \longrightarrow \mathbb{R}$
- $W \in \mathbb{R}^{p+1}, W = (w_1, \ldots, w_{p+1})$
- $\forall x \in \mathbb{R}^p,\ f(W,x) = g\left(\sum_{i=1}^{p} w_i x_i + w_{p+1}\right)$ avec $x = (x_1, \ldots, x_p)$

---

entrée originale (page 27)

Définition D1 : neurone distance

Un neurone distance à p entrées est une fonction $f : \mathbb{R}^{p+1} \times \mathbb{R}^p \longrightarrow \mathbb{R}$ définie par :

- $g : \mathbb{R} \to \mathbb{R}$
- $W \in \mathbb{R}^{p+1}, W = (w_1, \ldots, w_{p+1}) = (W', w_{p+1})$
- $\forall x \in \mathbb{R}^p,\ f(W,x) = e^{-\left\|W'-x\right\|^2 + w_{p+1}}$ avec $x = (x_1, \ldots, x_p)$

---

entrée originale (page 62)

Définition D1 : orthonormalisation de Schmidt

L'orthonormalisation de Shmidt :

Soit $(e_i)_{1 \leqslant i \leqslant N}$ une base de $\mathbb{R}^p$

On définit la famille $(\varepsilon_i)_{1 \leqslant i \leqslant p}$ par :

$$\begin{aligned} \varepsilon_1 &= \frac{e_1}{\|e_1\|} \\ \forall i \in \{1, \cdots, p\},\ \varepsilon_i &= \frac{e_i - \sum_{j=1}^{i-1} < e_i, \varepsilon_j > \varepsilon_j}{\left\| e_i - \sum_{j=1}^{i-1} < e_i, \varepsilon_j > \varepsilon_j \right\|} \end{aligned}$$

---

entrée originale (page 69)

Définition D2 : Dynamic Minimum Keystroke modifié

On définit la façon optimale de saisir une requête sachant un système de complétion S comme étant le minimum obtenu :

$$M"(q,S) = \min \begin{cases} \min_{1 \leqslant k \leqslant l(q)} \{M"(q[1..k-1],S) + 1 + \min(K(q,k,S), l(q)-k)\} \\ \min_{0 \leqslant k \leqslant l(q)} \{M"(q[1..k],S) + \delta + \min(K(q,k,S), l(q)-k)\} \end{cases}$$

entrée originale (page 118)

Définition D2 : Régression quantile

On dispose d'un ensemble de n couples (X_i, Y_i) avec $X_i \in \mathbb{R}^d$ et $Y_i \in \mathbb{R}$. La régression quantile consiste à trouver α, β tels que la somme $\sum_i p\left|\alpha + \beta X_i - Y_i\right|^+ + (1-p)\left|\alpha + \beta X_i - Y_i\right|^-$ est minimale.

entrée originale (page 246)

Définition D2 : couche de neurones

Soit p et n deux entiers naturels, on note $W \in \mathbb{R}^{n(p+1)} = (W_1, \ldots, W_n)$ avec $\forall i \in \{1, \cdots, n\},\ W_i \in \mathbb{R}^{p+1}$. Une couche de n neurones et p entrées est une fonction :

$$F : \mathbb{R}^{n(p+1)} \times \mathbb{R}^p \longrightarrow \mathbb{R}^n$$

vérfifiant :

— $\forall i \in \{1, \cdots, n\},\ f_i$ est un neurone.
— $\forall W \in \mathbb{R}^{n(p+1)} \times \mathbb{R}^p,\ F(W, x) = (f_1(W_1, x), \ldots, f_n(W_n, x))$

entrée originale (page 27)

Définition D2 : distance d'édition

La distance d'édition d sur $\mathcal{S}_\mathcal{C}$ est définie par :

$$\begin{array}{rccc} d: & \mathcal{S}_\mathcal{C} \times \mathcal{S}_\mathcal{C} & \longrightarrow & \mathbb{R}^+ \\ & (m_1, m_2) & \longrightarrow & \min\limits_{\substack{O \text{ séquence} \\ \text{d'opérations}}} d(m_1, m_2, O) \end{array}$$

entrée originale (page 162)

Définition D2 : neurone distance pondérée

Pour un vecteur donné $W \in \mathbb{R}^p = (w_1, \ldots, w_p)$, on note $W_i^j = (w_i, \ldots, w_j)$. Un neurone distance pondérée à p entrées est une fonction $f : \mathbb{R}^{2p+1} \times \mathbb{R}^p \longrightarrow \mathbb{R}$ définie par :

— $g : \mathbb{R} \to \mathbb{R}$
— $W \in \mathbb{R}^{2p+1}, W = (w_1, \ldots, w_{2p+1}) = (w_1, w_{2p+1})$
— $\forall x \in \mathbb{R}^p,\ f(W, x) = \exp\left[-\left[\sum_{i=1}^p w_{p+i}(w_i - x_i)^2\right] + w_{p+1}\right]$ avec $x = (x_1, \ldots, x_p)$

entrée originale (page 62)

Définition D2 : taux de classification à erreur fixe

On cherche un taux de reconnaissance pour un taux d'erreur donné. On dispose pour cela d'une courbe ROC obtenue par l'algorithme de la *courbe ROC* (page **??**) et définie par les points $R_{OC} = \{(e_j, r_j) \,|1 \leqslant j \leqslant k\}$. On suppose ici que $(e_1, r_1) = (1, 1)$ et $(e_k, r_k) = (0,)$. Si ce n'est pas le cas, on ajoute ces valeurs à l'ensemble R_{OC}.

Pour un taux d'erreur donné e^*, on cherche j^* tel que :

$$e_{j^*+1} \leqslant e^* \leqslant e_{j^*}$$

Le taux de reconnaissance ρ cherché est donné par :

$$\rho = \frac{e^* - x_{j^*}}{x_{j^*+1} - x_{j^*}} \left[r_{j^*+1} - r_{j^*}\right] + r_{j^*}$$

---

entrée originale (page 134)

Définition D3 : distance entre caractères

Soit $\mathcal{C}' = \mathcal{C} \bigcup \{.\}$ l'ensemble des caractères ajouté au caractère vide $..$ On note $c : (\mathcal{C}')^2 \longrightarrow \mathbb{R}^+$ la fonction coût définie comme suit :

$$\forall (x,y) \in (\mathcal{C}')^2 \,,\ c(x,y) \text{ est le coût } \begin{cases} \text{d'une comparaison} & \text{si } (x,y) \in (\mathcal{C})^2 \\ \text{d'une insertion} & \text{si } (x,y) \in (\mathcal{C}) \times \{.\} \\ \text{d'une suppression} & \text{si } (x,y) \in \{.\} \times (\mathcal{C}) \\ 0 & \text{si } (x,y) = (\{.\},\{.\}) \end{cases}$$

On note $\mathcal{S}^2_{\mathcal{C}'} = \cup_{n=1}^{\infty} \left(\mathcal{C}'^2\right)^n$ l'ensemble des suites finies de $\mathcal{C}'$.

---

entrée originale (page 162)

Définition D3 : réseau de neurones multi-couches ou perceptron

Un réseau de neurones multi-couches à n sorties, p entrées et C couches est une liste de couches $(C_1, \ldots, C_C)$ connectées les unes aux autres de telle sorte que :

— $\forall i \in \{1, \cdots, C\}$, chaque couche C_i possède n_i neurones et p_i entrées

— $\forall i \in \{1, \cdots, C-1\}\,,\ n_i = p_{i+1}$, de plus $p_1 = p$ et $n_C = n$

Les coefficients de la couche C_i sont notés $\left(W^i_1, \ldots, W^i_{n_i}\right)$, cette couche définit une fonction F_i. Soit la suite $(Z_i)_{0 \leqslant i \leqslant C}$ définie par :

$$\begin{array}{l} Z_0 \in \mathbb{R}^p \\ \forall i \in \{1, \cdots, C\}\,,\ Z_i = F_i\left(W^i_1, \ldots, W^i_{n_i}, Z_{i-1}\right) \end{array}$$

On pose $M = M = \sum_{i=1}^{C} n_i \left(p_i + 1\right)$, le réseau de neurones ainsi défini est une fonction F telle que :

$$\begin{array}{rccc} F : & \mathbb{R}^M \times \mathbb{R}^p & \longrightarrow & \mathbb{R}^n \\ & (W, Z_0) & \longrightarrow & Z_C \end{array}$$

---

entrée originale (page 27)

Définition D4 : mot acceptable

Soit $m = (m_1, \ldots, m_n)$ un mot tel qu'il est défini précédemment. Soit $M = (M_i)_{i \geqslant 1}$ une suite infinie de caractères, on dit que M est un mot acceptable pour m si et seulement si la sous-suite extraite de M contenant tous les caractères différents de $\{.\}$ est égal au mot m. On note $acc(m)$ l'ensemble des mots acceptables pour le mot m.

---

entrée originale (page 162)

Définition D5 : distance d'édition

Soit c la *distance d'édition* (page 261), d définie sur $\mathcal{S}_{\mathcal{C}}$ est définie par :

$$\begin{array}{rccc} d : & \mathcal{S}_{\mathcal{C}} \times \mathcal{S}_{\mathcal{C}} & \longrightarrow & \mathbb{R}^+ \\ & (m_1, m_2) & \longrightarrow & \min\left\{\sum_{i=1}^{+\infty} c\left(M^i_1, M^i_2\right) \mid (M_1, M_2) \in acc(m_1) \times acc(m_2)\right\} \end{array} \tag{11.7}$$

entrée originale (page 162)

Définition D6 : distance d'édition étendue

Soit *d^** la distance d'édition définie en *2* (page 261) pour laquelle les coûts de comparaison, d'insertion et de suppression sont tous égaux à 1. La distance d'édition d' sur $\mathcal{S}_\mathcal{C}$ est définie par :

$$\begin{array}{rccl} d' : & \mathcal{S}_\mathcal{C} \times \mathcal{S}_\mathcal{C} & \longrightarrow & \mathbb{R}^+ \\ & (m_1, m_2) & \longrightarrow & d'(m_1, m_2) = \dfrac{d^*(m_1, m_2)}{\max\{l(m_1), l(m_2)\}} \end{array}$$

où $l(m)$ est la longueur du mot m

entrée originale (page 162)

Définition D7 : distance d'édition tronquée

Soient deux mots (m_1, m_2), on définit la suite :

$$\left(d_{i,j}^{m_1,m_2}\right)_{\substack{0 \leqslant i \leqslant n_1 \\ 0 \leqslant j \leqslant n_2}} \left(= (d_{i,j})_{\substack{0 \leqslant i \leqslant n_1 \\ 0 \leqslant j \leqslant n_2}} \text{ pour ne pas alourdir les notations} \right)$$

Par :

$$\left\{ \begin{array}{l} d_{0,0} = 0 \\ d_{i,j} = \min \left\{ \begin{array}{lcl} d_{i-1,j-1} & + & \text{comparaison } \left(m_1^i, m_2^j\right), \\ d_{i,j-1} & + & \text{insertion } \left(m_2^j\right), \\ d_{i-1,j} & + & \text{suppression } \left(m_1^i\right) \end{array} \right. \end{array} \right.$$

entrée originale (page 162)

Définition D8 : distance d'édition tronquée étendue

Soit deux mots (m_1, m_2), on définit la suite :

$$\left(d_{i,j}^{m_1,m_2}\right)_{\substack{0 \leqslant i \leqslant n_1 \\ 0 \leqslant j \leqslant n_2}} \left(= (d_{i,j})_{\substack{0 \leqslant i \leqslant n_1 \\ 0 \leqslant j \leqslant n_2}} \text{ pour ne pas alourdir les notations} \right)$$

par :

$$\left\{ \begin{array}{l} d_{0,0} = 0 \\ d_{i,j} = \min \left\{ \begin{array}{lcl} d_{i-1,j-1} & + & \text{comparaison } \left(m_1^i, m_2^j\right), \\ d_{i,j-1} & + & \text{insertion } \left(m_2^j, i\right), \\ d_{i-1,j} & + & \text{suppression } \left(m_1^i, j\right), \\ d_{i-2,j-2} & + & \text{permutation } \left(\left(m_1^{i-1}, m_1^i\right), \left(m_2^{j-1}, m_2^j\right)\right) \end{array} \right. \end{array} \right.$$

entrée originale (page 162)

Lemmes

Lemme L1 : Dynamic Minimum Keystroke

On note $d(q, S)$ la longueur du plus long préfixe de q inclus dans S.

$$d(q, S) = \max \{l(p) | p \prec q,\ p \in S,\ p \neq q\}$$

$$M'(q, S) \quad = \quad \min_{d(q,S) \leqslant k < l(q)} \{M'(q[1..k], S) + \min(K(q, k, S), l(q) - k)\}$$

entrée originale (page 120)

Lemme L1 : M¿et sous-ensemble

On suppose que la complétion q est préfixe pour la requête q' et $\sigma(q) < \sigma(q')$ ce qui signifie que la complétion q est toujours affichée avant la complétion q' si elles apparaissent ensemble. Alors $M'(q, S) < M'(q', S)$. Plus spécifiquement, si on considère l'ensemble $S'(q) = \{s - q \in S | q \prec s\}$ ($s - q$ est la complétion s sans son préfixe q).

$$M'(q', S) = M'(q' - q, S') + M'(q, S)$$

entrée originale (page 123)

Lemme L1 : Rang k

On note $M = (m_{ij})$, $W^k = (w_{il}^k)$, $H^k = (h_{lj}^k)$ avec $1 \leqslant i \leqslant p$, $1 \leqslant j \leqslant q$, et $1 \leqslant l \leqslant k$ avec $k < \min(p, q)$. On suppose que les matrices sont solution du problème d'optimisation $\min_{W,H} \|M - WH\|^2$. On suppose que $rang(M) \geqslant k$. Alors les les matrices W^k et H^k sont de rang k.

entrée originale (page 79)

Lemme L1 : inertie minimum

Soit $(X_1, \ldots, X_P) \in \left(\mathbb{R}^N\right)^P$, P points de $\mathbb{R}^N$, le minimum de la quantité $Q\left(Y \in \mathbb{R}^N\right)$:

$$Q(Y) \quad = \quad \sum_{i=1}^{P} d^2(X_i, Y) \tag{11.8}$$

est atteint pour $Y = G = \frac{1}{P} \sum_{i=1}^{P} X_i$ le barycentre des points $(X_1, \ldots, X_P)$.

entrée originale (page 17)

Lemme L2 : Projection

On note $M = (m_{ij})$, $W^k = (w_{il}^k)$, $H^k = (h_{lj}^k)$ avec $1 \leqslant i \leqslant p$, $1 \leqslant j \leqslant q$, et $1 \leqslant l \leqslant k$ avec $k < \min(p, q)$. On suppose que les matrices sont solution du problème d'optimisation $\min_{W,H} \|M - WH\|^2$. On considère que la matrice M est un ensemble de q points dans dans un espace vectoriel de dimension p. La matrice WH représente des projections de ces points dans l'espace vectoriel engendré par les k vecteurs colonnes de la matrice W.

entrée originale (page 79)

Lemme L2 : calcul de *M¿(q, S)*

On suppose que $p(q, S)$ est la complétion la plus longue de l'ensemble S qui commence q :

$$\begin{aligned} k^* &= \max\{k | q[[1..k]] \prec q \text{ et } q \in S\} \\ p(q, S) &= q[[1..k^*]] \end{aligned}$$

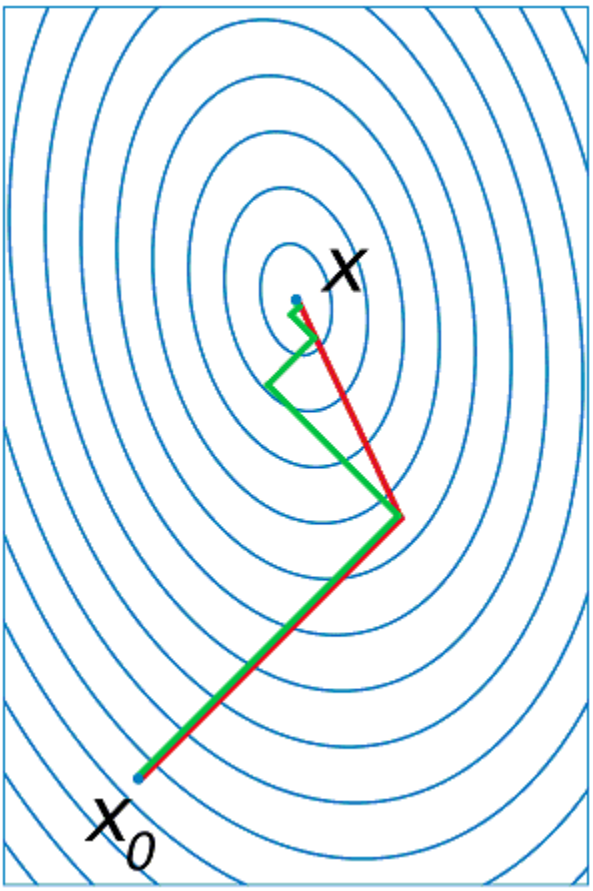
La métrique $M'(q, S)$ vérifie la propriété suivante :

$$M'(q, S) = M'(p(q, S), S) + l(q) - l(p(q, S))$$

entrée originale (page 120)

Figures

Figure F1 : Gradient conjugué

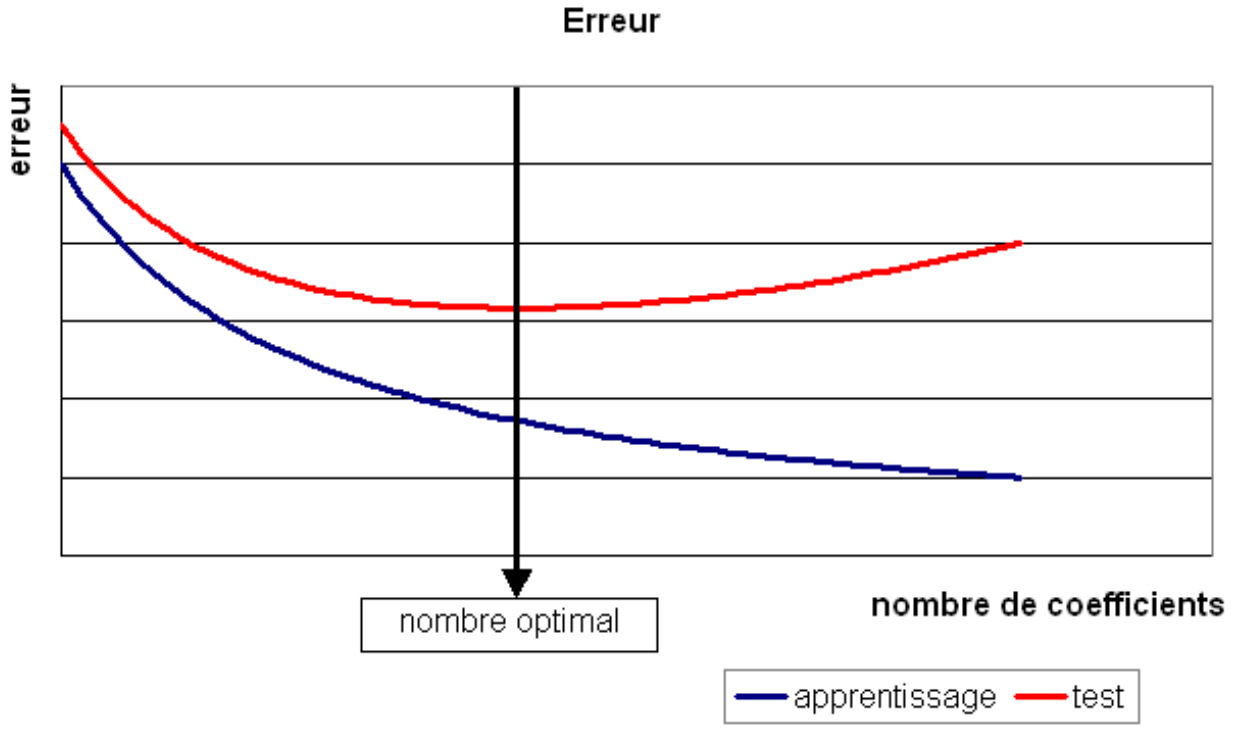


Gradient et gradient conjugué sur une ligne de niveau de la fonction $G(x, y) = 3x^2 + y^2$, le gradient est orthogonal aux lignes de niveaux de la fonction G, mais cette direction est rarement la bonne à moins que le point (x, y) se situe sur un des axes des ellipses, le gradient conjugué agrège les derniers déplacements et propose une direction de recherche plus plausible pour le minimum de la fonction. Voir Conjugate Gradient Method [182].

182. https://en.wikipedia.org/wiki/Conjugate_gradient_method

entrée originale (page 50)

Figure F1 : Modèle optimal pour la base de test



entrée originale (page 62)

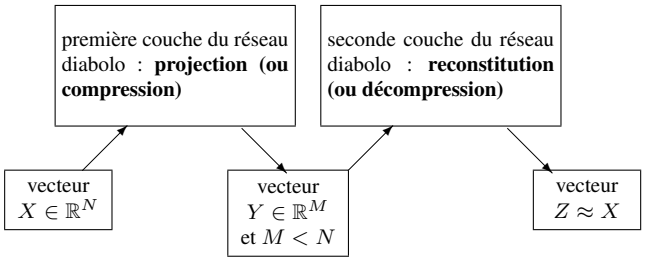
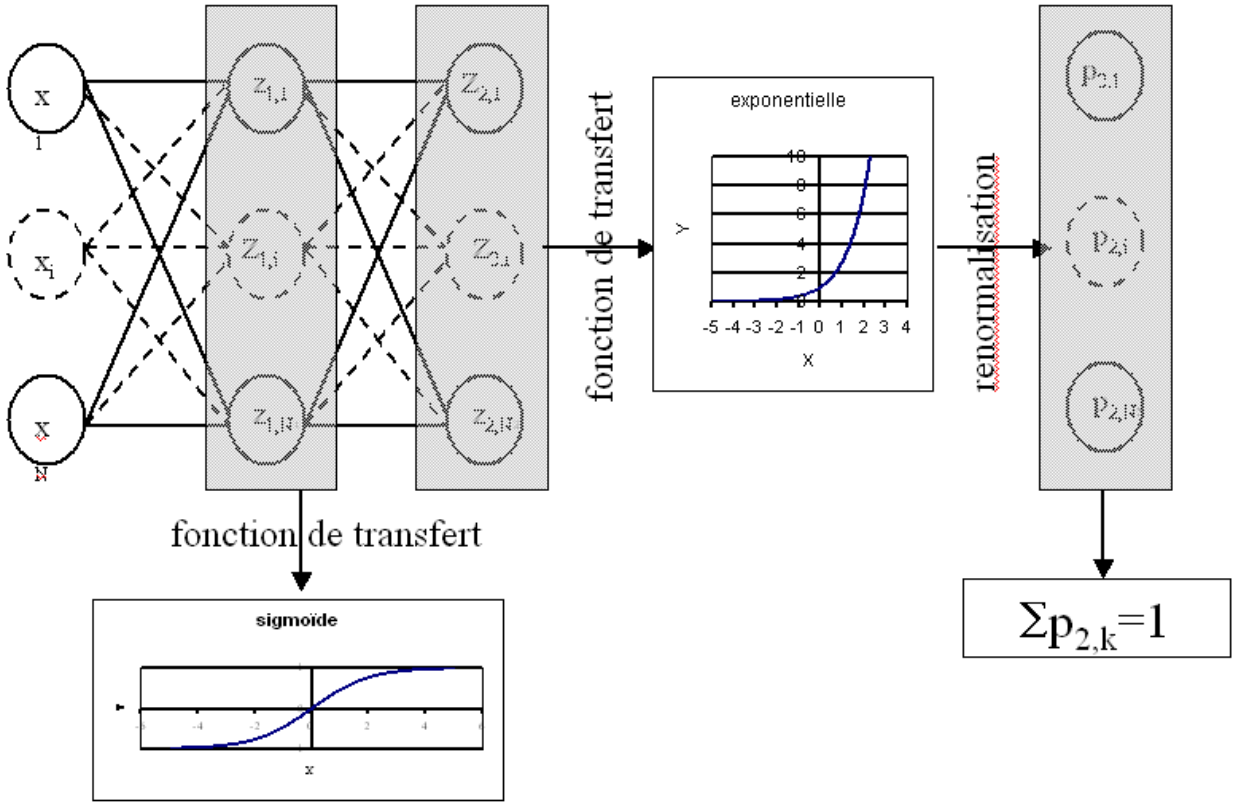


Figure F1 : Principe de la compression par un réseau diabolo

entrée originale (page 69)

Figure F1 : Réseau de neurones adéquat pour la classification



entrée originale (page 55)

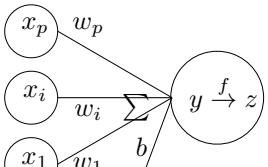


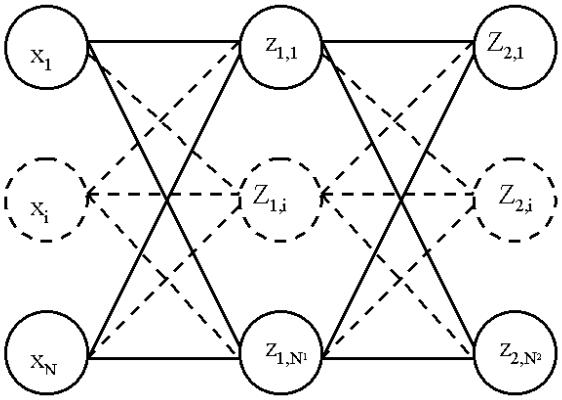
Figure F1 : neurone graphique Le vecteur $(x_1, ..., x_p) \in \mathbb{R}^p$ joue le rôle des *entrées*. y est appelé parfois le *potentiel*. $y = \sum_{i=1}^{p} w_i x_i + b$. z est appelée la sortie du neurone. f est appelée la fonction de transfert ou de seuil. $z = f(y) = f(\sum_{i=1}^{p} w_i x_i + b)$.[1]

entrée originale (page 27)

Figure F2 : Exemple de minimal locaux

entrée originale (page 50)

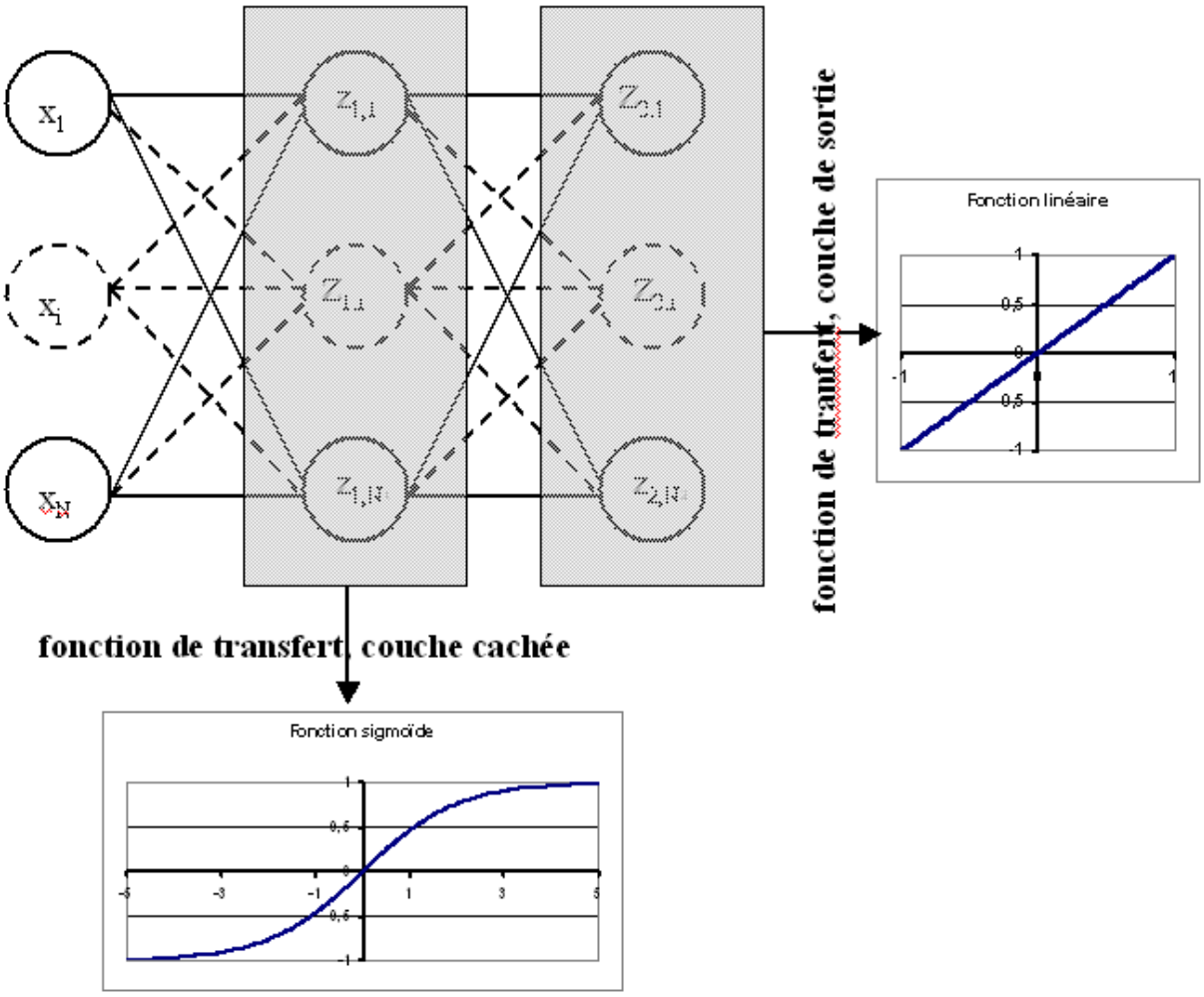
Figure F2 : Modèle du perceptron multi-couche (multi-layer perceptron, MLP)



— $(x_1, \ldots, x_p)$: entrées
— C_i nombre de neurones sur la couche i, $C_0 = p$
— $z_{c,i}$ sortie du neurone i, de la couche c, par extension, $z_{0,i} = x_i$
— $y_{c,i}$ potentiel du neurone i de la couche c
— $w_{c,i,j}$ coefficient associé à l'entrée j du neurone i de la couche c,
— $b_{c,i}$ biais du neurone i de la couche c
— $f_{c,i}$ fonction de seuil du neurone i de la couche c

entrée originale (page 27)

Figure F2 : Réseau de neurones pour lequel la sélection de connexions s'applique



entrée originale (page 62)

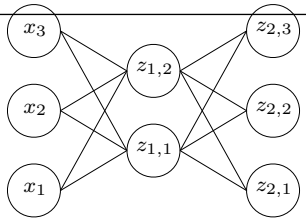
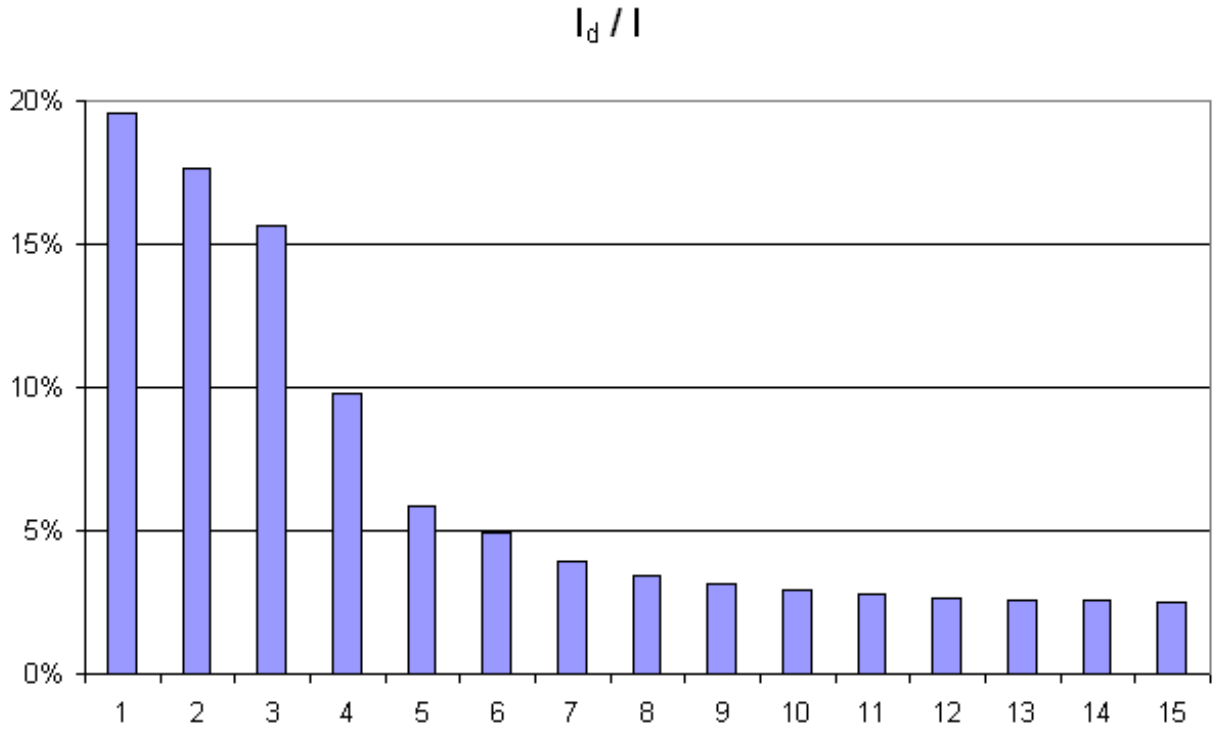


Figure F2 : Réseau diabolo : réduction d'une dimension Ce réseau possède 3 entrées et 3 sorties Minimiser l'erreur $\sum_{k=1}^{N} E(X_k, X_k)$ revient à compresser un vecteur de dimension 3 en un vecteur de dimension 2. Les coefficients de la première couche du réseau de neurones permettent de compresser les données. Les coefficients de la seconde couche permettent de les décompresser.

entrée originale (page 69)

Figure F3 : Courbe d'inertie pour l'ACP



Courbe d'inertie : point d'inflexion pour $d = 4$, l'expérience montre que généralement, seules les projections sur un ou plusieurs des quatre premiers vecteurs propres reflètera l'information contenue par le nuage de points.

---

entrée originale (page 69)

Problèmes

Problème P1 : Classification

Soit une variable aléatoire X et une variable aléatoire discrète Y, l'objectif est d'approximer la fonction $\mathbb{E}(Y|X) = f(X)$. Les données du problème sont un échantillon de points : $\{(X_i, Y_i)\,|1 \leqslant i \leqslant N\}$ avec $\forall i \in \{1, \ldots, N\}$, $Y_i \in \{1, \ldots, C\}$ et un modèle paramétré avec θ :

$$\forall i \in \{1, \cdots, N\}\,,\ \forall c \in \{1, \cdots, C\}\,,\ \mathbb{P}(Y_i = c|X_i, \theta) = h(\theta, X_i, c)$$

avec $n \in \mathbb{N}$, h est une fonction de paramètre θ à valeur dans $[0, 1]$ et vérifiant la contrainte : $\sum_{c=1}^{C} h(\theta, X, c) = 1$.

---

entrée originale (page 32)

Problème P1 : Factorisation de matrices positifs

Soit $M \in \mathcal{M}_{pq}$, on cherche les matrices à coefficients positifs $W \in \mathcal{M}_{pk}$ et $H \in \mathcal{M}_{kq}$ qui sont solution du problème d'optimisation :

$$\min_{W,h} \left\{ \|M - WH\|^2 \right\} = \min_{W,H} \sum_{ij} (m_{ij} - \sum_{k} w_{ik} h_{kj})^2$$

---

entrée originale (page 79)

Problème P1 : Optimiser un système de complétion

On suppose que l'ensemble des complétions $C = \{c_j\}$ est connu. On souhaite ordonner cet ensemble pour obtenir l'ensemble ordonné des complétions $S = (s_i)$ qu'on considère comme une permutation σ de l'ensemble de départ : $S(\sigma) = (s_i) = (c_{\sigma(j)})$. Ce système de complétion est destiné à un des utilisateurs qui forment des recherches ou requêtes $Q = (q_i, w_i)_{1 \leqslant i \leqslant N_Q}$. q_i est la requête, w_i est la fréquence associée à cette requête. On définit l'effort demandé aux utilisateurs par ce système de complétion :

$$E(C, Q, \sigma) = \sum_{i=1}^{N_Q} w_i M'(q_i, S(\sigma))$$

Déterminer le meilleur système de complétion revient à trouver la permutation σ qui minimise $E(C, Q, \sigma)$.

entrée originale (page 123)

Problème P1 : Régression

Soient deux variables aléatoires X et Y, l'objectif est d'approximer la fonction $\mathbb{E}(Y|X) = f(X)$. Les données du problème sont un échantillon de points $\{(X_i, Y_i) \,|1 \leqslant i \leqslant N\}$ et un modèle paramétré avec :math :theta‘ :

$$\forall i \in \{1, \cdots, N\}, \; Y_i = f(\theta, X_i) + \epsilon_i$$

avec $n \in \mathbb{N}$, $(\epsilon_i)_{1 \leqslant i \leqslant N}$ *bruit blanc* (page 258), f est une fonction de paramètre θ.

entrée originale (page 28)

Problème P1 : analyse en composantes principales (ACP)

Soit $(X_i)_{1 \leqslant i \leqslant N}$ avec $\forall i \in \{1, \ldots, N\}, \; X_i \in \mathbb{R}^p$. Soit $W \in M_{p,d}(\mathbb{R})$, $W = (C_1, \ldots, C_d)$ où les vecteurs (C_i) sont les colonnes de W et $d < p$. On suppose également que les (C_i) forment une base othonormée. Par conséquent :

$$W'W = I_d$$

$(W'X_i)_{1 \leqslant i \leqslant N}$ est l'ensemble des vecteurs (X_i) projetés sur le sous-espace vectoriel engendré par les vecteurs (C_i). Réaliser une analyse en composantes principales, c'est trouver le meilleur plan de projection pour les vecteurs (X_i), celui qui maximise l'inertie de ce nuage de points, c'est donc trouver W^* tel que :

$$W^* = \underset{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}}{\arg\max} \; E(W) = \underset{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}}{\arg\max} \left[\sum_{i=1}^{N} \|W'X_i\|^2 \right]$$

Le terme $E(W)$ est l'inertie du nuage de points (X_i) projeté sur le sous-espace vectoriel défini par les vecteurs colonnes de la matrice W.

entrée originale (page 69)

Problème P1 : estimateur du maximum de vraisemblance

Soit un vecteur $(d_1, \ldots, d_N)$ tel que :

$$\left\{ \begin{array}{l} \sum_{k=1}^{N} d_k = 1 \\ \forall k \in \{1, \ldots, N\}, \; d_k \geqslant 0 \end{array} \right.$$

On cherche le vecteur $(p_1^*, \ldots, p_N^*)$ vérifiant :

$$(p_1^*, \ldots, p_N^*) = \underset{(p_1,\ldots,p_C)\in\mathbb{R}^C}{\arg\max} \sum_{k=1}^{C} d_k \ln p_k$$

$$\text{avec } \left\{ \begin{array}{l} \forall k \in \{1, \cdots, C\},\ p_k \geqslant 0 \\ \text{et } \sum_{k=1}^{C} p_k = 1 \end{array} \right.$$

---

entrée originale (page 55)

Problème P2 : Optimiser un système de complétion filtré

On suppose que l'ensemble des complétions $C = \{c_j\}$ est connu. On souhaite ordonner cet ensemble pour obtenir l'ensemble ordonné des complétions $S = (s_i)$ qu'on considère comme une permutation σ de l'ensemble de départ : $S(\sigma) = (s_i) = (c_{\sigma(j)})$. On utilise aussi une fonction f qui filtre les suggestions montrées à l'utilisateur, elle ne change pas l'ordre mais peut cacher certaines suggestions si elles ne sont pas pertinentes. Ce système de complétion est destiné à un des utilisateurs qui forment des recherches ou requêtes $Q = (q_i, w_i)_{1\leqslant i\leqslant N_Q}$. q_i est la requête, w_i est la fréquence associée à cette requête. On définit l'effort demandé aux utilisateurs par ce système de complétion :

$$E(C, Q, \sigma, f) = \sum_{i=1}^{N_Q} w_i M'(q_i, S(\sigma), f)$$

Déterminer le meilleur système de complétion revient à trouver la permutation σ qui minimise $E(C, Q, \sigma, f)$.

---

entrée originale (page 123)

Problème P2 : classification

Soit A l'échantillon suivant :

$$A = \left\{ \left(X_i, y_i = \left(\eta_i^k\right)_{1\leqslant k\leqslant C} \right) \in \mathbb{R}^p \times \mathbb{R}^C \text{ tel que } \sum_{k=1}^{c} \eta_i^k = 1 \middle| \ 1 \leqslant i \leqslant N \right\}$$

y_i^k représente la probabilité que l'élément X_i appartiennent à la classe k : $\eta_i^k = \mathbb{P}\left(Y_i = k | X_i\right)$

Le classifieur cherché est une fonction f définie par :

$$\begin{array}{rcl} f : \mathbb{R}^M \times \mathbb{R}^p & \longrightarrow & \mathbb{R}^C \\ (W, X) & \longrightarrow & \left(f_1(W, X), \ldots, f_p(W, X)\right) \end{array}$$

Dont le vecteur de poids W^* est égal à :

$$W^* = \underset{W}{\arg\max} \sum_{i=1}^{N} \sum_{k=1}^{C} \eta_i^k f_k(W, X_i) - \sum_{i=1}^{N} \ln \left[\sum_{l=1}^{C} e^{f_l(W, X_i)} \right]$$

---

entrée originale (page 55)

Propriétés

Problème P1 : Classification

Soit une variable aléatoire X et une variable aléatoire discrète Y, l'objectif est d'approximer la fonction $\mathbb{E}(Y|X) = f(X)$. Les données du problème sont un échantillon de points : $\{(X_i, Y_i)\,|1 \leqslant i \leqslant N\}$ avec $\forall i \in \{1, \ldots, N\},\ Y_i \in \{1, \ldots, C\}$ et un modèle paramétré avec θ :

$$\forall i \in \{1, \cdots, N\},\ \forall c \in \{1, \cdots, C\},\ \mathbb{P}(Y_i = c|X_i, \theta) = h(\theta, X_i, c)$$

avec $n \in \mathbb{N}$, h est une fonction de paramètre θ à valeur dans $[0, 1]$ et vérifiant la contrainte : $\sum_{c=1}^{C} h(\theta, X, c) = 1$.

---

entrée originale (page 32)

Problème P1 : Factorisation de matrices positifs

Soit $M \in \mathcal{M}_{pq}$, on cherche les matrices à coefficients positifs $W \in \mathcal{M}_{pk}$ et $H \in \mathcal{M}_{kq}$ qui sont solution du problème d'optimisation :

$$\min_{W,h} \left\{ \|M - WH\|^2 \right\} = \min_{W,H} \sum_{ij} (m_{ij} - \sum_{k} w_{ik} h_{kj})^2$$

---

entrée originale (page 79)

Problème P1 : Optimiser un système de complétion

On suppose que l'ensemble des complétions $C = \{c_j\}$ est connu. On souhaite ordonner cet ensemble pour obtenir l'ensemble ordonné des complétions $S = (s_i)$ qu'on considère comme une permutation σ de l'ensemble de départ : $S(\sigma) = (s_i) = (c_{\sigma(j)})$. Ce système de complétion est destiné à un des utilisateurs qui forment des recherches ou requêtes $Q = (q_i, w_i)_{1 \leqslant i \leqslant N_Q}$. q_i est la requête, w_i est la fréquence associée à cette requête. On définit l'effort demandé aux utilisateurs par ce système de complétion :

$$E(C, Q, \sigma) = \sum_{i=1}^{N_Q} w_i M'(q_i, S(\sigma))$$

Déterminer le meilleur système de complétion revient à trouver la permutation σ qui minimise $E(C, Q, \sigma)$.

---

entrée originale (page 123)

Problème P1 : Régression

Soient deux variables aléatoires X et Y, l'objectif est d'approximer la fonction $\mathbb{E}(Y|X) = f(X)$. Les données du problème sont un échantillon de points $\{(X_i, Y_i)\,|1 \leqslant i \leqslant N\}$ et un modèle paramétré avec :math :theta‘ :

$$\forall i \in \{1, \cdots, N\},\ Y_i = f(\theta, X_i) + \epsilon_i$$

avec $n \in \mathbb{N}$, $(\epsilon_i)_{1 \leqslant i \leqslant N}$ *bruit blanc* (page 258), f est une fonction de paramètre θ.

---

entrée originale (page 28)

Problème P1 : analyse en composantes principales (ACP)

Soit $(X_i)_{1 \leqslant i \leqslant N}$ avec $\forall i \in \{1, \ldots, N\},\ X_i \in \mathbb{R}^p$. Soit $W \in M_{p,d}(\mathbb{R})$, $W = (C_1, \ldots, C_d)$ où les vecteurs (C_i) sont les colonnes de W et $d < p$. On suppose également que les (C_i) forment une base othonormée. Par conséquent :

$$W'W = I_d$$

$(W'X_i)_{1 \leqslant i \leqslant N}$ est l'ensemble des vecteurs (X_i) projetés sur le sous-espace vectoriel engendré par les vecteurs (C_i). Réaliser une analyse en composantes principales, c'est trouver le meilleur plan de projection pour les vecteurs (X_i), celui qui maximise l'inertie de ce nuage de points, c'est donc trouver W^* tel que :

$$W^* = \underset{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}}{\arg\max} E(W) = \underset{\substack{W \in M_{p,d}(\mathbb{R}) \\ W'W = I_d}}{\arg\max} \left[\sum_{i=1}^{N} \|W'X_i\|^2 \right]$$

Le terme $E(W)$ est l'inertie du nuage de points (X_i) projeté sur le sous-espace vectoriel défini par les vecteurs colonnes de la matrice W.

---

entrée originale (page 69)

Problème P1 : estimateur du maximum de vraisemblance

Soit un vecteur $(d_1, \ldots, d_N)$ tel que :

$$\left\{ \begin{array}{l} \sum_{k=1}^{N} d_k = 1 \\ \forall k \in \{1, \ldots, N\}, \; d_k \geqslant 0 \end{array} \right.$$

On cherche le vecteur $(p_1^*, \ldots, p_N^*)$ vérifiant :

$$(p_1^*, \ldots, p_N^*) = \underset{(p_1, \ldots, p_C) \in \mathbb{R}^C}{\arg\max} \sum_{k=1}^{C} d_k \ln p_k$$

$$\text{avec} \left\{ \begin{array}{l} \forall k \in \{1, \cdots, C\}, \; p_k \geqslant 0 \\ \text{et } \sum_{k=1}^{C} p_k = 1 \end{array} \right.$$

---

entrée originale (page 55)

Problème P2 : Optimiser un système de complétion filtré

On suppose que l'ensemble des complétions $C = \{c_j\}$ est connu. On souhaite ordonner cet ensemble pour obtenir l'ensemble ordonné des complétions $S = (s_i)$ qu'on considère comme une permutation σ de l'ensemble de départ : $S(\sigma) = (s_i) = (c_{\sigma(j)})$. On utilise aussi une fonction f qui filtre les suggestions montrées à l'utilisateur, elle ne change pas l'ordre mais peut cacher certaines suggestions si elles ne sont pas pertinentes. Ce système de complétion est destiné à un des utilisateurs qui forment des recherches ou requêtes $Q = (q_i, w_i)_{1 \leqslant i \leqslant N_Q}$. q_i est la requête, w_i est la fréquence associée à cette requête. On définit l'effort demandé aux utilisateurs par ce système de complétion :

$$E(C, Q, \sigma, f) = \sum_{i=1}^{N_Q} w_i M'(q_i, S(\sigma), f)$$

Déterminer le meilleur système de complétion revient à trouver la permutation σ qui minimise $E(C, Q, \sigma, f)$.

---

entrée originale (page 123)

Problème P2 : classification

Soit A l'échantillon suivant :

$$A = \left\{ \left(X_i, y_i = \left(\eta_i^k\right)_{1 \leqslant k \leqslant C} \right) \in \mathbb{R}^p \times \mathbb{R}^C \text{ tel que } \sum_{k=1}^{c} \eta_i^k = 1 \middle| \; 1 \leqslant i \leqslant N \right\}$$

y_i^k représente la probabilité que l'élément X_i appartiennent à la classe k : $\eta_i^k = \mathbb{P}(Y_i = k | X_i)$

Le classifieur cherché est une fonction f définie par :

$$\begin{array}{rcl} f : \mathbb{R}^M \times \mathbb{R}^p & \longrightarrow & \mathbb{R}^C \\ (W, X) & \longrightarrow & (f_1(W, X), \ldots, f_p(W, X)) \end{array}$$

Dont le vecteur de poids W^* est égal à :

$$W^* = \arg\max_W \sum_{i=1}^{N} \sum_{k=1}^{C} \eta_i^k f_k(W, X_i) - \sum_{i=1}^{N} \ln \left[\sum_{l=1}^{C} e^{f_l(W, X_i)} \right]$$

---

entrée originale (page 55)

Tables

Théorèmes

Théorème T1 : Aire sous la courbe (AUC)

On utilise les notations de la définition de la *Courbe ROC* (page 257). L'aire sous la courbe ROC est égale à $\mathbb{P}(Y > X)$.

---

entrée originale (page 134)

Théorème T1 : La factorisation de matrice est équivalente à une analyse en composantes principales

On note $M = (m_{ij})$, $W^k = (w_{il}^k)$, $H^k = (h_{lj}^k)$ avec $1 \leqslant i \leqslant p$, $1 \leqslant j \leqslant q$, et $1 \leqslant l \leqslant k$ avec $k < \min(p, q)$. On suppose que les matrices sont solution du problème d'optimisation $\min_{W,H} \|M - WH\|^2$. On considère que la matrice M est un ensemble de q points dans dans un espace vectoriel de dimension p. On suppose $p < q$. La matrice W_k définit un hyperplan identique à celui défini par les k vecteurs propres associés aux k plus grande valeurs propres de la matrice MM' où M' est la transposée de M.

---

entrée originale (page 79)

Théorème T1 : M¿, ordre et sous-ensemble

Soit q une requête de l'ensemble de complétion S ordonnées selon $sigma$. Si cet ordre vérifie :

$$\forall k,\ \sigma(q[1..k]) \leqslant \sigma(q[1..k+1]) \tag{11.9}$$

On note l'ensemble $S'(q[1..k]) = \{q[k+1..len(q)] \in S\}$:

alors :

$$\forall k,\ M'(q[1..k], S) = M'(q[k+1..l(q)], S'(q[1..k]) + M'(q[1..k], S)$$

---

entrée originale (page 123)

Théorème T1 : [Farago1993]_ 1

Les notations sont celles de l'algorithme précédent. Il retourne le plus proche voisin x^* de x inclus dans E. Autrement dit, $\forall x \in X,\ x^* \in F(x)$.

---

entrée originale (page 76)

Théorème T1 : convergence de la méthode de Newton

[Bottou1991] (page **??**)

Soit une fonction continue $g : W \in \mathbb{R}^M \to \mathbb{R}$ de classe C^1. On suppose les hypothèses suivantes vérifiées :

— **H1** : $\arg\min_{W \in \mathbb{R}^q} g(W) = \{W^*\}$ est un singleton

— **H2** : $\forall \varepsilon > 0, \inf_{|W - W^*| > \varepsilon} \left[(W - W^*)' . \nabla g(W) \right] > 0$

— **H3** : $\exists (A, B) \in \mathbb{R}^2$ tels que $\forall W \in \mathbb{R}^p, \ \|\nabla g(W)\|^2 \leqslant A^2 + B^2 \|W - W^*\|^2$

— **H4** : la suite $(\varepsilon_t)_{t \geqslant 0}$ vérifie, $\forall t > 0, \ \varepsilon_t \in \mathbb{R}_+^*$ et $\sum_{t \geqslant 0} \varepsilon_t = +\infty$, $\sum_{t \geqslant 0} \varepsilon_t^2 < +\infty$

Alors la suite $(W_t)_{t \geqslant 0}$ construite de la manière suivante $W_0 \in \mathbb{R}^M$, $\forall t \geqslant 0 : W_{t+1} = W_t - \varepsilon_t \nabla g(W_t)$ vérifie $\lim_{t \to +\infty} W_t = W^*$.

---

entrée originale (page 44)

Théorème T1 : convergence des k-means

Quelque soit l'initialisation choisie, la suite $(I_t)_{t \geqslant 0}$ construite par l'algorithme des *k-means* (page **??**) converge.

---

entrée originale (page 17)

Théorème T1 : convexité des classes formées par une régression logistique

On définit l'application $\mathbb{R}^d \to \mathbb{N}$ qui associe la plus grande coordonnée $f(X) = \arg\max_k (AX + B)_k$. *A* est une matrice $\mathcal{M}_{dc}$, *B* est un vecteur de $\mathbb{R}^d$, c est le nombre de parties. L'application f définit une partition convexe de l'espace vectoriel $\mathbb{R}^d$.

---

entrée originale (page 84)

Théorème T1 : densité des réseaux de neurones (Cybenko1989)

[Cybenko1989] (page **??**) Soit E_p^q l'espace des réseaux de neurones à p entrées et q sorties, possédant une couche cachée dont la fonction de seuil est une fonction sigmoïde $\left(x \to 1 - \frac{2}{1+e^x} \right)$, une couche de sortie dont la fonction de seuil est linéaire Soit F_p^q l'ensemble des fonctions continues de $C \subset \mathbb{R}^p \longrightarrow \mathbb{R}^q$ avec C compact muni de la norme $\|f\| = \sup_{x \in C} \|f(x)\|$ Alors E_p^q est dense dans F_p^q.

---

entrée originale (page 39)

Théorème T1 : distance d'édition

Soit c et d les fonctions définies respectivement par *(1)* (page **??**) et *(2)* (page **??**), alors :

c est une distance sur $\mathcal{C} \iff d$ est une distance sur $\mathcal{S}_{\mathcal{C}}$

---

entrée originale (page 162)

Théorème T1 : loi asymptotique des coefficients

Soit f un réseau de neurone défini par *perceptron* (page 26) composé de :

— une couche d'entrées

— une couche cachée dont les fonctions de transfert sont sigmoïdes

— une couche de sortie dont les fonctions de transfert sont linéaires

---

Ce réseau sert de modèle pour la fonction f dans le problème de *régression* (page 28) avec un échantillon $((X_1, Y_1), \ldots, (X_N, Y_N))$, les résidus sont supposés normaux. La suite $(\widehat{\epsilon_k})$ définie par *(2)* (page **??**) vérifie :

$$\frac{1}{N}\sum_{i=1}^{N}\widehat{\epsilon_k} = 0 = \mathbb{E}\left[f\left(\widehat{W}, X\right) - Y\right]$$

Et le vecteur aléatoire $\widehat{W} - W^*$ vérifie :

$$\sqrt{N}\left[\widehat{W} - W^*\right] \overset{T\to+\infty}{\longrightarrow} \mathcal{N}\left(0, \widehat{\sigma_N}^2\widehat{\Sigma_N}\right)$$

Où la matrice $\widehat{\Sigma_N}$ est définie par *(2)* (page **??**).

end{xtheorem}

---

entrée originale (page 62)

Théorème T1 : résolution de l'ACP

Les notations utilisées sont celles du problème de l¿*ACP* (page 272). Dans ce cas :

$$S = \underset{\substack{W\in M_{p,d}(\mathbb{R})\\ W'W=I_d}}{\arg\max}\left[\sum_{i=1}^{N}\left\|W'X_i\right\|^2\right] = \underset{W\in M_{p,d}(\mathbb{R})}{\arg\min}\left[\sum_{i=1}^{N}\left\|WW'X_i - X_i\right\|^2\right]$$

De plus S est l'espace vectoriel engendré par les d vecteurs propres de la matrice $XX' = \sum_{i=1}^{N} X_iX_i'$ associées aux d valeurs propres de plus grand module.

---

entrée originale (page 69)

Théorème T1 : résolution du problème du maximum de vraisemblance

La solution du problème du *maximum de vraisemblance* (page 273) est le vecteur :

$$(p_1^*, \ldots, p_N^*) = (d_1, \ldots, d_N)$$

---

entrée originale (page 55)

Théorème T1 : simulation d'une loi quelconque

Soit $F = \int f$ une fonction de répartition de densité f vérifiant $f > 0$, soit U une variable aléatoire uniformément distribuée sur $[0, 1]$ alors $F^{-1}(U)$ est variable aléatoire de densité f.

---

entrée originale (page 219)

Théorème T2 : Borne supérieure de l'erreur produite par k-means++

On définit l'inertie par $J_(X) = \sum_{i=1}^{P} \min_G d^2(X_i, G)$. Si J_{OPT} définit l'inertie optimale alors $\mathbb{E}J(X) \leqslant 8(\ln C + 2)J_{OPT}(X)$.

---

entrée originale (page 17)

Théorème T2 : [Farago1993]_ 2

Les notations sont celles du même algorithme. On définit une mesure sur l'ensemble X, $B(x,r)$ désigne la boule de centre x et de rayon r, $Z \in X$ une variable aléatoire, de plus :

$$p(x,r) = P_X(B(x,r)) = \mathbb{P}(Z \in B(x,r))$$

On suppose qu'il existe $d > 0$ et une fonction $f : X \longrightarrow \mathbb{R}$ tels que :

$$\lim_{r \to 0} \frac{p(x,r)}{r^d} = f(x) > 0$$

La convergence doit être uniforme et presque sûre. On note également F_N le nombre de calculs de dissimilarité effectués par l'algorithme où N est le nombre d'élément de E, P désigne toujours le nombre de pivots, alors :

$$\limsup_{n \to \infty} \mathbb{E}F_N \leqslant k + \left(\frac{\alpha}{\beta}\right)^{2d}$$

---

entrée originale (page 76)

Théorème T2 : rétropropagation

Cet algorithme s'applique à un réseau de neurones vérifiant la définition du *perceptron* (page 26). Il s'agit de calculer sa dérivée par rapport aux poids. Il se déduit des formules *(3)* (page **??**), *(4)* (page **??**), *(5)* (page **??**) et *(7)* (page **??**) et suppose que l'algorithme de *propagation* (page 27) a été préalablement exécuté. On note $y'_{c,i} = \frac{\partial e}{\partial y_{c,i}}$, $w'_{c,i,j} = \frac{\partial e}{\partial w_{c,i,j}}$ et $b'_{c,i} = \frac{\partial e}{\partial b_{c,i}}$.

Initialisation

for i in $1..C_C$
    $y'_{C,i} \longleftarrow \frac{\partial e}{\partial z_{C,i}}(W,X)\, f'_{c,i}(y_{C,i})$

Récurrence

for c in $1..C-1$
    for i in $1..C_c$
        $y'_{c,i} \longleftarrow 0$
        for j in $1..C_{c+1}$
            $y'_{c,i} \longleftarrow y'_{c,i} + y'_{c+1,j}\, w_{c+1,j,i}$
        $y'_{c,i} \longleftarrow y'_{c,i}\, f'_{c,i}\left(y'_{c,i}\right)$

Terminaison

for c in $1..C$
    for i in $1..C_c$
        for j in $1..C_{c-1}$
            $w'_{c,i,j} \longleftarrow z_{c-1,j}\, y'_{c,i}$
            $b'_{c,i,j} \longleftarrow y'_{c,i}$

---

entrée originale (page 44)

Théorème T2 : simulation d'une loi de Poisson

On définit une suite infinie $(X_i)_i > 0$ de loi exponentielle de paramètre λ. On définit ensuite la série de variables aléatoires $S_i = \sum_{k=1}^{i} X_k$ et enfin $N(t) = \inf\{i | S_i > t\}$. Alors la variable aléatoire $N(t)$ suit une loi de Poisson de paramètre λt.

entrée originale (page 219)

Théorème T2 : sélection d'architecture

Les notations utilisées sont celles du théorème *loi asymptotique des coefficients* (page 275). f est un réseau de neurones de paramètres W. On définit la constante τ, en général $\tau = 3,84$ puisque $\mathbb{P}(X < \tau) = 0,95$ si $X \sim \chi^2_1$.

Initialisation

Une architecture est choisie pour le réseau de neurones f incluant un nombre *M* de paramètres.

Apprentissage

Le réseau de neurones f est appris. On calcule les nombre et matrice $\widehat{\sigma_N}^2$ et $\widehat{\Sigma_N}$. La base d'apprentissage contient N exemples.

Test

for k in $1..M$

$$t_k \longleftarrow N \frac{\widehat{w_k}^2}{\widehat{\sigma_N}^2 \left(\widehat{\Sigma_N}^{-1}\right)_{kk}}$$

Sélection

$k' \longleftarrow \arg\min_k t_k$

si $t_{k'} < \tau$

Le modèle obtenu est supposé être le modèle optimal. L'algorithme s'arrête.

sinon

La connexion k' est supprimée ou le poids $w_{k'}$ est maintenue à zéro.

$M \longleftarrow M - 1$

Retour à l'apprentissage.

entrée originale (page 62)

Théorème T3 : somme de loi exponentielle iid

Soit $X_1, ..., X_n$ n variables aléatoires indépendantes et identiquement distribuées de loi $Exp(\lambda)$ alors la somme $\sum_{k=1}^{n} X_k$ suit une loi $Gamma(n, \lambda)$.

entrée originale (page 219)

Bugs et améliorations

Sommaire

Bugs

0 items

Amélioration

0 items

Terminé

0 items

Changes

List of recent changes :

#	change number	date	author	comment
407	1c32afe [183]	2018-10-20	xavier dupré	use of xgboost
406	446e682 [184]	2018-10-15	xavier dupré	update unit tests
405	de6e64c [185]	2018-10-15	xavier dupré	Fix unit test aftrer scikit-learn update
404	3c76576 [186]	2018-10-15	xavier dupré	update notebooks
403	80ec86f [187]	2018-09-23	xavier dupré	release one condition for a unit test
402	714ddad [188]	2018-09-23	xavier dupré	update CI, fix notebook for matplotlib 3.0
401	3703ccc [189]	2018-09-17	xavier dupré	update circleci
400	81ba916 [190]	2018-09-12	xavier dupré	blog post
399	5a35e41 [191]	2018-09-09	xavier dupré	minor fixes
398	55d9247 [192]	2018-09-05	xavier dupré	fix setup.py
397	b481314 [193]	2018-08-28	xavier dupré	add missing dependencies for circleci
396	7df33e6 [194]	2018-08-26	xavier dupré	update circleci
395	505a9d8 [195]	2018-08-22	xavier dupré	update local jenkins linux
394	7aa2fa0 [196]	2018-08-21	xavier dupré	add mlsinghts as a dependency
393	40e05f9 [197]	2018-08-21	xavier dupré	update dependencies on local builds

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
392	d05417f [198]	2018-08-21	xavier dupré	fix missing imports
391	5b031cc [199]	2018-08-21	xavier dupré	add local job jenkins linux
390	bbe53e5 [200]	2018-08-18	xavier dupré	update dependencies
389	e47673d [201]	2018-07-23	xavier dupré	back to python 3.6 for appveyor, simplification
388	8115bda [202]	2018-07-22	xavier dupré	fix style
387	e7a73bc [203]	2018-07-22	xavier dupré	remove inheritance from object
386	d1195b4 [204]	2018-07-22	xavier dupré	back to python 3.6 (line_profiler not available)
385	c9a09e1 [205]	2018-07-22	xavier dupré	update requirements
384	e54bf0d [206]	2018-07-22	xavier dupré	update circleci about statsmodels
383	55c826c [207]	2018-07-22	xavier dupré	update CI python 3.7
382	3fd2e23 [208]	2018-07-07	xavier dupré	update pandoc
381	728af23 [209]	2018-07-03	xavier dupré	documentation, fix a bug in roc.py
380	cd731c8 [210]	2018-06-25	xavier dupré	remove conda from automated build
379	fd68b07 [211]	2018-06-18	xavier dupré	fix unit test
378	54c62dd [212]	2018-06-18	xavier dupré	update requirements
377	2bcf256 [213]	2018-06-18	xavier dupré	more on quantile regression
376	8d5c5e2 [214]	2018-06-18	xavier dupré	start explaining quantile regression
375	58c989b [215]	2018-05-17	xavier dupré	fix unit test on ROC, add better message #13
374	e4843de [216]	2018-05-17	xavier dupré	minor fixes in setup, documentation, circleci
373	0b87bcd [217]	2018-05-12	xavier dupré	update code style
372	0728951 [218]	2018-05-12	xavier dupré	remove unnecessary import
371	2355d97 [219]	2018-05-12	xavier dupré	update for pylint
370	bea10fb [220]	2018-05-12	xavier dupré	remove warning on one import
369	40bdcc6 [221]	2018-05-12	xavier dupré	update for pylint
368	59998a9 [222]	2018-05-10	xavier dupré	fix unused variable
367	eb44d27 [223]	2018-05-08	xavier dupré	update requirements
366	2b2d6f1 [224]	2018-05-08	xavier dupré	inference of voronoi diagram #12
365	346d9de [225]	2018-05-07	xavier dupré	add voronoi inference #12
364	e11cf53 [226]	2018-05-07	xavier dupré	logreg et voronoï

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
363	c75eb22 [227]	2018-05-04	xavier dupré	voronoi
362	40bd586 [228]	2018-05-03	xavier dupré	continues on voronoi
361	c637f30 [229]	2018-05-02	xavier dupré	update latex, remove new commands
360	6e73062 [230]	2018-05-01	xavier dupré	update requirements
359	5cf2f7a [231]	2018-05-01	xavier dupré	update history
358	d1f99e6 [232]	2018-05-01	xavier dupré	complete study on voronoï and logistic regression
357	3d6076c [233]	2018-04-30	xavier dupré	voronoi, log reg
356	6601fb6 [234]	2018-04-28	xavier dupré	update unit test
355	03dd7fc [235]	2018-04-27	xavier dupré	e few modifications
354	3eff2bf [236]	2018-04-23	xavier dupré	voronoi #11
353	e5d0df7 [237]	2018-04-19	xavier dupré	biblio
352	083fec0 [238]	2018-04-18	xavier dupré	change and fix unit test
351	f0e8753 [239]	2018-04-18	xavier dupré	series of fixes for #10, segment detection
350	1c366a0 [240]	2018-04-18	xavier dupré	update requirement
349	6a4c0af [241]	2018-04-18	xavier dupré	segment detection #10
348	0ad2a13 [242]	2018-04-17	xavier dupré	fix gradient for detect_segments
347	2e1cb0f [243]	2018-04-17	xavier dupré	requirements opencv
346	5f87945 [244]	2018-04-17	xavier dupré	update requirements
345	3bdb45e [245]	2018-04-17	xavier dupré	add code for #10, build an image is left
344	055758a [246]	2018-04-14	xavier dupré	update requirements
343	63f0c68 [247]	2018-04-14	xavier dupré	replace flake8 by code_style
342	aaa2b4b [248]	2018-04-01	xavier dupré	remove unexisting function
341	a5f6f40 [249]	2018-04-01	xavier dupré	history, links, issue #9, fix unittest on wikipedia_dump
340	4836356 [250]	2018-04-01	xavier dupré	fix url
339	0b85e3b [251]	2018-03-19	xavier dupré	update conf.py
338	8b56312 [252]	2018-03-10	xavier dupré	update setup.py
337	2dad004 [253]	2018-02-25	xavier dupré	replace png by jpg for latex conversion
336	8a91d2a [254]	2018-02-25	xavier dupré	update notebook
335	9fe5e1a [255]	2018-02-24	xavier dupré	fix wikipedia download
334	af2ea57 [256]	2018-02-24	xavier dupré	update requirements

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
333	ea03d57 [257]	2018-02-24	xavier dupré	fix a test failing due to pyquickhelper update
332	f212630 [258]	2018-02-24	xavier dupré	requirements setup.py
331	25ac176 [259]	2018-02-17	xavier dupré	update requirements
330	f50c382 [260]	2018-02-07	xavier dupré	update ROC
328	6dd5e25 [261]	2018-02-03	xavier dupré	fix image path
327	0bb5b03 [262]	2018-02-03	xavier dupré	fix image path
326	9a07d49 [263]	2018-02-03	xavier dupré	add link to written documentation
325	c1800c8 [264]	2018-01-31	xavier dupré	remov readme from latex
324	9439785 [265]	2018-01-19	xavier dupré	update notebook, orthograph
323	02524be [266]	2018-01-02	xavier dupré	update license
322	7ca5fc3 [267]	2017-12-17	xavier dupré	disable one cell in a unit test of a notebook
321	f856ba3 [268]	2017-12-16	xavier dupré	fix notebook for matplotlib 2.1.1
320	f44b47d [269]	2017-12-09	xavier dupré	un petit peu plus sur la complétion
319	311c979 [270]	2017-12-01	xavier dupré	update script
318	b081b22 [271]	2017-11-29	xavier dupré	update script
317	7d825ca [272]	2017-11-27	xavier dupré	update notebook and unit test
316	274c505 [273]	2017-11-27	xavier dupré	update notebook
315	c67f3aa [274]	2017-11-27	xavier dupré	add notebook on linear regression
313	918c4ac [275]	2017-11-19	xavier dupré	update requirements
312	79f76d5 [276]	2017-11-18	xavier dupré	add artifacts on appveyor
311	2586501 [277]	2017-11-16	xavier dupré	extend notebook coverage
310	8fe5d58 [278]	2017-11-12	xavier dupré	update python version
309	851ef49 [279]	2017-11-06	xavier dupré	p-values
307	b741325 [280]	2017-10-20	xavier dupré	add notebook number due to notebook 5.1.0
306	021d334 [281]	2017-10-19	xavier dupré	appveyor
305	de11f38 [282]	2017-10-18	xavier dupré	update ROC
304	f461a09 [283]	2017-10-18	xavier dupré	add image
303	1bda5b8 [284]	2017-10-17	xavier dupré	fixes
301	0ea8e00 [285]	2017-09-17	xavier dupré	add history in setup.py
300	ca05df8 [286]	2017-09-09	xavier dupré	fix a few link
299	9fad562 [287]	2017-09-08	xavier dupré	extend coverage + fix internal links

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
298	4289732 [288]	2017-09-08	xavier dupré	add section on knn
297	8923b89 [289]	2017-09-07	xavier dupré	add missing image
296	aeb80ce [290]	2017-09-07	xavier dupré	fix image name
295	3375f01 [291]	2017-09-07	xavier dupré	update requirements
294	749b622 [292]	2017-09-07	xavier dupré	fix unit test
293	1bc0c2e [293]	2017-09-07	xavier dupré	add strategie_alea
292	aa29981 [294]	2017-09-05	xavier dupré	update requirements
291	d5c46ef [295]	2017-09-04	xavier dupré	fix missing command
290	159dbd8 [296]	2017-09-04	xavier dupré	fix formulas
289	07d0397 [297]	2017-09-04	xavier dupré	update pvalues, fix notebook
288	1cb2150 [298]	2017-09-03	xavier dupré	first draft for p-values
287	968cd01 [299]	2017-09-02	xavier dupré	add missing import
286	07aa0e3 [300]	2017-09-02	xavier dupré	reduce requirements + disable one unit test on travis
285	e1a093c [301]	2017-09-02	xavier dupré	requirements
284	926125f [302]	2017-09-02	xavier dupré	update requirements
283	cf132a3 [303]	2017-09-02	xavier dupré	update requirements
282	3963916 [304]	2017-09-01	xavier dupré	increase notebook coverage
281	199fa2e [305]	2017-08-31	xavier dupré	update requirements
280	3cd6e68 [306]	2017-08-31	xavier dupré	pep8
279	add42ed [307]	2017-08-31	xavier dupré	update documentation and unit test
277	ee31a1d [308]	2017-08-31	xavier dupré	fix a bug in mf, file d'attente
276	bef17f1 [309]	2017-08-28	xavier dupré	fix appveyor build
274	d6f1ca5 [310]	2017-08-26	xavier dupré	update build
273	809ce7a [311]	2017-08-25	xavier dupré	fix travis
272	2454aa3 [312]	2017-08-25	xavier dupré	fix appveyor
271	191f08c [313]	2017-08-24	xavier dupré	use minepy, compute MIC
270	502a99b [314]	2017-08-23	xavier dupré	fix latex build
269	4e2910f [315]	2017-08-23	xavier dupré	update requirements
268	cb22e2d [316]	2017-08-23	xavier dupré	fix documentation
267	6eeff5e [317]	2017-08-23	xavier dupré	requirements
266	82e3e23 [318]	2017-08-23	xavier dupré	requirements
265	ea572df [319]	2017-08-23	xavier dupré	requirements
264	c44aba4 [320]	2017-08-22	xavier dupré	pep8
263	c8b1f35 [321]	2017-08-22	xavier dupré	fix kernel name
262	74adf50 [322]	2017-08-22	xavier dupré	fix misspelling
261	f57e4f8 [323]	2017-08-22	xavier dupré	fix bad import + travis
260	f8718c9 [324]	2017-08-22	xavier dupré	fix requirements + run_notebooks
259	b424afc [325]	2017-08-22	xavier dupré	update requirements
258	ac716c1 [326]	2017-08-22	xavier dupré	add dependency

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
257	7d604a0 [327]	2017-08-22	xavier dupré	remove datashaer
256	ed9ced7 [328]	2017-08-22	xavier dupré	minimze requirements
255	a0c7b3c [329]	2017-08-22	xavier dupré	circleci
253	b19ea91 [330]	2017-08-22	xavier dupré	pep8
252	6940eef [331]	2017-08-22	xavier dupré	add reference
251	08012e4 [332]	2017-08-22	xavier dupré	move graph_distance from ensae_teaching_cs to mlstatpy
248	06773f0 [333]	2017-08-15	xavier dupré	pep8
247	848f05f [334]	2017-08-15	xavier dupré	fix appveyor dependencies
246	8dd303d [335]	2017-08-15	xavier dupré	update notebook unit test
244	d1360d0 [336]	2017-08-08	xavier dupré	fix code included in the documentation
243	7bd59f4 [337]	2017-08-08	xavier dupré	licence
242	e815702 [338]	2017-07-26	xavier dupré	update appveyor
241	30ab406 [339]	2017-07-26	xavier dupré	nmf and pca, notebooks
240	c3c7c9d [340]	2017-07-25	xavier dupré	update mf
239	4b16c5e [341]	2017-07-20	xavier dupré	utf-8
238	982b545 [342]	2017-07-20	xavier dupré	update notebook mf
237	be1cb09 [343]	2017-07-20	xavier dupré	beginning of matrix factorization
236	ab861a0 [344]	2017-07-20	xavier dupré	renaming a file
235	352b880 [345]	2017-07-20	xavier dupré	api + missing data beginning
234	7015a10 [346]	2017-07-20	xavier dupré	documentartion API + refactoring
233	0528674 [347]	2017-07-01	xavier dupré	stop testing again 3.5
232	365e051 [348]	2017-07-01	xavier dupré	update dependencies for travis
231	43de2fa [349]	2017-07-01	xavier dupré	enable 3.6 on travis
230	06e187b [350]	2017-06-30	xavier dupré	update setup.py
229	e73d17d [351]	2017-06-29	xavier dupré	update setup.py
228	0f4c522 [352]	2017-06-29	xavier dupré	35 to 36
227	3ce7be2 [353]	2017-06-17	xavier dupré	fix missing function
226	bb02c80 [354]	2017-06-16	xavier dupré	make a fix for matplotlib
225	71a0de2 [355]	2017-06-11	xavier dupré	update documentation
224	8932a9b [356]	2017-06-11	xavier dupré	update jenkins build
223	e04af41 [357]	2017-05-23	xavier dupré	fix path in jenkins job definition
222	a866bbc [358]	2017-05-22	xavier dupré	update setup.py

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
221	435e6d4 [359]	2017-05-22	xavier dupré	update jenkins_build
220	4c1c119 [360]	2017-05-21	xavier dupré	update setup.py
219	d94fc69 [361]	2017-05-21	xavier dupré	remove .ix
218	07bbbb8 [362]	2017-05-08	xavier dupré	update license
217	fe5e59e [363]	2017-05-01	xavier dupré	add reference
216	d1839fb [364]	2017-04-22	xavier dupré	documentation configuration
215	7482cc1 [365]	2017-04-22	xavier dupré	documentation configuration
214	d2d8cd3 [366]	2017-04-21	xavier dupré	update configuration
213	9e9ea91 [367]	2017-04-20	xavier dupré	update configuration
212	b5d340f [368]	2017-04-18	xavier dupré	update documentation configuration
211	442d23f [369]	2017-04-11	xavier dupré	update formulas
210	32fbdbd [370]	2017-04-11	xavier dupré	pep8
209	8156a28 [371]	2017-04-11	xavier dupré	add a notebook on classification
208	002bdbf [372]	2017-04-01	xavier dupré	update noteebook
207	62243e5 [373]	2017-03-30	xavier dupré	udpate notebook a little
206	2954b87 [374]	2017-03-29	xavier dupré	change title
205	0e1a37c [375]	2017-03-28	xavier dupré	update style
204	d6d3e9f [376]	2017-03-22	xavier dupré	update appveyor
203	979d0fc [377]	2017-03-21	xavier dupré	more logging for a unit test
202	6723c13 [378]	2017-03-20	xavier dupré	fix pep8
201	8a63ad4 [379]	2017-03-20	xavier dupré	update notebook to catch an exception
200	7ec24f7 [380]	2017-03-20	xavier dupré	add more logging
198	40f91c0 [381]	2017-03-19	xavier dupré	update requirements
197	3e8e1f2 [382]	2017-03-19	xavier dupré	requirements
196	cc455b0 [383]	2017-03-19	xavier dupré	benchmark for machine learn models
195	afc81b5 [384]	2017-03-18	xavier dupré	reduce the number of files for requirements
194	786b4ef [385]	2017-03-18	xavier dupré	fix requirements
193	532e026 [386]	2017-03-18	xavier dupré	update configuration
192	0d57cc5 [387]	2017-03-18	xavier dupré	update documentation
191	7b6b106 [388]	2017-03-11	xavier dupré	update script
190	28c9f65 [389]	2017-02-23	xavier dupré	jenkins and documentation
189	445f008 [390]	2017-02-22	xavier dupré	better support of matplotlib when unit testing

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
188	8e8b6b9 [391]	2017-02-17	xavier dupré	update jenkins anaconda
187	fb1e7ee [392]	2017-02-16	xavier dupré	update jenkins winpython
186	4ebee42 [393]	2017-02-16	xavier dupré	fix encoding
185	3e248f2 [394]	2017-02-16	xavier dupré	blog post
184	5d2567e [395]	2017-02-13	xavier dupré	inttroduction
183	b2bd06b [396]	2017-02-12	xavier dupré	fix a couple of issues
182	d4a7553 [397]	2017-02-12	xavier dupré	ravis
181	657e404 [398]	2017-02-12	xavier dupré	update appveyor
180	59216f5 [399]	2017-02-12	xavier dupré	change documentation style + a few fixes
179	535d099 [400]	2017-02-09	xavier dupré	update notebook
178	7866c5f [401]	2017-02-07	xavier dupré	pep8
177	816564a [402]	2017-02-07	xavier dupré	add unit test to test peregrination
176	b6fe252 [403]	2017-02-05	xavier dupré	notebook pérégrinations
175	fd7d95e [404]	2017-02-05	xavier dupré	notebooks and peregrination
174	41364e8 [405]	2017-01-17	xavier dupré	remove python 3.6
173	f22c92f [406]	2017-01-17	xavier dupré	update travis build
172	e125000 [407]	2017-01-17	xavier dupré	update travis
171	7a991d0 [408]	2017-01-15	xavier dupré	update notebook
170	b5ab4dc [409]	2017-01-07	xavier dupré	update dependencies
169	83cb57e [410]	2017-01-07	xavier dupré	fix an encoding issue
168	a0d607f [411]	2017-01-06	xavier dupré	update requirements for travis
167	04c6c9e [412]	2017-01-06	xavier dupré	update local jenkins
166	858ccb6 [413]	2017-01-03	xavier dupré	update jenkins build
165	a0d511d [414]	2017-01-01	xavier dupré	add python 3.6 for ci
164	d44a5d6 [415]	2016-12-30	xavier dupré	documentation and styles
163	7c8cc0d [416]	2016-12-29	xavier dupré	rst cleaning
161	bd0f386 [417]	2016-12-26	xavier dupré	update documentation
159	087b6d8 [418]	2016-12-21	xavier dupré	documentation and updates notebook for galleries
158	8f5cc56 [419]	2016-12-20	xavier dupré	update notebooks on ROC + code
157	bd95594 [420]	2016-12-18	xavier dupré	update gitignore
155	d2cd205 [421]	2016-12-14	xavier dupré	fix bug documentation
154	a9c4d0e [422]	2016-12-13	xavier dupré	roc
153	4a0e100 [423]	2016-12-10	xavier dupré	citations

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
151	0d78919 [424]	2016-12-10	xavier dupré	biblio
150	73dba1a [425]	2016-12-10	xavier dupré	update roc curve
149	de8e462 [426]	2016-12-09	xavier dupré	roc intermediate steps
148	0d312a7 [427]	2016-12-03	xavier dupré	remove download badge
147	9b947b9 [428]	2016-11-19	xavier dupré	typo
146	0e8bb15 [429]	2016-11-11	xavier dupré	utf8
145	0925878 [430]	2016-11-11	xavier dupré	kmenas
144	487319d [431]	2016-11-05	dupre	latex
143	df98dc9 [432]	2016-11-05	dupre	latex
142	5b771fe [433]	2016-11-05	dupre	pep8
141	544641c [434]	2016-11-05	dupre	add edit distance
140	b70e00e [435]	2016-10-24	dupre	add link
139	9cdeaa3 [436]	2016-10-24	dupre	fix latex
138	91f0045 [437]	2016-10-24	dupre	fix latex
137	006cac5 [438]	2016-10-24	dupre	latex fix
136	b230570 [439]	2016-10-24	dupre	latex issue
135	d941143 [440]	2016-10-23	dupre	latex fix
134	4463d92 [441]	2016-10-23	dupre	latex fix
133	af079b7 [442]	2016-10-23	dupre	fix latex
132	9415823 [443]	2016-10-23	dupre	fix documentation
131	23e440a [444]	2016-10-23	dupre	latex fix
130	4f0a424 [445]	2016-10-23	dupre	udpate conf
129	ca42f1b [446]	2016-10-23	dupre	update conf.py
128	2e8299c [447]	2016-10-23	dupre	clustering
127	f28aa32 [448]	2016-10-22	dupre	avoid uploading with a wrong subversion number
126	9b6b43d [449]	2016-10-17	dupre	update ROC
125	c5c9860 [450]	2016-10-10	dupre	typo
124	dded8c6 [451]	2016-09-25	dupre	update appveyor
123	272d8fd [452]	2016-09-25	dupre	update requirements
122	c3aa84d [453]	2016-09-25	dupre	update notebook menu
121	0a17641 [454]	2016-09-23	dupre	update travis
120	e990aa2 [455]	2016-09-23	dupre	update notebook
119	e9e712d [456]	2016-09-18	dupre	remove file
118	6ef1ae2 [457]	2016-09-18	dupre	clean unicode characters from a notebook
117	91d99ff [458]	2016-09-18	dupre	remove weird characters
116	269352c [459]	2016-09-15	dupre	update metric
115	aa5d20e [460]	2016-09-14	dupre	fix issue with latex
114	a8b5cde [461]	2016-09-11	dupre	fix wikipedia dumps
113	d3498c5 [462]	2016-09-11	dupre	update function to download wikipedia dump

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
112	109b4e0 [463]	2016-09-11	dupre	implement the metric on a test set of queries
111	10697ae [464]	2016-09-09	dupre	add link
110	20bf80d [465]	2016-09-05	dupre	typo
109	cfcc586 [466]	2016-09-05	dupre	sphinx typo
107	322191d [467]	2016-09-04	dupre	split completion into multiple documents
106	36b89ec [468]	2016-09-04	dupre	pep8
105	d1dec17 [469]	2016-09-04	dupre	fix unit tests
103	21d7f72 [470]	2016-09-04	dupre	rename function name
102	4eb5848 [471]	2016-09-04	dupre	update notebooks and completion
101	0984b14 [472]	2016-09-03	dupre	update local jenkins
100	5b2f0ec [473]	2016-09-03	dupre	idée
99	5cee7aa [474]	2016-09-02	dupre	add more unit test
98	1e6078c [475]	2016-09-01	dupre	update local jenkins
97	383ae8f [476]	2016-08-31	dupre	segment_detection beginning
96	f632171 [477]	2016-08-31	dupre	update local jenkins
95	ac5365b [478]	2016-08-31	dupre	todo
94	5ed936d [479]	2016-08-31	dupre	fix the elegant algorithm
93	793eaaa [480]	2016-08-30	dupre	fix computation of mks
92	c573353 [481]	2016-08-29	dupre	work in progress
91	8057956 [482]	2016-08-28	dupre	update appveyor
90	6f69a2a [483]	2016-08-28	dupre	updates about completion
89	7c94bfd [484]	2016-08-28	dupre	update local jenkins
88	a137626 [485]	2016-08-28	dupre	update local jenkins
87	8d7c007 [486]	2016-08-27	dupre	update locale jenkins
86	5400677 [487]	2016-08-27	dupre	fixes
85	f3b1ff7 [488]	2016-08-27	dupre	update local.jenkins
84	faad343 [489]	2016-08-26	dupre	update jenkins definition
83	328fea7 [490]	2016-08-26	dupre	update jenkins build
82	1fd17b0 [491]	2016-08-25	dupre	pep8, remove unnecessary import
81	143dd70 [492]	2016-08-24	dupre	update setup.py
80	53827f4 [493]	2016-08-22	dupre	update notebooks
79	e1f001c [494]	2016-08-21	dupre	update notebooks
78	0ffd5fa [495]	2016-08-20	dupre	remarque sur les lignes de commandes
77	7e1a3b4 [496]	2016-08-17	dupre	blog post
75	f2791ad [497]	2016-08-13	dupre	pep8

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
74	b676f9b [498]	2016-08-13	dupre	est complet, bug fix
73	d1e3da1 [499]	2016-08-12	dupre	typo, correction latex
72	3e49ce7 [500]	2016-08-10	dupre	still a bug in mlstatpy, adding a unit to catch it
71	0a26d76 [501]	2016-08-08	dupre	pep8
69	7576220 [502]	2016-08-04	dupre	compression
68	3c6e4b7 [503]	2016-08-03	dupre	update notebook, travis, appveyor
67	9303c2d [504]	2016-08-03	dupre	update appveyor and notebook on profiling
66	e81b42f [505]	2016-08-03	dupre	fix unit test
64	2e68c22 [506]	2016-08-03	dupre	fix travis build
63	173aaa1 [507]	2016-08-03	dupre	update travis + fix a bug __iter__ (order of vertices)
62	56f5140 [508]	2016-08-03	dupre	update notebooks
61	fb64be1 [509]	2016-08-03	dupre	notebook about profiling
60	d66847a [510]	2016-08-02	dupre	completion
59	f8d2e56 [511]	2016-08-02	dupre	pep8
57	354bb24 [512]	2016-08-02	dupre	rename suggestion into completion
56	9b15d97 [513]	2016-08-02	dupre	reduce duration of a unit test
55	07dcd7b [514]	2016-08-02	dupre	fix notebook
54	2f8c515 [515]	2016-08-02	dupre	fix computation of MKS
53	da53941 [516]	2016-08-02	dupre	examples for unit tests
52	54d4085 [517]	2016-08-02	dupre	complétion théorie
51	ad44caa [518]	2016-08-01	dupre	update completion
50	255721a [519]	2016-07-26	dupre	fix unittests
49	87c0e70 [520]	2016-07-26	dupre	update setup.py
48	15e8b88 [521]	2016-07-26	dupre	fix unit test
47	c0b5998 [522]	2016-07-24	dupre	update completion
46	b79965b [523]	2016-07-23	dupre	add function to download wikipedia titles
45	a3cd811 [524]	2016-07-23	dupre	pep8
44	64ba3a4 [525]	2016-07-23	dupre	add a function to download wikipedia pagecount
43	3725884 [526]	2016-07-22	dupre	pep8
42	5e859b1 [527]	2016-07-22	dupre	update completion
41	ad8e3a5 [528]	2016-07-20	dupre	todo
40	2e7dcf4 [529]	2016-07-20	dupre	update links and equations

Suite sur la page suivante

Tableau 1 – suite de la page précédente

#	change number	date	author	comment
39	2b2a2f4 [530]	2016-07-20	dupre	moving documentation
38	f2b8cf0 [531]	2016-07-19	dupre	pep8
37	7305b29 [532]	2016-07-19	dupre	roc, notebooks
36	cded6ef [533]	2016-07-18	dupre	ROC curve
35	7bb8437 [534]	2016-07-17	dupre	update nlp
33	82f6cc7 [535]	2016-07-16	dupre	dynamic minimum
32	01faa12 [536]	2016-07-16	dupre	nlp
31	4b452c4 [537]	2016-07-15	dupre	réseaux de neurones, complétion
30	7a03d19 [538]	2016-07-10	dupre	update appveyor
29	ecf18f9 [539]	2016-07-09	dupre	update conf.py
28	1f80ff7 [540]	2016-07-05	dupre	fix labels
27	810bb80 [541]	2016-07-03	dupre	fix rn
26	ad1d2c4 [542]	2016-07-03	dupre	rn
25	029b746 [543]	2016-07-01	dupre	update rn
24	1fa8e59 [544]	2016-06-30	dupre	update rn
23	9a76975 [545]	2016-06-30	dupre	intro + newton
22	ab7fccf [546]	2016-06-29	dupre	update gitignore and style
21	cd94a5c [547]	2016-06-29	dupre	réseaux de neurons + todos
20	b9ae7af [548]	2016-06-28	dupre	update demo
19	270a80d [549]	2016-06-27	dupre	quick fix
18	0d4ea39 [550]	2016-06-27	dupre	update on neural network
17	221a6bd [551]	2016-06-25	dupre	calcul matriciel
16	b03c680 [552]	2016-06-25	dupre	update travis
15	afdae4b [553]	2016-06-25	dupre	update travis
14	21809ec [554]	2016-06-25	dupre	update style
13	eab10f1 [555]	2016-06-25	dupre	style + mkl
12	0d77465 [556]	2016-06-25	dupre	update margins
11	0ed6de3 [557]	2016-06-25	dupre	update style for lineblock + travis
10	729c0d4 [558]	2016-06-25	dupre	travis mkl
9	7eef7d4 [559]	2016-06-25	dupre	a little bit more
8	7fb23b6 [560]	2016-06-24	dupre	links and appveyor
7	a975817 [561]	2016-06-22	dupre	add a notebook to test and fix the test
6	7f9e63e [562]	2016-06-22	dupre	notebooks
5	e1f1940 [563]	2016-06-22	dupre	update rn
4	6a276a0 [564]	2016-06-19	dupre	update codecov
3	70391c3 [565]	2016-06-19	dupre	update setup.py
2	bcfc7f0 [566]	2016-06-19	dupre	second version, empty shelve
1	c58f261 [567]	2016-06-18	dupre	first commit

183. https://github.com/sdpython/mlstatpy/commit/1c32afe545fb82575a80a7209743f4f12a2fb332
184. https://github.com/sdpython/mlstatpy/commit/446e682ec5a776cd02c678e80bff461d3c5e10dc
185. https://github.com/sdpython/mlstatpy/commit/de6e64c1a853c2558314739df4a213c79736f9fd
186. https://github.com/sdpython/mlstatpy/commit/3c765763c3b8a5c8802b4803b64b648a69dd0061
187. https://github.com/sdpython/mlstatpy/commit/80ec86f8665aa2f5fe43e1c0899aacb007661923
188. https://github.com/sdpython/mlstatpy/commit/714ddad5b92f2d20f3e2ca3034602575d5efe994
189. https://github.com/sdpython/mlstatpy/commit/3703cccc8969d857a62ac88579e265625dcb236a
190. https://github.com/sdpython/mlstatpy/commit/81ba916e3af826db5956bba865497a2c386ce114
191. https://github.com/sdpython/mlstatpy/commit/5a35e41f2a4420b1e7b64e29de0b70a1a2f69592
192. https://github.com/sdpython/mlstatpy/commit/55d924768c75fdf347477428fc5e87c018a2a575
193. https://github.com/sdpython/mlstatpy/commit/b48131408734b624108694779ba36adcc1e24b93
194. https://github.com/sdpython/mlstatpy/commit/7df33e621b3a2f914f01e3dfe309298d0daa592c
195. https://github.com/sdpython/mlstatpy/commit/505a9d8008df1d2cd0e145a3faf4276d1f49c68c
196. https://github.com/sdpython/mlstatpy/commit/7aa2fa06e8a1102b1d328684b1fe691ed6037d96
197. https://github.com/sdpython/mlstatpy/commit/40e05f997b32fc3bc1e9f36fbb3e12c7ac45dc8d
198. https://github.com/sdpython/mlstatpy/commit/d05417f971fed8e143bfd639417a2f4a2372739c
199. https://github.com/sdpython/mlstatpy/commit/5b031cc2eac0985e7eefc1007b7f107831c32c62
200. https://github.com/sdpython/mlstatpy/commit/bbe53e5d55ac975269bfeccb62aa5652a4327e6d
201. https://github.com/sdpython/mlstatpy/commit/e47673d74eb68a786285a7250bad52b33f330e2d
202. https://github.com/sdpython/mlstatpy/commit/8115bda4a69dbba08b48a0746f0842410ec89867
203. https://github.com/sdpython/mlstatpy/commit/e7a73bcb1ff45cafbb45ca8a3e69efbb5de028a0
204. https://github.com/sdpython/mlstatpy/commit/d1195b4fb76d9e7c77c54765ada7c18caf378fab
205. https://github.com/sdpython/mlstatpy/commit/c9a09e1305c1b1e5cc27f1381e066c0b009e6fdd
206. https://github.com/sdpython/mlstatpy/commit/e54bf0da2342360568aa44164525cad285280de6
207. https://github.com/sdpython/mlstatpy/commit/55c826c4047714e7b8137649b5347926d22947f5
208. https://github.com/sdpython/mlstatpy/commit/3fd2e239a2571a54c60c757324f5fb46235eef7e
209. https://github.com/sdpython/mlstatpy/commit/728af235c6c0a803b063515eee40dd973ec3c532
210. https://github.com/sdpython/mlstatpy/commit/cd731c8b6de8dc0fd19888a96ae4344867752b08
211. https://github.com/sdpython/mlstatpy/commit/fd68b079ef84f4522fe340c9a5ae4434284abdef
212. https://github.com/sdpython/mlstatpy/commit/54c62dd7d9ce97e49098f6079abd950be0cb9825
213. https://github.com/sdpython/mlstatpy/commit/2bcf256a85dcab0d4438b3916eab94a59665785e
214. https://github.com/sdpython/mlstatpy/commit/8d5c5e2b3f25dabf2f4eca1e98526dd285c468ef
215. https://github.com/sdpython/mlstatpy/commit/58c989b3b9c3494678d58280d0e4423988415ed2
216. https://github.com/sdpython/mlstatpy/commit/e4843def35d28e0f2f42bb5af9f09dc204198a58
217. https://github.com/sdpython/mlstatpy/commit/0b87bcdac69470ebd65541c5201d47877088ad72
218. https://github.com/sdpython/mlstatpy/commit/07289516bf3783c64344016c5a96e6c83c7c6230
219. https://github.com/sdpython/mlstatpy/commit/2355d97bfab1e381c704478b5b8bbd6fbf9f019e
220. https://github.com/sdpython/mlstatpy/commit/bea10fb071e7844b83e4f2370bd1f81b66e7094b
221. https://github.com/sdpython/mlstatpy/commit/40bdcc6df54a5133543925dfd4da9e8d3def3e45
222. https://github.com/sdpython/mlstatpy/commit/59998a996fb9b770f2d3285f905595943c20f46b
223. https://github.com/sdpython/mlstatpy/commit/eb44d270aa240733ee8b4c10c44b5633545cbcaa
224. https://github.com/sdpython/mlstatpy/commit/2b2d6f1f54cb85db985dd0a6ae5435ddf24361f7
225. https://github.com/sdpython/mlstatpy/commit/346d9decad4df69ded30e2e0319151e492dbe95f
226. https://github.com/sdpython/mlstatpy/commit/e11cf5388a10a916f76e4feb0bac501cc1776cf2
227. https://github.com/sdpython/mlstatpy/commit/c75eb22ec4b75f654957cd74aad19c3113617270
228. https://github.com/sdpython/mlstatpy/commit/40bd586ac8d727c3fd2cd16ecbff8e6d069bb3a7
229. https://github.com/sdpython/mlstatpy/commit/c637f30591524c2f1e1e3f5c31fbaf833c3ebab8
230. https://github.com/sdpython/mlstatpy/commit/6e730627e64dcf13bc3067b2213b1b6d39d4203d
231. https://github.com/sdpython/mlstatpy/commit/5cf2f7a6cdbdbe4590be82a64dc9686e650d6dbe
232. https://github.com/sdpython/mlstatpy/commit/d1f99e62073c48ae275ce3a4e9bcd0c9e80407ff
233. https://github.com/sdpython/mlstatpy/commit/3d6076c9320539794641a3ca4bc034d181a3adac
234. https://github.com/sdpython/mlstatpy/commit/6601fb6d22cd31a687d80f9a1214d5065ecdec88
235. https://github.com/sdpython/mlstatpy/commit/03dd7fcc06b5c69d272a46984d6f4cfe13a39766
236. https://github.com/sdpython/mlstatpy/commit/3eff2bf93d9671c108ec41f5ae2ed426dc316440
237. https://github.com/sdpython/mlstatpy/commit/e5d0df7e82f2807b6793560ce7cf8ac55d1879ad
238. https://github.com/sdpython/mlstatpy/commit/083fec0f787ac686f2dfcb7c42c29bf271009a13
239. https://github.com/sdpython/mlstatpy/commit/f0e8753fdd2dfebc2fd5043415d0c7ca5d06f345
240. https://github.com/sdpython/mlstatpy/commit/1c366a070a7c9331623e04bee431362c75c5522f
241. https://github.com/sdpython/mlstatpy/commit/6a4c0af3fba747696f9d61fdcbcbc84ebc8bf905
242. https://github.com/sdpython/mlstatpy/commit/0ad2a1352dd9948c140df5134663b7b0f4907ac1
243. https://github.com/sdpython/mlstatpy/commit/2e1cb0f079a4505cd3e2bb36531d1ef0aaef6b73
244. https://github.com/sdpython/mlstatpy/commit/5f879453315176cd91bd95b3fa8f069e498dc939
245. https://github.com/sdpython/mlstatpy/commit/3bdb45e6c491a5731ea9757ad799887e4310667d
246. https://github.com/sdpython/mlstatpy/commit/055758a83a642afaedb7b2ac7f8df034c137aac3
247. https://github.com/sdpython/mlstatpy/commit/63f0c68e47eab84d45a5fa0b1fb75972f803bfd8
248. https://github.com/sdpython/mlstatpy/commit/aaa2b4b31d104f9b403e844e0d2e8a8b06575c0c
249. https://github.com/sdpython/mlstatpy/commit/a5f6f40b8b8b0814950a60ae2b282c571a562598
250. https://github.com/sdpython/mlstatpy/commit/48363565cd13cc7456cc539e660c9d0095ead6b2
251. https://github.com/sdpython/mlstatpy/commit/0b85e3b1b92c5fc641e59e7314dc83bab848beda
252. https://github.com/sdpython/mlstatpy/commit/8b56312b6fa377cd4b620568cf8785ac8b49338b
253. https://github.com/sdpython/mlstatpy/commit/2dad0044ff72c9a9db31bc2e0d709e4409c7a791
254. https://github.com/sdpython/mlstatpy/commit/8a91d2a16948bc5cf766a4529a1002dd432b1e60
255. https://github.com/sdpython/mlstatpy/commit/9fe5e1a6bff335e348a05f09a6d953f64cd8bdef
256. https://github.com/sdpython/mlstatpy/commit/af2ea578f7ddfb6470823d3364094ec09db5279e
257. https://github.com/sdpython/mlstatpy/commit/ea03d571bcb80deb3afdb876946069f1c41400d8
258. https://github.com/sdpython/mlstatpy/commit/f212630952...

Statistics on code

extension/kind	nb lines	nb doc lines	nb files
.py	4374	1482	28
__init__.py	86	47	8

11.1.2 README

11.1.3 Setup

Installation

```
pip install mlstatpy
```

Le module peut être aussi téléchargé depuis pypi/mlstatpy [568] ou github/mlstaty [569] pour une version plus récente.

Generate this documentation

See Generating the documention with pyquickhelper [570].

Configuration :

```
# -*- coding: utf-8 -*-
import sys
import os
from pyquickhelper.helpgen.default_conf import set_sphinx_variables, get_default_
↪stylesheet

choice = "bootstrap"

if choice == "sphtheme":
    import sphinx_theme_pd as sphtheme
    html_theme = sphtheme.__name__
    html_theme_path = [sphtheme.get_html_theme_path()]
elif choice == "bootstrap":
    import sphinx_bootstrap_theme
    html_theme = 'bootstrap'
    html_theme_path = sphinx_bootstrap_theme.get_html_theme_path()
else:
    raise NotImplementedError()

sys.path.insert(0, os.path.abspath(os.path.join(os.path.split(__file__)[0])))

local_template = os.path.join(os.path.abspath(
    os.path.dirname(__file__)), "phdoc_templates")

set_sphinx_variables(__file__, "mlstatpy", "Xavier Dupré", 2018,
                     html_theme, html_theme_path, locals(),
                     extlinks=dict(
                         issue=('https://github.com/sdpython/mlstatpy/issues/%s',
↪'issue')),
```

(suite sur la page suivante)

568. https://pypi.python.org/pypi/mlstatpy
569. https://github.com/sdpython/mlstatpy/
570. http://www.xavierdupre.fr/app/pyquickhelper/helpsphinx/generatedoc.html

(suite de la page précédente)

```
                    title="Machine Learning, Statistiques et Programmation",␣
↪book=True, nblayout='table')

# next

blog_root = "http://www.xavierdupre.fr/app/mlstatpy/helpsphinx/"

html_context = {
    'css_files': get_default_stylesheet() + ['_static/my-styles.css'],
}

html_logo = "project_ico_small.png"

if choice == "bootstrap":
    html_theme_options = {
        'navbar_title': ".",
        'navbar_site_name': "Site",
        'navbar_links': [
            ("XD", "http://www.xavierdupre.fr", True),
            ("blog", "blog/main_0000.html", True),
            ("index", "genindex"),
        ],
        'navbar_sidebarrel': True,
        'navbar_pagenav': True,
        'navbar_pagenav_name': "Page",
        'bootswatch_theme': "readable",
        # united = weird colors, sandstone=green, simplex=red, paper=trop bleu
        # lumen: OK
        # to try, yeti, flatly, paper
        'bootstrap_version': "3",
        'source_link_position': "footer",
    }

language = "fr"

preamble = '''
\\usepackage{etex}
\\usepackage{fixltx2e} % LaTeX patches, \\textsubscript
\\usepackage{cmap} % fix search and cut-and-paste in Acrobat
\\usepackage[raccourcis]{fast-diagram}
\\usepackage{titlesec}
\\usepackage{amsmath}
\\usepackage{amssymb}
\\usepackage{amsfonts}
\\usepackage{graphics}
\\usepackage{epic}
\\usepackage{eepic}
%\\usepackage{pict2e}
%%% Redefined titleformat
\\setlength{\\parindent}{0cm}
\\setlength{\\parskip}{1ex plus 0.5ex minus 0.2ex}
\\newcommand{\\hsp}{\\hspace{20pt}}
\\newcommand{\\acc}[1]{\\left\\{#1\\right\\}}
\\newcommand{\\cro}[1]{\\left[#1\\right]}
\\newcommand{\\pa}[1]{\\left(#1\\right)}
\\newcommand{\\R}{\\mathbb{R}}
\\newcommand{\\HRule}{\\rule{\\linewidth}{0.5mm}}
```

(suite sur la page suivante)

(suite de la page précédente)

```
%\\titleformat{\\chapter}[hang]{\\Huge\\bfseries\\sffamily}{\\thechapter\\hsp}{0pt}
↪{\\Huge\\bfseries\\sffamily}
'''

custom_preamble = """\n
\\usepackage[all]{xy}
\\newcommand{\\vecteur}[2]{\\pa{#1,\\dots,#2}}
\\newcommand{\\N}[0]{\\mathbb{N}}
\\newcommand{\\indicatrice}[1]{\\mathbf{1\\!\\!1}_{\\acc{#1}}}
\\newcommand{\\infegal}[0]{\\leqslant}
\\newcommand{\\supegal}[0]{\\geqslant}
\\newcommand{\\ensemble}[2]{\\acc{#1,\\dots,#2}}
\\newcommand{\\fleche}[1]{\\overrightarrow{ #1 }}
\\newcommand{\\intervalle}[2]{\\left\\{#1,\\cdots,#2\\right\\}}
\\newcommand{\\independant}[0]{\\;\\makebox[3ex]{\\makebox[0ex]{\\rule[-0.2ex]{3ex}{.
↪1ex}}\\!\\!\\!\\!\\!\\makebox[.5ex][l]{\\rule[-.2ex]{.1ex}{2ex}}\\makebox[.5ex][l]
↪{\\rule[-.2ex]{.1ex}{2ex}}} \\,\\,}
\\newcommand{\\esp}{\\mathbb{E}}
\\newcommand{\\espf}[2]{\\mathbb{E}_{#1}\\pa{#2}}
\\newcommand{\\var}{\\mathbb{V}}
\\newcommand{\\pr}[1]{\\mathbb{P}\\pa{#1}}
\\newcommand{\\loi}[0]{{\\cal L}}
\\newcommand{\\vecteurno}[2]{#1,\\dots,#2}
\\newcommand{\\norm}[1]{\\left\\Vert#1\\right\\Vert}
\\newcommand{\\norme}[1]{\\left\\Vert#1\\right\\Vert}
\\newcommand{\\scal}[2]{\\left<#1,#2\\right>}
\\newcommand{\\dans}[0]{\\rightarrow}
\\newcommand{\\partialfrac}[2]{\\frac{\\partial #1}{\\partial #2}}
\\newcommand{\\partialdfrac}[2]{\\dfrac{\\partial #1}{\\partial #2}}
\\newcommand{\\trace}[1]{tr\\pa{#1}}
\\newcommand{\\sac}[0]{|}
\\newcommand{\\abs}[1]{\\left|#1\\right|}
\\newcommand{\\loinormale}[2]{{\\cal N} \\pa{#1,#2}}
\\newcommand{\\loibinomialea}[1]{{\\cal B} \\pa{#1}}
\\newcommand{\\loibinomiale}[2]{{\\cal B} \\pa{#1,#2}}
\\newcommand{\\loimultinomiale}[1]{{\\cal M} \\pa{#1}}
\\newcommand{\\variance}[1]{\\mathbb{V}\\pa{#1}}
"""
# \\usepackage{eepic}

imgmath_latex_preamble = preamble + custom_preamble
latex_elements['preamble'] = preamble + custom_preamble
mathdef_link_only = True

epkg_dictionary["R-tree"] = "https://en.wikipedia.org/wiki/R-tree"
epkg_dictionary["B+ tree"] = "https://en.wikipedia.org/wiki/B%2B_tree"
epkg_dictionary["R* tree"] = "https://en.wikipedia.org/wiki/R*_tree"
epkg_dictionary["X-tree"] = "https://en.wikipedia.org/wiki/X-tree"
epkg_dictionary["Branch and Bound"] = "https://en.wikipedia.org/wiki/Branch_and_bound"
epkg_dictionary["LAESA"] = "https://tavianator.com/aesa/"
epkg_dictionary["AESA"] = "https://tavianator.com/aesa/"
```

Extensions to install

— pyquickhelper [571]
— wild_sphinx_theme [572]

Tips

Module pyquickhelper [573] defines sphinx command runpython [574] which generates from a python script included in the documentation itself. The following snippet produces a table.

<<<

```
from pyquickhelper.pandashelper import df2rst
import pandas
df = pandas.DataFrame([{"x": 3, "y": 4}, {"x": 3.5, "y": 5}])
print(df2rst(df))
```

>>>

x	y
3.0	4.0
3.5	5.0

The next one is more complex. The code produces titles, label and references. It requires Sphinx engine to be processed.

<<<

```
rows = []
list_title = ["T1", "T2", "T3"]
back = None
for t in list_title:
    rows.append("")
    rows.append(".. _l-fake_title-" + t + ":")
    rows.append("")
    rows.append(t*3)
    rows.append("^" * len(t*3))
    rows.append("")
    if back:
        rows.append("link :ref:`l-fake_title-" + back + "`")
    else:
        rows.append("no link")
    rows.append("")
    back = t
print("\n".join(rows))
```

>>>

T1T1T1

no link

571. https://pypi.python.org/pypi/pyquickhelper/
572. https://pypi.python.org/pypi/wild_sphinx_theme
573. https://pypi.python.org/pypi/pyquickhelper/
574. http://www.xavierdupre.fr/app/pyquickhelper/helpsphinx/pyquickhelper/helpgen/sphinx_runpython_extension.html

T2T2T2

link *T1T1T1* (page 295)

T3T3T3

link *T2T2T2* (page 296)

Sortie brute

```
.. _l-fake_title-T1:

T1T1T1
^^^^^^

no link


.. _l-fake_title-T2:

T2T2T2
^^^^^^

link :ref:`l-fake_title-T1`


.. _l-fake_title-T3:

T3T3T3
^^^^^^

link :ref:`l-fake_title-T2`
```

Generate the setup

See Generating the setup with pyquickhelper [575].

Extensions to install

— pyquickhelper [576]

Documentation, unit tests, setup

See Unit tests with pyquickhelper [577].

Extensions to install

— pyquickhelper [578]

575. http://www.xavierdupre.fr/app/pyquickhelper/helpsphinx/generatesetup.html
576. https://pypi.python.org/pypi/pyquickhelper/
577. http://www.xavierdupre.fr/app/pyquickhelper/helpsphinx/doctestunit.html
578. https://pypi.python.org/pypi/pyquickhelper/

11.1.4 Blog Gallery

2018-08-10 One Hundred Probability/Statistics Inequalities

Découvert dans un tweet : One Hundred Probability/Statistics Inequalities [579].

2017-02-16 Adam

Adam [580] n'est pas le personnage de la saison 4 de Buffy contre les vampires [581] mais un algorithme de descente de gradient : Adam : A Method for Stochastic Optimization [582]. Si vous ne me croyez pas, vous devriez lire cette petite revue An overview of gradient descent optimization algorithms [583]. Un autre algorithme intéressant est Hogwild [584], asynchrone et distribué. Bref, a unicorn comme disent les anglais.

2016-08-17 Articles autour du gradient

— DSA : Decentralized Double Stochastic Averaging Gradient Algorithm [585]
— Distributed Coordinate Descent Method for Learning with Big Data [586]

2016-06-19 Premier blog, juste un essai

Premier blog.

2016-06-19 Lectures

Un article intéressant plus pratique que théorique : Recent Advances in Convolutional Neural Networks [587].

579. http://www.npslagle.info/articles/onehundredprobabilityinequalities.pdf
580. https://en.wikipedia.org/wiki/Adam_(Buffy_the_Vampire_Slayer)
581. https://en.wikipedia.org/wiki/Buffy_the_Vampire_Slayer
582. https://arxiv.org/abs/1412.6980
583. http://sebastianruder.com/optimizing-gradient-descent/
584. http://sebastianruder.com/optimizing-gradient-descent/index.html#hogwild
585. http://www.jmlr.org/papers/volume17/15-292/15-292.pdf
586. http://www.jmlr.org/papers/volume17/15-001/15-001.pdf
587. http://arxiv.org/abs/1512.07108

CHAPITRE 12

Galleries

12.1 Le petit coin des data scientists

Ce sont quelques notebooks sur des points particuliers qui surgissent au quotidien quand on traite des données.

12.1.1 Classification multiple

Explorations autour d'un problème de classification multiple.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```


Début de l'histoire

$\mathbb{1}_{y_i}$

Confusions

Un des premiers réflexes après avoir appris une classification multi-classe est de regarder la matrice de confusion [588]. Certaines classes sont difficiles à classer, d'autres non. Je me demandais s'il existait un moyen de déterminer cela sans apprendre un classifieur. On souhaite apprendre la classification des points (X_i, y_i), X_i est un vecteur, y_i la classe attendue. Si $\hat{y_i}$ est la classe prédite, l'erreur de classification est :

$$E = \sum_i \mathbb{1}_{y_i \neq \hat{y_i}}$$

On note $c_{ij} = \mathbb{1}_{y_i=j}$ et $\hat{c_{ij}} = \mathbb{1}_{\hat{y_i}=j}$. On note le vecteur $C_j = (c_{ij})_i$ et $\hat{C_j} = (\hat{c_{ij}})_i$. On peut réécrire l'erreur comme :

$$E = \sum_{ij} \mathbb{1}_{y_i=j} \mathbb{1}_{\hat{y_i} \neq j} = \sum_{ij} \mathbb{1}_{y_i=j}(1 - \mathbb{1}_{\hat{y_i}=j}) = \sum_{ij} c_{ij}(1 - \hat{c_{ij}}) = \sum_j < C_j, 1 - \hat{C_j} >$$

C'est aussi égal à :

$$E = \sum_{k \neq j} < C_j, \hat{C_k} >$$

Et $< C_j, \hat{C_k} >$ correspond au nombre d'erreurs de confusion : le nombre d'éléments de la classe j classés dans la classe k. $< C_j, \hat{C_k} >$ est le nombre d'éléments correctement classés dans la classe j. On peut montrer que

$$\sum_{k,j} < C_j, \hat{C_k} >= N$$

où N est le nombre d'observations.

Clustering

Et si nous introduisions un clustering intermédiaire. On construit Q cluster, q_i est le cluster du point X_i et on note $d_{il} = \mathbb{1}_{q_i=l}$ et le vecteur $D_l = (d_{il})_i$.

$$E = \sum_{k \neq j} < C_j, \hat{C_k} >$$

On note $X.Y$ le produit terme à terme de deux vecteurs.

$$E = \sum_{k \neq j, l} < C_j.D_l, \hat{C_k} >= \sum_{k \neq j, l} < C_j.D_l, \hat{C_k}.D_l >$$

Le nombre d'erreurs est la somme des erreurs faites sur chaque cluster. Supposons maintenant qu'un classifieur retourne une réponse constante sur chacun des clusters, on choisit la classe plus représentée. Ca ressemble beaucoup à un classifieur bayésien [589]. On note $f(l)$ cette classe la plus représentée. Elle vérifie :

$$f(l) = \arg\max_j < C_j, D_l >$$

Cela signifie que $\hat{c_{ij}} = \sum_l \mathbb{1}_{j=f(l)} d_{il}$. Si on note $l(i)$ le cluster associé à i. On continue : $\hat{c_{ij}} = \mathbb{1}_{j=f(l(i))}$. On définit l'erreur $e(l)$ l'erreur de classification faite sur chaque cluster l :

$$e(l) = \sum_i d_{il} \sum_j c_{ij}(1 - \mathbb{1}_{j=f(l)}) = \sum_i d_{il} \left(\sum_j c_{ij} - \sum_j c_{ij} \mathbb{1}_{j=f(l)} \right) = \sum_i d_{il} \left(1 - c_{i,f(l)}\right) = \sum_i d_{il} - \sum_i d_{il} c_{i,f(l)}$$

Pour résumer, l'erreur est le nombre d'éléments moins le nombre d'éléments dans la classe majoritaire du cluster. Si le nombre de clusters Q devient supérieur ou égal au nombre d'observations, cette erreur devient nulle.

588. https://fr.wikipedia.org/wiki/Matrice_de_confusion
589. http://scikit-learn.org/stable/modules/naive_bayes.html

Mise en pratique

L'idée est de voir comment évolue cette erreur de classification naïve en fonction du nombre de clusters. La différence par rapport à un classifieur est qu'on sait comment sont fabriqués les clusters et qu'on peut imaginer les classes comme un assemblage de clusters d'une forme connue.

12.1.2 File d'attente, un exemple simple

Cet exemple vient illustrer le paragraphe sur les files d'attente et l'espérance de vie des ampoules.

```
%matplotlib inline
```

```
import math
import random

def generate_expo(mu):
    return random.expovariate(mu)

generate_expo(2)
```

```
0.0749720223112896
```

Les paramètres de la simulation.

```
S = 10000
iteration = 500
mu = 1.0 / 100
```

On crée un tableau de `S` ampoules qui contient la durée de vie restante de chaque ampoule.

```
ampoule = [0 for a in range(0,S)]
moyenne_grille = 0
stats = []


for i in range(0,iteration):
    grille = 0
    mean   = 0

    for n in range(0,S):
        mean += ampoule[n]
        if ampoule[n] == 0:
            # remplacement d'une ampoule grillée
            grille += 1
            # on détermine la durée de vie de cette ampoule
            # on arrondit à l'entier le plus proche
            ampoule[n] = int (generate_expo(mu))
        else :
            # on enlève une heure à la durée de vie de l'ampoule
            ampoule[n] -= 1

    mean /= S

    stats.append(dict(i=i, mean=mean, grille=grille))

    if i > 0:
```

(suite sur la page suivante)

(suite de la page précédente)

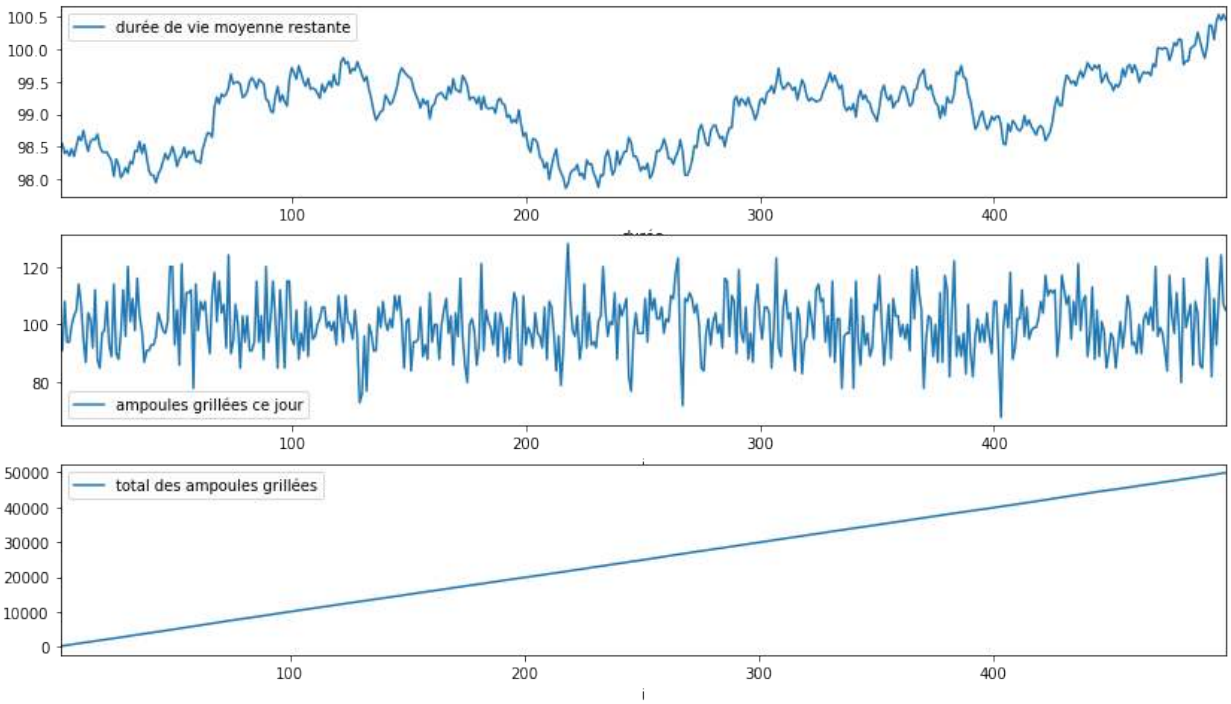
```
        moyenne_grille += grille
    if i % 100 == 0:
        print("itération : ", i, " moyenne durée : ", mean, " grillées :", grille)

moyenne_grille = float (moyenne_grille) / float (iteration - 1)
print("nombre moyen d'ampoules grillées :", moyenne_grille)
```

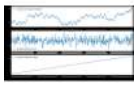
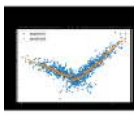
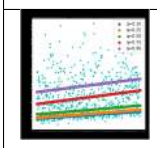
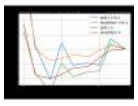
```
itération :  0  moyenne durée :  0.0  grillées : 10000
itération :  100  moyenne durée :  99.7184  grillées : 95
itération :  200  moyenne durée :  98.7154  grillées : 93
itération :  300  moyenne durée :  99.2155  grillées : 101
itération :  400  moyenne durée :  98.9101  grillées : 108
nombre moyen d'ampoules grillées : 99.88577154308618
```

```
import pandas
df = pandas.DataFrame(stats)
df = df[["i", "mean", "grille"]]
df["grille_sum"] = df["grille"].cumsum() - 10000
df.head()
```

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(3, 1, figsize=(14,8))
df[1:].plot(x="i", y="mean", label="durée de vie moyenne restante", ax=ax[0])
df[1:].plot(x="i", y="grille", label="ampoules grillées ce jour", ax=ax[1])
df[2:].plot(x="i", y="grille_sum", label="total des ampoules grillées", ax=ax[2])
ax[0].set_xlabel("durée")
```

```
<matplotlib.text.Text at 0x1e2155ce4a8>
```



	*Classification multiple* (page 299)	Explorations autour d'un problème de classification multiple.
	*Corrélations non linéaires* (page 192)	Les corrélations indiquent si deux variables sont linéairement équivalentes. Comment étendre cette notion à des variables liées mais pas de façon linéaire.
	*File d'attente, un exemple simple* (page 301)	Cet exemple vient illustrer le paragraphe sur les files d'attente et l'espérance de vie des ampoules.
array([0.,	*Le gradient et le discret* (page 234)	Les méthodes d'optimisation à base de gradient s'appuie sur une fonction d'erreur dérivable qu'on devrait appliquer de préférence sur des variables aléatoires réelles. Ce notebook explore quelques idées.
	*Régression linéaire et résultats numériques* (page 221)	Ce notebook s'intéresse à la façon d'interpréter les résultats d'une régression linéaire lorsque les variables sont corrélées puis il explore une façon d'associer arbre de décision et régression linéaire pour construire une régression linéaire par morceaux.
	*Régression quantile illustrée* (page 246)	La régression quantile est moins sensible aux points aberrants. Elle peut être définie comme une régression avec une norme *L1* (une valeur absolue). Ce notebook explore des régressions avec des quantiles différents.
	*Répartir en base d'apprentissage et de test* (page 179)	C'est un problème plutôt facile puisqu'il s'agit de répartir aléatoirement les lignes d'une base de données d'un côté ou de l'autre. Lorsque le problème de machine learning à résoudre est un problème de classification, il faut s'assurer que chaque côté contient une proportion raisonnable de chaque classe.

12.2 Images

	*Détection de segments dans une image* (page 172)	

12.3 Métriques

12.3.1 ROC

A few graphs about ROC on the iris datasets.

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```


Iris datasets

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
import numpy
ypred = clf.predict(X_test)
yprob = clf.predict_proba(X_test)
score = numpy.array(list(yprob[i,ypred[i]] for i in range(len(ypred))))
```

```
data = numpy.zeros((len(ypred), 2))
data[:,0] = score.ravel()
data[ypred==y_test,1] = 1
data[:5]
```

```
array([[ 0.70495209,  1.        ],
       [ 0.56148737,  0.        ],
       [ 0.56148737,  1.        ],
       [ 0.77416227,  1.        ],
       [ 0.58631799,  0.        ]])
```

ROC with scikit-learn

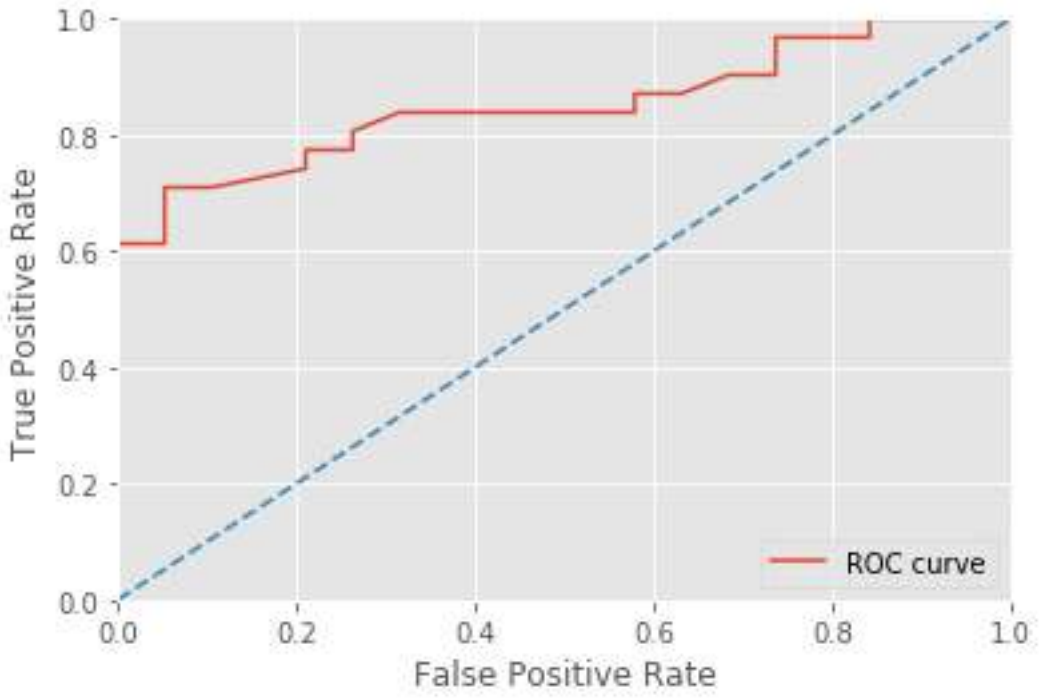
We use the following example Receiver Operating Characteristic (ROC) [590].

```
from sklearn.metrics import roc_curve
fpr, tpr, th = roc_curve(y_test == ypred, score)
```

```
import matplotlib.pyplot as plt
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
```

```
<matplotlib.legend.Legend at 0x268373a2128>
```

590. http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#sphx-glr-auto-examples-model-selection-plot-roc-py




```
import pandas
df = pandas.DataFrame(dict(fpr=fpr, tpr=tpr, threshold=th))
df
```

ROC - TPR / FPR

We do the same with the class this module provides ROC [591].

— TPR = True Positive Rate

— FPR = False Positive Rate

You can see as TPR the distribution function of a score for a positive example and the FPR the same for a negative example.

```
from mlstatpy.ml.roc import ROC
```

```
roc = ROC(df=data)
```

```
roc
```

```
Overall precision: 0.63 - AUC=0.850594
--------------
      score  label  weight
0  0.375573    0.0     1.0
1  0.385480    0.0     1.0
2  0.412314    0.0     1.0
3  0.412703    1.0     1.0
4  0.417941    0.0     1.0
--------------
       score  label  weight
```

(suite sur la page suivante)

591. http://www.xavierdupre.fr/app/mlstatpy/helpsphinx/mlstatpy/ml/roc.html?highlight=roc#module-mlstatpy.ml.roc

(suite de la page précédente)

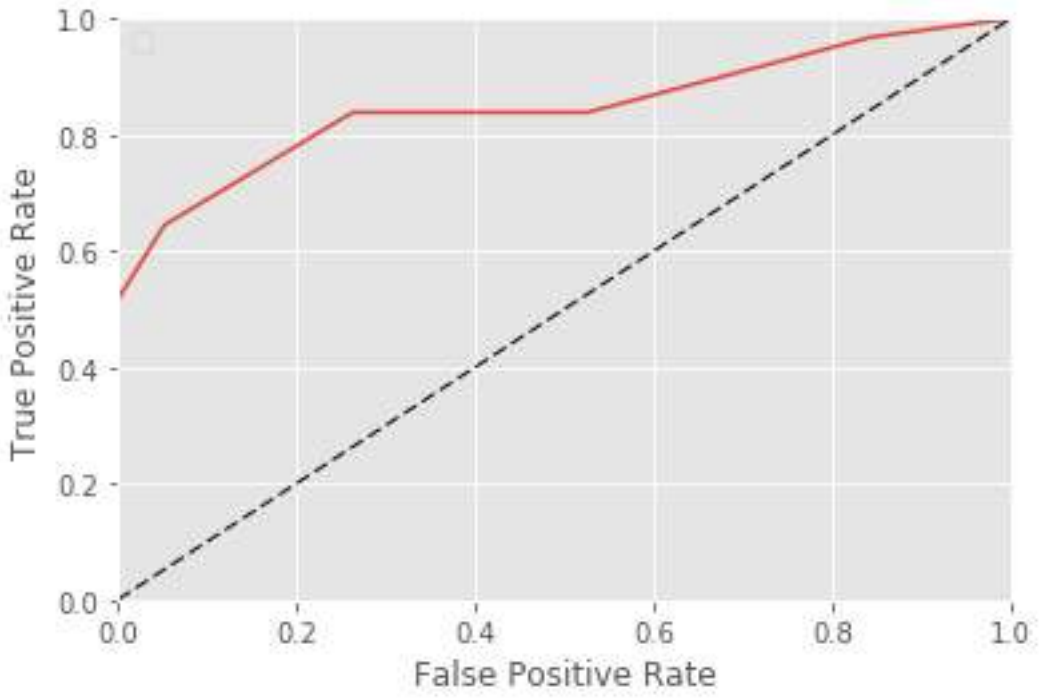
```
45  0.863174    1.0     1.0
46  0.863174    1.0     1.0
47  0.869794    1.0     1.0
48  0.903335    1.0     1.0
49  0.910712    1.0     1.0
--------------
    False Positive Rate  True Positive Rate  threshold
0              0.000000            0.032258   0.910712
1              0.000000            0.193548   0.828617
2              0.000000            0.354839   0.790909
3              0.000000            0.516129   0.737000
4              0.052632            0.645161   0.627589
5              0.157895            0.741935   0.607975
6              0.263158            0.838710   0.561487
7              0.526316            0.838710   0.542211
8              0.684211            0.903226   0.520835
9              0.842105            0.967742   0.417941
10             1.000000            1.000000   0.375573
--------------
       error  recall  threshold
0   0.000000    0.02   0.910712
1   0.000000    0.12   0.828617
2   0.000000    0.22   0.790909
3   0.000000    0.32   0.737000
4   0.047619    0.42   0.627589
5   0.115385    0.52   0.607975
6   0.161290    0.62   0.561487
7   0.277778    0.72   0.542211
8   0.317073    0.82   0.520835
9   0.347826    0.92   0.417941
10  0.380000    1.00   0.375573
```

```
roc.auc()
```

```
0.85059422750424452
```

```
roc.plot(nb=10)
```

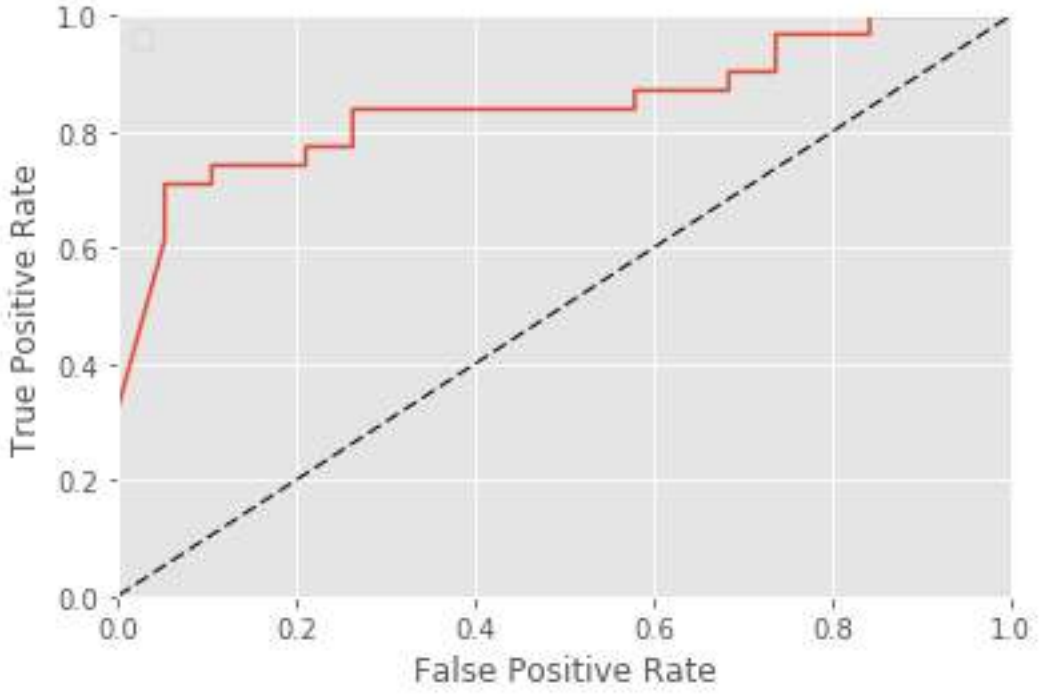
```
<matplotlib.axes._subplots.AxesSubplot at 0x2683ff2b668>
```



This function draws the curve with only 10 points but we can ask for more.

```
roc.plot(nb=100)
```

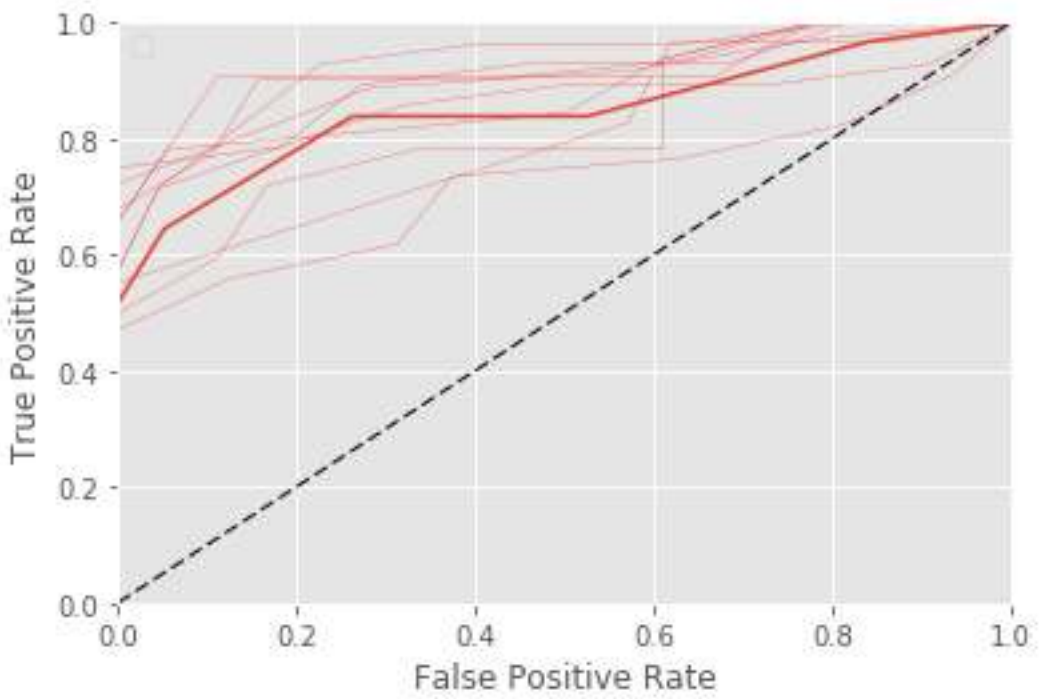
```
<matplotlib.axes._subplots.AxesSubplot at 0x2683feba240>
```



We can also ask to draw bootstropped curves to get a sense of the confidence.

```
roc.plot(nb=10, bootstrap=10)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x26840008748>
```

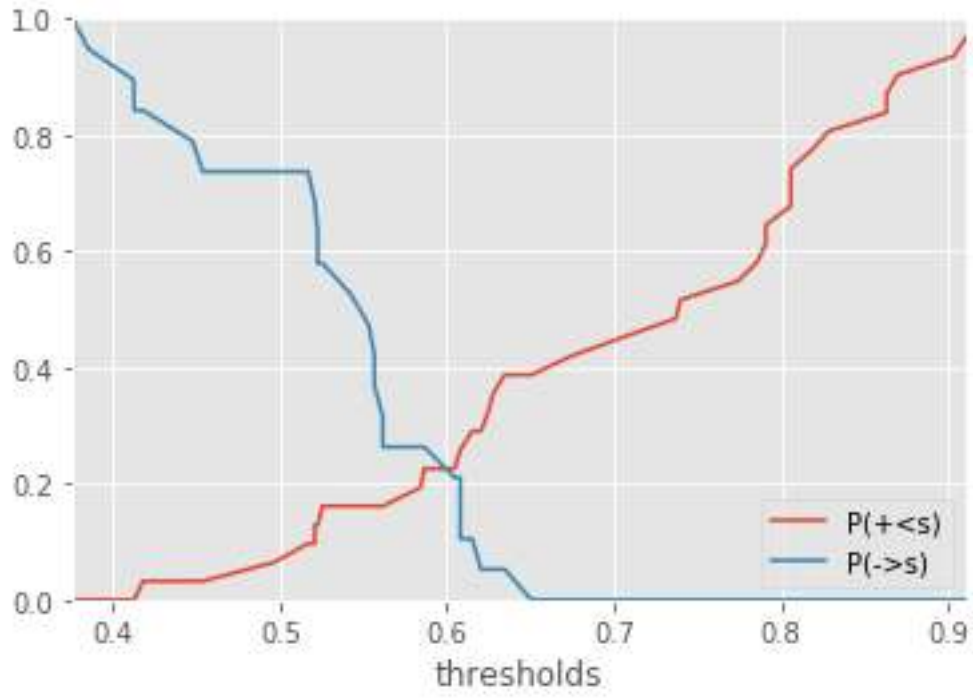


ROC - score distribution

This another representation for the metrics FPR and TPR. $P(x < s)$ is the probability that a score for a positive example to be less than s. $P(- > s)$ is the probability that a score for a negative example to be higher than s. We assume in this case that the higher the better for the score.

```
roc.plot(curve=ROC.CurveType.PROBSCORE, thresholds=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x268410618d0>
```



When curves intersects at score s^*, error rates for positive and negative examples are equal. If we show the confusion matrix for this particular score s^*, it gives :

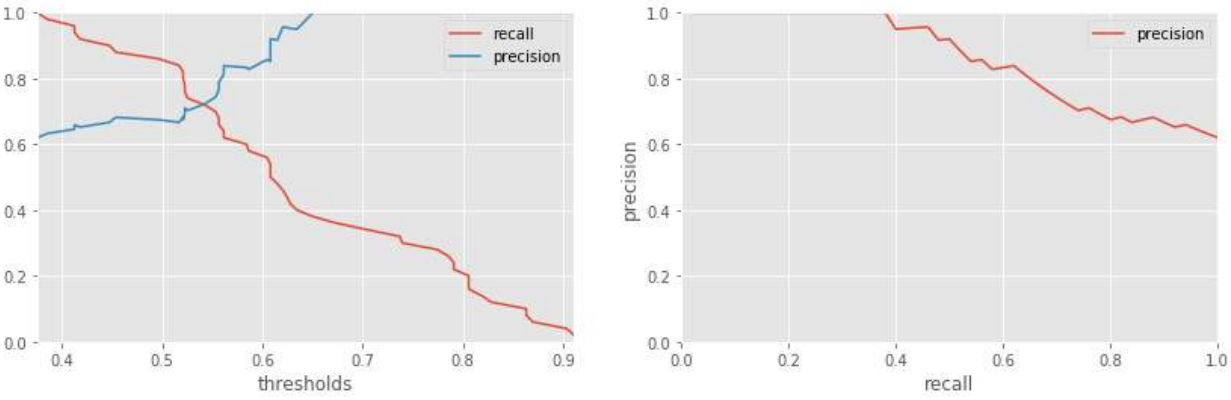
```
conf = roc.confusion()
conf["P(+<s)"] = 1 - conf["True Positive"] / conf.loc[len(conf)-1,"True Positive"]
conf["P(->s)"] = 1 - conf["True Negative"] / conf.loc[0,"True Negative"]
conf
```

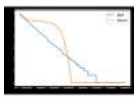
ROC - recall / precision

In this representation, we show the score.

```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(ncols=2, nrows=1, figsize=(14,4))
roc.plot(curve=ROC.CurveType.RECPREC, thresholds=True, ax=axes[0])
roc.plot(curve=ROC.CurveType.RECPREC, ax=axes[1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2684151d3c8>
```



	*ROC* (page 304)	A few graphs about ROC on the iris datasets.
	*p-values* (page 145)	Compute p-values.

12.4 Machine Learning

12.4.1 Benchmark

Ce notebook compare différents modèles depuis un notebook.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```


Si le message *Widget Javascript not detected. It may not be installed or enabled properly.* apparaît, vous devriez exécuter la commande `jupyter nbextension enable --py --sys-prefix widgetsnbextension` depuis la ligne de commande. Le code suivant vous permet de vérifier que cela a été fait.

```
from tqdm import tnrange, tqdm_notebook
from time import sleep

for i in tnrange(3, desc='1st loop'):
    for j in tqdm_notebook(range(20), desc='2nd loop'):
        sleep(0.01)
```

```
%matplotlib inline
```

Petit bench sur le clustering

Définition du bench

```
import dill
from tqdm import tnrange
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.datasets import make_blobs
from mlstatpy.ml import MlGridBenchMark

params = [dict(model=lambda : KMeans(n_clusters=3), name="KMeans-3", shortname="km-3
↪"),
```

(suite sur la page suivante)

(suite de la page précédente)

```
          dict(model=lambda : AgglomerativeClustering(), name="AgglomerativeClustering
↪", shortname="aggclus")]

datasets = [dict(X=make_blobs(100, centers=3)[0], Nclus=3,
                 name="blob-100-3", shortname="b-100-3", no_split=True),
            dict(X=make_blobs(100, centers=5)[0], Nclus=5,
                 name="blob-100-5", shortname="b-100-5", no_split=True) ]

bench = MlGridBenchMark("TestName", datasets, fLOG=None, clog=None,
                        cache_file="cache.pickle", pickle_module=dill,
                        repetition=3, progressbar=tnrange,
                        graphx=["_time", "time_train", "Nclus"],
                        graphy=["silhouette", "Nrows"])
```

Lancer le bench

```
bench.run(params)
```

```
0/|/2017-03-19 20:11:11 [BenchMark.run] number of cached run: 4:   0%|| 0/4 [00:00<?,␣
↪?it/s]
```

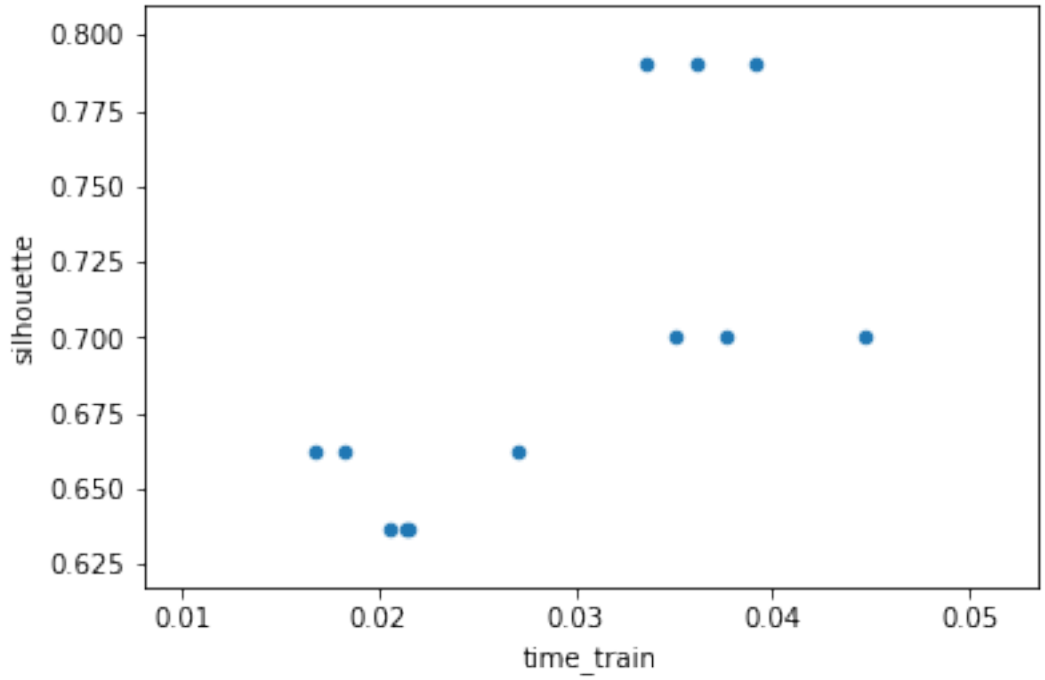
```
3/|/2017-03-19 20:11:13 [BenchMark.run] done.:  75%|| 3/4 [00:02<00:00,  1.10it/s]    ␣
↪                                  11it/s]_train': 0.02142968022685221, 'time_test':␣
↪0.0025012412126208527, '_btry': 'aggclus-b-100-5', '_iexp': 2, 'model_name':
↪'AgglomerativeClustering', 'ds_name': 'blob-100-5', 'Nrows': 100, 'Nfeat': 2, 'Nclus
↪': 5, 'no_split': True, '_date': datetime.datetime(2017, 3, 19, 20, 11, 11, 647355),
↪ '_time': 0.1007650830318858, '_span': datetime.timedelta(0, 0, 112581), '_i': 3, '_
↪name': 'TestName'}:  75%|| 3/4 [00:00<00:00,  4.22it/s]]0:00,  3.53it/s]]
```

Récupérer les résultats

```
df = bench.to_df()
df
```

```
df.plot(x="time_train", y="silhouette", kind="scatter")
```

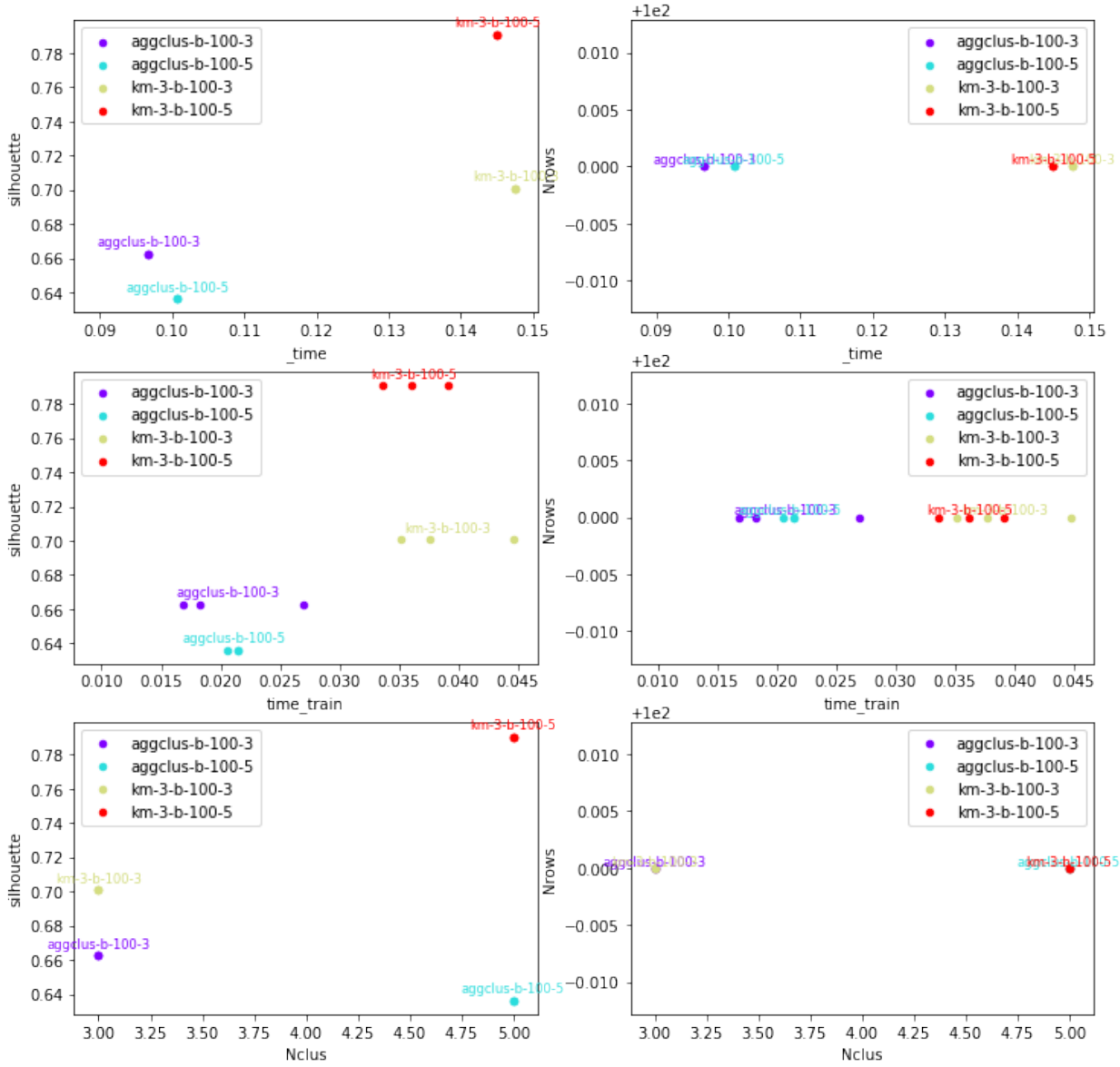
```
<matplotlib.axes._subplots.AxesSubplot at 0x122b8004748>
```



Dessin, Graphs

```
bench.plot_graphs(figsize=(12,12))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000122B8269A90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000122B82E1DA0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000122B83512E8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000122B83A1828>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000122B8409D68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000122B8462588>]],␣
↪dtype=object)
```



12.4.2 Factorisation et matrice et ACP

Un exemple pour montrer l'équivalence entre l'ACP et une factorisation de matrice.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```


— *Factorisation de matrices* (page 315)
— *ACP : analyse en composantes principales* (page 316)


```
%matplotlib inline
```

Factorisation de matrices

```
def erreur_mf(M, W, H):
    d = M - W @ H
    a = d.ravel()
    e = a @ a.T
    e ** 0.5 / (M.shape[0] * M.shape[1])
    return e
```

On crée un nuage de points avec que des coordonnées positives pour satisfaire les hypothèses de la factorisation de matrices.

```
from numpy.random import rand
M = rand(2, 20)
M[1,:] += 3 * M[0,:]
M
```

```
array([[ 0.81960047,  0.63887134,  0.74019269,  0.96110175,  0.0685406 ,
         0.11103301,  0.06033529,  0.67913157,  0.10460611,  0.98860048,
         0.50497448,  0.26893866,  0.73143267,  0.32617974,  0.1332449 ,
         0.83328515,  0.3775355 ,  0.69163261,  0.53095348,  0.15601268],
       [ 2.48031078,  2.2279066 ,  2.85929872,  3.27833973,  0.27323095,
         0.53806662,  0.48019992,  2.09428487,  0.40521666,  3.94539474,
         2.36639105,  1.66857684,  3.14027534,  1.94032092,  1.22602705,
         3.09679803,  1.696636  ,  2.69144798,  1.84350664,  1.16862532]])
```

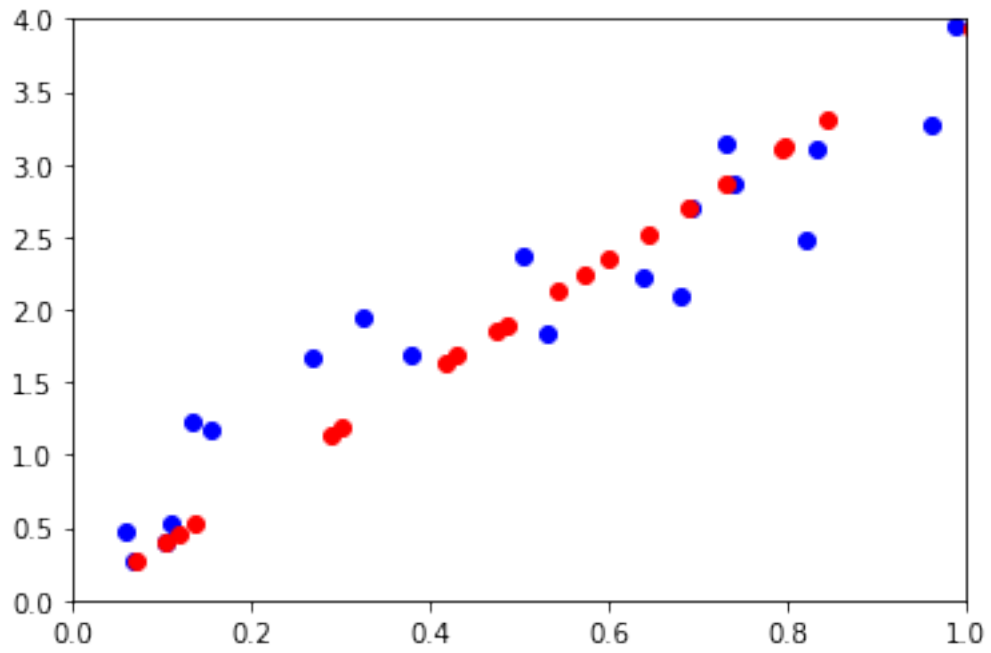
```
from sklearn.decomposition import NMF
mf = NMF(1)
W = mf.fit_transform(M)
H = mf.components_
erreur_mf(M, W, H)
```

```
0.19729615330190822
```

```
wh = W @ H
```

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1)
ax.plot(M[0,:], M[1,:], "ob")
ax.plot(wh[0,:], wh[1,:], "or")
ax.set_xlim([0,1])
ax.set_ylim([0,4])
```

```
(0, 4)
```



ACP : analyse en composantes principales

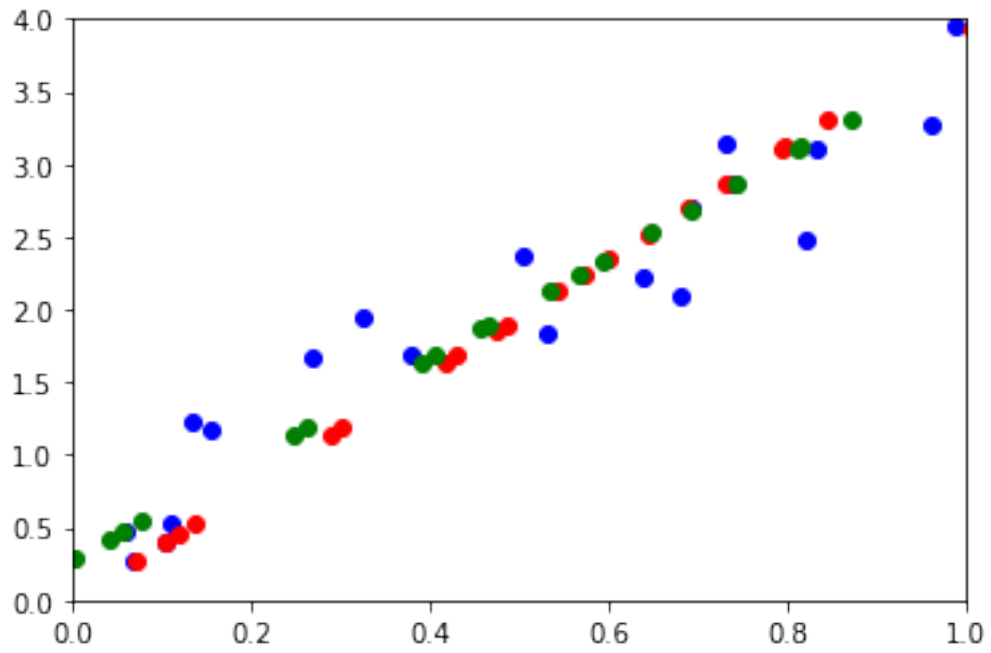
```
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
pca.fit(M.T)
```

```
PCA(copy=True, iterated_power='auto', n_components=1, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

```
projected_points = pca.inverse_transform(pca.transform(M.T))
pj = projected_points.T
```

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1)
ax.plot(M[0,:], M[1,:], "ob")
ax.plot(wh[0,:], wh[1,:], "or")
ax.plot(pj[0,:], pj[1,:], "og")
ax.set_xlim([0,1])
ax.set_ylim([0,4])
```

```
(0, 4)
```



Les résultats ne sont pas exactement identiques car l'ACP [592] centre le nuage de points par défaut. On utilise celui de statsmodels [593] pour éviter cela.

```
from statsmodels.multivariate.pca import PCA
pca = PCA(M.T, ncomp=1, standardize=False, demean=False, normalize=False)
pca
```

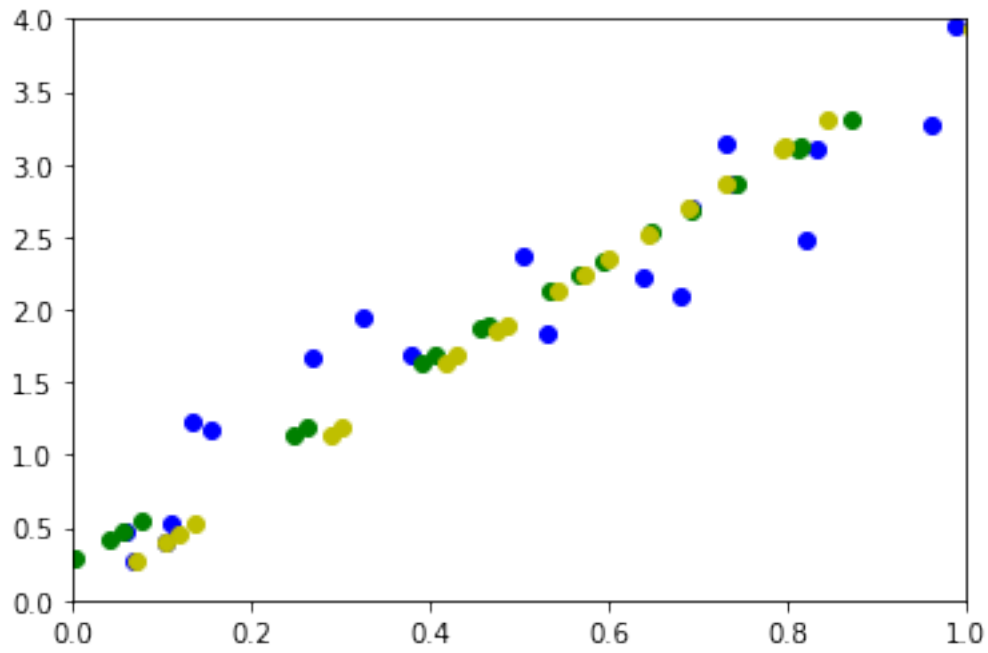
```
Principal Component Analysis(nobs: 20, nvar: 2, transformation: None, normalization:␣
↪False, number of components: 1, SVD, id: 0x1c01a2861d0)
```

```
pj2 = pca.projection.T
```

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1)
ax.plot(M[0,:], M[1,:], "ob")
#ax.plot(wh[0,:], wh[1,:], "or")
ax.plot(pj[0,:], pj[1,:], "og")
ax.plot(pj2[0,:], pj2[1,:], "oy")
ax.set_xlim([0,1])
ax.set_ylim([0,4])
```

```
(0, 4)
```

592. http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
593. http://www.statsmodels.org/dev/generated/statsmodels.multivariate.pca.PCA.html



On retrouve exactement les mêmes résultats.

12.4.3 Réseaux de neurones

Réseaux de neurones avec scikit-learn.

```
%matplotlib inline
```

```
from sklearn.linear_model import Perceptron
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = Perceptron()
clf.fit(X, y)
```

```
Perceptron(alpha=0.0001, class_weight=None, eta0=1.0, fit_intercept=True,
      n_iter=5, n_jobs=1, penalty=None, random_state=0, shuffle=True,
      verbose=0, warm_start=False)
```

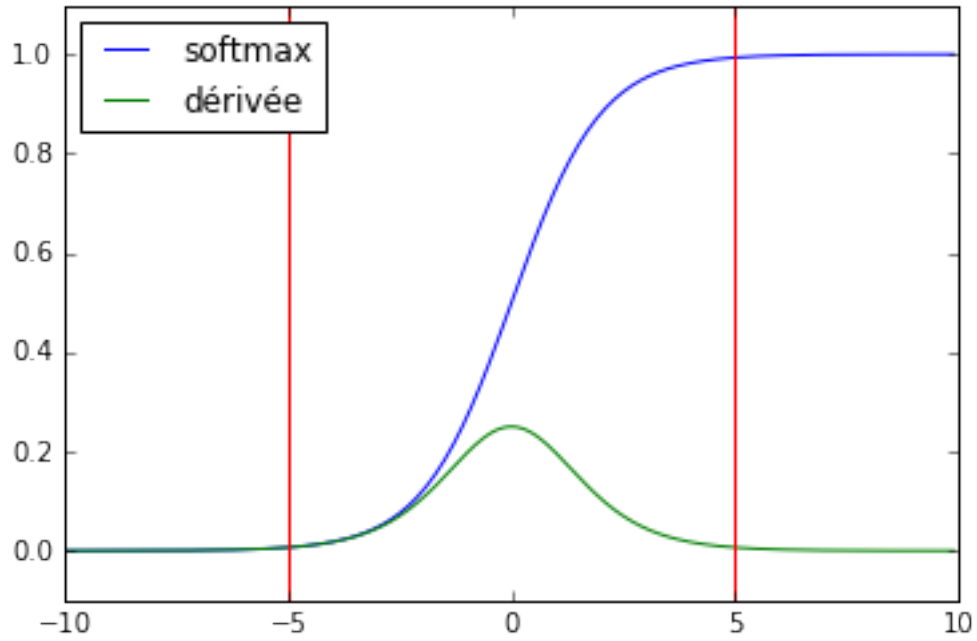
```
import matplotlib.pyplot as plt
import numpy
def softmax(x):
    return 1.0 / (1 + numpy.exp(-x))
def dsoftmax(x):
    t = numpy.exp(-x)
    return t / (1 + t)**2
x = numpy.arange(-10,10, 0.1)
y = softmax(x)
dy = dsoftmax(x)
fig, ax = plt.subplots(1,1)
ax.plot(x,y, label="softmax")
ax.plot(x,dy, label="dérivée")
ax.set_ylim([-0.1, 1.1])
```

(suite sur la page suivante)

(suite de la page précédente)

```
ax.plot([-5, -5], [-0.1, 1.1], "r")
ax.plot([5, 5], [-0.1, 1.1], "r")
ax.legend(loc=2)
```

```
<matplotlib.legend.Legend at 0x1b651aeacf8>
```




```
x
```

```
array([ -1.00000000e+01,  -9.90000000e+00,  -9.80000000e+00,
        -9.70000000e+00,  -9.60000000e+00,  -9.50000000e+00,
        -9.40000000e+00,  -9.30000000e+00,  -9.20000000e+00,
        -9.10000000e+00,  -9.00000000e+00,  -8.90000000e+00,
        -8.80000000e+00,  -8.70000000e+00,  -8.60000000e+00,
        -8.50000000e+00,  -8.40000000e+00,  -8.30000000e+00,
        -8.20000000e+00,  -8.10000000e+00,  -8.00000000e+00,
        -7.90000000e+00,  -7.80000000e+00,  -7.70000000e+00,
        -7.60000000e+00,  -7.50000000e+00,  -7.40000000e+00,
        -7.30000000e+00,  -7.20000000e+00,  -7.10000000e+00,
        -7.00000000e+00,  -6.90000000e+00,  -6.80000000e+00,
        -6.70000000e+00,  -6.60000000e+00,  -6.50000000e+00,
        -6.40000000e+00,  -6.30000000e+00,  -6.20000000e+00,
        -6.10000000e+00,  -6.00000000e+00,  -5.90000000e+00,
        -5.80000000e+00,  -5.70000000e+00,  -5.60000000e+00,
        -5.50000000e+00,  -5.40000000e+00,  -5.30000000e+00,
        -5.20000000e+00,  -5.10000000e+00,  -5.00000000e+00,
        -4.90000000e+00,  -4.80000000e+00,  -4.70000000e+00,
        -4.60000000e+00,  -4.50000000e+00,  -4.40000000e+00,
        -4.30000000e+00,  -4.20000000e+00,  -4.10000000e+00,
        -4.00000000e+00,  -3.90000000e+00,  -3.80000000e+00,
        -3.70000000e+00,  -3.60000000e+00,  -3.50000000e+00,
        -3.40000000e+00,  -3.30000000e+00,  -3.20000000e+00,
        -3.10000000e+00,  -3.00000000e+00,  -2.90000000e+00,
```

(suite sur la page suivante)

(suite de la page précédente)

```
       -2.80000000e+00,  -2.70000000e+00,  -2.60000000e+00,
       -2.50000000e+00,  -2.40000000e+00,  -2.30000000e+00,
       -2.20000000e+00,  -2.10000000e+00,  -2.00000000e+00,
       -1.90000000e+00,  -1.80000000e+00,  -1.70000000e+00,
       -1.60000000e+00,  -1.50000000e+00,  -1.40000000e+00,
       -1.30000000e+00,  -1.20000000e+00,  -1.10000000e+00,
       -1.00000000e+00,  -9.00000000e-01,  -8.00000000e-01,
       -7.00000000e-01,  -6.00000000e-01,  -5.00000000e-01,
       -4.00000000e-01,  -3.00000000e-01,  -2.00000000e-01,
       -1.00000000e-01,  -3.55271368e-14,   1.00000000e-01,
        2.00000000e-01,   3.00000000e-01,   4.00000000e-01,
        5.00000000e-01,   6.00000000e-01,   7.00000000e-01,
        8.00000000e-01,   9.00000000e-01,   1.00000000e+00,
        1.10000000e+00,   1.20000000e+00,   1.30000000e+00,
        1.40000000e+00,   1.50000000e+00,   1.60000000e+00,
        1.70000000e+00,   1.80000000e+00,   1.90000000e+00,
        2.00000000e+00,   2.10000000e+00,   2.20000000e+00,
        2.30000000e+00,   2.40000000e+00,   2.50000000e+00,
        2.60000000e+00,   2.70000000e+00,   2.80000000e+00,
        2.90000000e+00,   3.00000000e+00,   3.10000000e+00,
        3.20000000e+00,   3.30000000e+00,   3.40000000e+00,
        3.50000000e+00,   3.60000000e+00,   3.70000000e+00,
        3.80000000e+00,   3.90000000e+00,   4.00000000e+00,
        4.10000000e+00,   4.20000000e+00,   4.30000000e+00,
        4.40000000e+00,   4.50000000e+00,   4.60000000e+00,
        4.70000000e+00,   4.80000000e+00,   4.90000000e+00,
        5.00000000e+00,   5.10000000e+00,   5.20000000e+00,
        5.30000000e+00,   5.40000000e+00,   5.50000000e+00,
        5.60000000e+00,   5.70000000e+00,   5.80000000e+00,
        5.90000000e+00,   6.00000000e+00,   6.10000000e+00,
        6.20000000e+00,   6.30000000e+00,   6.40000000e+00,
        6.50000000e+00,   6.60000000e+00,   6.70000000e+00,
        6.80000000e+00,   6.90000000e+00,   7.00000000e+00,
        7.10000000e+00,   7.20000000e+00,   7.30000000e+00,
        7.40000000e+00,   7.50000000e+00,   7.60000000e+00,
        7.70000000e+00,   7.80000000e+00,   7.90000000e+00,
        8.00000000e+00,   8.10000000e+00,   8.20000000e+00,
        8.30000000e+00,   8.40000000e+00,   8.50000000e+00,
        8.60000000e+00,   8.70000000e+00,   8.80000000e+00,
        8.90000000e+00,   9.00000000e+00,   9.10000000e+00,
        9.20000000e+00,   9.30000000e+00,   9.40000000e+00,
        9.50000000e+00,   9.60000000e+00,   9.70000000e+00,
        9.80000000e+00,   9.90000000e+00])
```

12.4.4 Valeurs manquantes et factorisation de matrices

Réflexion autour des valeur manquantes et de la factorisation de matrice positive.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```


— *Matrice à coefficients aléatoires* (page 321)
— *Matrice avec des vecteurs colonnes corrélés* (page 322)
— *Matrice identité* (page 324)

— *Matrice identité et représentation spatiale* (page 325)

```
%matplotlib inline
```

Matrice à coefficients aléatoires

On étudie la factorisation d'une matrice à coefficients tout à fait aléatoires qui suivent une loi uniforme sur l'intervalle $[0, 1]$. Essayons sur une petite matrice :

```
from numpy.random import rand
M = rand(3, 3)
M
```

```
array([[ 0.05119593,  0.43722929,  0.9290821 ],
       [ 0.4588466 ,  0.14187813,  0.23762633],
       [ 0.9768084 ,  0.47674026,  0.79044526]])
```

```
from sklearn.decomposition import NMF
mf = NMF(1)
mf.fit_transform(M)
```

```
array([[ 0.67825803],
       [ 0.38030919],
       [ 1.02295362]])
```

La matrice précédente est la matrice W dans le produit WH, la matrice qui suit est H.

```
mf.components_
```

```
array([[ 0.73190904,  0.50765757,  0.92611883]])
```

```
mf.reconstruction_err_ / (M.shape[0] * M.shape[1])
```

```
0.07236890712696428
```

On recalcule l'erreur :

```
d = M - mf.fit_transform(M) @ mf.components_
a = d.ravel()
e = a @ a.T
e ** 0.5 / (M.shape[0] * M.shape[1])
```

```
0.07236890712696428 3
```

```
e.ravel()
```

```
array([ 0.42421796])
```

Et maintenant sur une grande et plus nécessairement carrée :

```
M = rand(300, 10)
mf = NMF(1)
```

(suite sur la page suivante)

(suite de la page précédente)

```
mf.fit_transform(M)
mf.reconstruction_err_ / (M.shape[0] * M.shape[1])
```

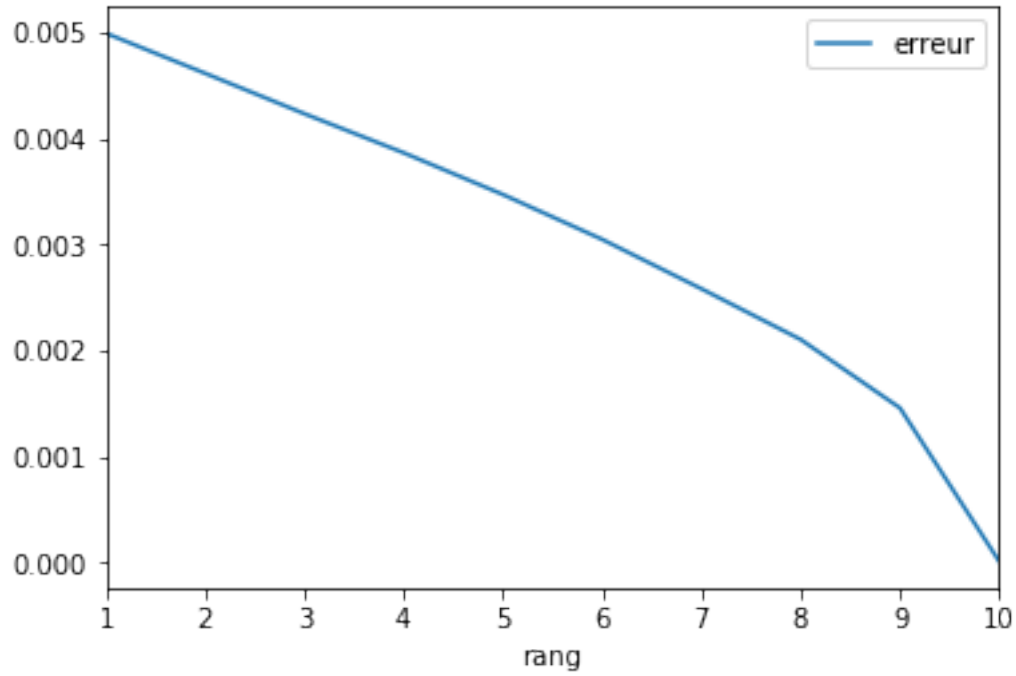
```
0.004996164872801101
```

L'erreur est la même :

```
errs = []
rangs = list(range(1, 11))
for k in rangs:
    mf = NMF(k)
    mf.fit_transform(M)
    e = mf.reconstruction_err_ / (M.shape[0] * M.shape[1])
    errs.append(e)
```

```
import pandas
df = pandas.DataFrame(dict(rang=rangs, erreur=errs))
df.plot(x="rang", y="erreur")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x199924d8668>
```



Matrice avec des vecteurs colonnes corrélés

Supposons maintenant que la matrice précédente M est de rang 3. Pour s'en assurer, on tire une matrice aléalatoire avec 3 vecteurs colonnes et on réplique des colonnes jusqu'à la dimension souhaitée.

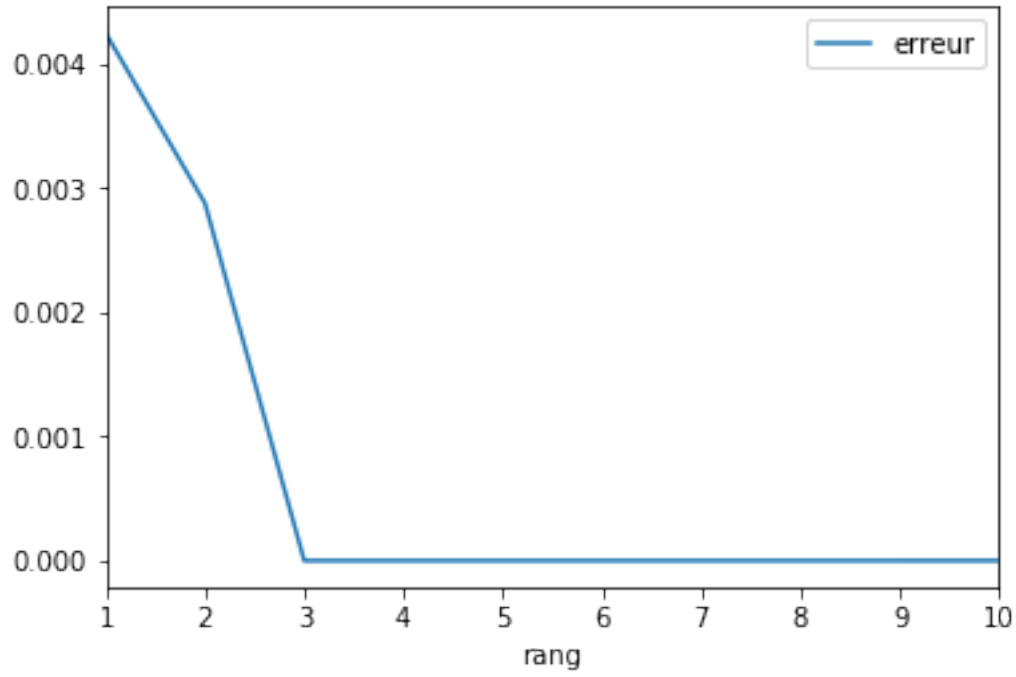
```
from numpy import hstack
M = rand(300, 3)
M = hstack([M, M, M, M[:,:1]])
M.shape
```

```
(300, 10)
```

```
errs = []
rangs = list(range(1, 11))
for k in rangs:
    mf = NMF(k)
    mf.fit_transform(M)
    e = mf.reconstruction_err_ / (M.shape[0] * M.shape[1])
    errs.append(e)
```

```
import pandas
df = pandas.DataFrame(dict(rang=rangs, erreur=errs))
df.plot(x="rang", y="erreur")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x199925d6630>
```



On essaye à nouveausur une matrice un peu plus petite.

```
M = rand(3, 2)
M = hstack([M, M[:,:1]])
M
```

```
array([[ 0.27190312,  0.6497563 ,  0.27190312],
       [ 0.44853292,  0.87097224,  0.44853292],
       [ 0.29424835,  0.65106952,  0.29424835]])
```

```
mf = NMF(2)
mf.fit_transform(M)
```

```
array([[ 0.61835197,  0.        ],
       [ 0.82887888,  0.29866219],
       [ 0.61960446,  0.07743224]])
```

```
mf.components_
```

```
array([[ 0.43972536,  1.05078419,  0.43972536],
       [ 0.28143493,  0.        ,  0.28143493]])
```

La dernière colonne est identique à la première.

Matrice identité

Et maintenant si la matrice M est la matrice identité, que se passe-t-il ?

```
from numpy import identity
M = identity(3)
M
```

```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

```
mf = NMF(1)
mf.fit_transform(M)
```

```
array([[ 0.],
       [ 1.],
       [ 0.]])
```

```
mf.components_
```

```
array([[ 0.,  1.,  0.]])
```

```
mf.reconstruction_err_ ** 2
```

```
2.0000000000000004
```

On essaye avec $k = 2$.

```
mf = NMF(2)
mf.fit_transform(M)
```

```
array([[ 0.        ,  0.        ],
       [ 0.        ,  1.03940448],
       [ 0.95521772,  0.        ]])
```

```
mf.components_
```

```
array([[ 0.        ,  0.        ,  1.04688175],
       [ 0.        ,  0.96208937,  0.        ]])
```

```
mf.reconstruction_err_ ** 2
```

```
1.0
```

Avec des vecteurs normés et indépendants (formant donc une base de l'espace vectoriel), l'algorithme aboutit à une matrice W égale au k premiers vecteurs et oublie les autres.

Matrice identité et représentation spatiale

Pour comprendre un peu mieux ce dernier exemple, il est utile de chercher d'autres solutions dont l'erreur est équivalente.

```
def erreur_mf(M, W, H):
    d = M - W @ H
    a = d.ravel()
    e = a @ a.T
    e ** 0.5 / (M.shape[0] * M.shape[1])
    return e

M = identity(3)
mf = NMF(2)
W = mf.fit_transform(M)
H = mf.components_
erreur_mf(M, W, H)
```

```
1.0
```

```
W
```

```
array([[ 0.        ,  0.        ],
       [ 0.9703523 ,  0.        ],
       [ 0.        ,  1.02721047]])
```

```
H
```

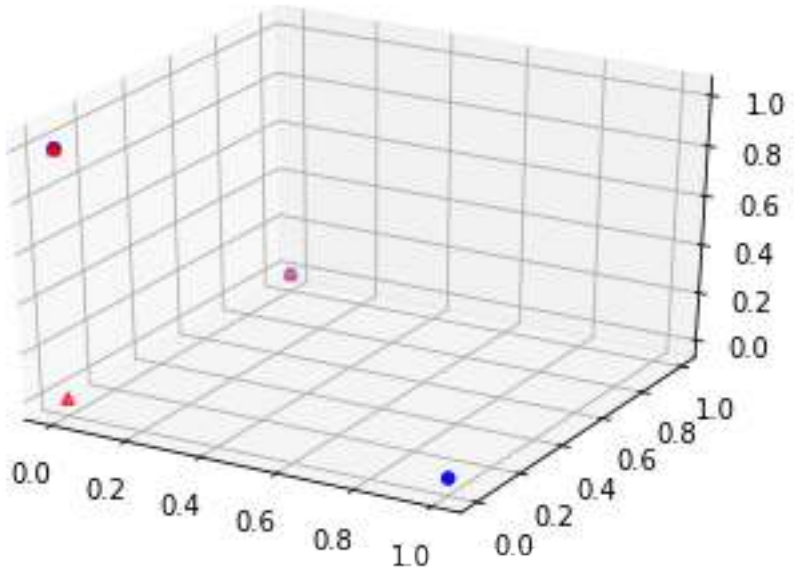
```
array([[ 0.        ,  1.03055354,  0.        ],
       [ 0.        ,  0.        ,  0.97351032]])
```

```
W @ H
```

```
array([[ 0.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
wh = W @ H
ax.scatter(M[:,0], M[:,1], M[:,2], c='b', marker='o', s=20)
ax.scatter(wh[:,0], wh[:,1], wh[:,2], c='r', marker='^')
```

```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x19992d2a5c0>
```



Et si on pose maintenant :

```
import numpy
W = numpy.array([[0.5, 0.5, 0], [0, 0, 1]]).T
H = numpy.array([[1, 1, 0], [0.0, 0.0, 1.0]])
W
```

```
array([[ 0.5,  0. ],
       [ 0.5,  0. ],
       [ 0. ,  1. ]])
```

```
H
```

```
array([[ 1.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

```
W @ H
```

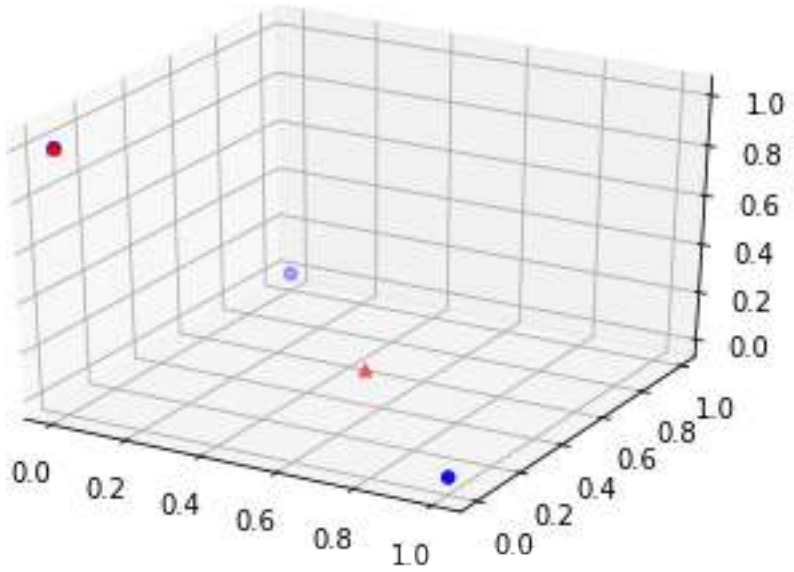
```
array([[ 0.5,  0.5,  0. ],
       [ 0.5,  0.5,  0. ],
       [ 0. ,  0. ,  1. ]])
```

```
erreur_mf(M, W, H)
```

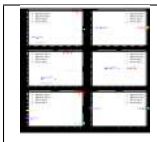
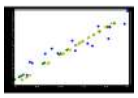
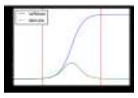
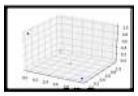
```
1.0
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
wh = W @ H
ax.scatter(M[:,0], M[:,1], M[:,2], c='b', marker='o', s=20)
ax.scatter(wh[:,0], wh[:,1], wh[:,2], c='r', marker='^')
```

```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x19992a2e5f8>
```



On peut voir la matrice M comme un ensemble de n points dans un espace vectoriel. La matrice W est un ensemble de $k < n$ points dans le même espace. La matrice WH, de rang k est une approximation de cet ensemble dans le même espace, c'est aussi n combinaisons linéaires de k points de façon à former n points les plus proches proches de n points de la matrice M.

	*Benchmark* (page 311)	Ce notebook compare différents modèles depuis un notebook.
	*Factorisation et matrice et ACP* (page 314)	Un exemple pour montrer l'équivalence entre l'ACP et une factorisation de matrice.
	*Réseaux de neurones* (page 318)	Réseaux de neurones avec scikit-learn.
	*Valeurs manquantes et factorisation de matrices* (page 320)	Réflexion autour des valeur manquantes et de la factorisation de matrice positive.
	*Voronoï et régression logistique* (page 88)	Le notebook étudie la pertinence d'un modèle de régression logistique dans certaines configurations. Il regarde aussi le diagramme de Voronoï associé à une régression logistique à trois classes. Il donne quelques intuitions sur les modèles que la régression logistique peut résoudre.

12.5 NLP - Natural Language Processing

12.5.1 Completion Trie and metrics

Evaluation of a completion system on wikpedia pages.

Wikipedia titles, uniform

```
(3108490,
 ['A',
  'A & A',
  'A (Airport Express)',
  'A (Ayumi Hamasaki)',
```

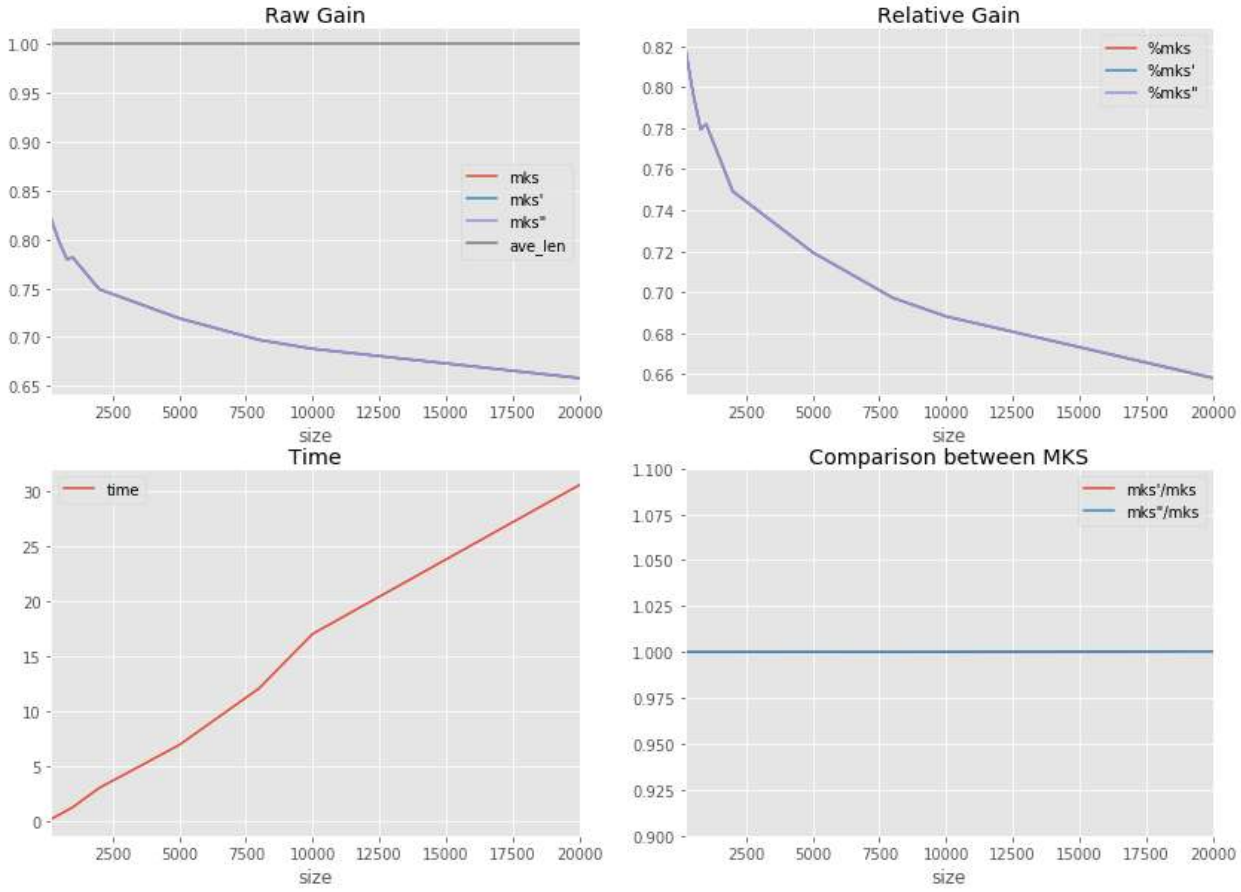
(suite sur la page suivante)

(suite de la page précédente)

```
  "A (Disque d'Ayumi Hamasaki)"],
 ['Fantasy in the sky',
  'Fantasy mythique',
  'Fantasy of manners',
  'Fantasy tennis',
  'Fantasy urbaine'])
```

```
time 0
time: 0.21504800644533353s - nb=200 gain (0.820872274143302, 0.820872274143302, 0.
↪820872274143302, 1.0)
time: 0.6058446756721159s - nb=500 gain (0.7976588628762532, 0.7976588628762532, 0.
↪7976588628762532, 1.0)
time: 1.009366944402156s - nb=800 gain (0.779308535065277, 0.779308535065277, 0.
↪779308535065277, 1.0)
time: 1.2731077523609795s - nb=1000 gain (0.7819106501794998, 0.7819106501794998, 0.
↪7819106501794998, 1.0)
time: 3.0382918326608044s - nb=2000 gain (0.7491075326810025, 0.7491075326810025, 0.
↪7491075326810025, 1.0)
time: 6.941259884811901s - nb=5000 gain (0.7193327903836085, 0.7193534087277493, 0.
↪7193534087277493, 1.0)
time: 12.096078319013222s - nb=8000 gain (0.6971821041145199, 0.6971821041145199, 0.
↪6971821041145199, 1.0)
time: 17.030497306746902s - nb=10000 gain (0.6881011563817098, 0.6881371807341721, 0.
↪6881371807341721, 1.0)
time: 30.55692095058407s - nb=20000 gain (0.6579791591697565, 0.6582343738435791, 0.
↪6582343738435791, 1.0)
```

```
<matplotlib.text.Text at 0x15a756202b0>
```



Reduce the alphabet size

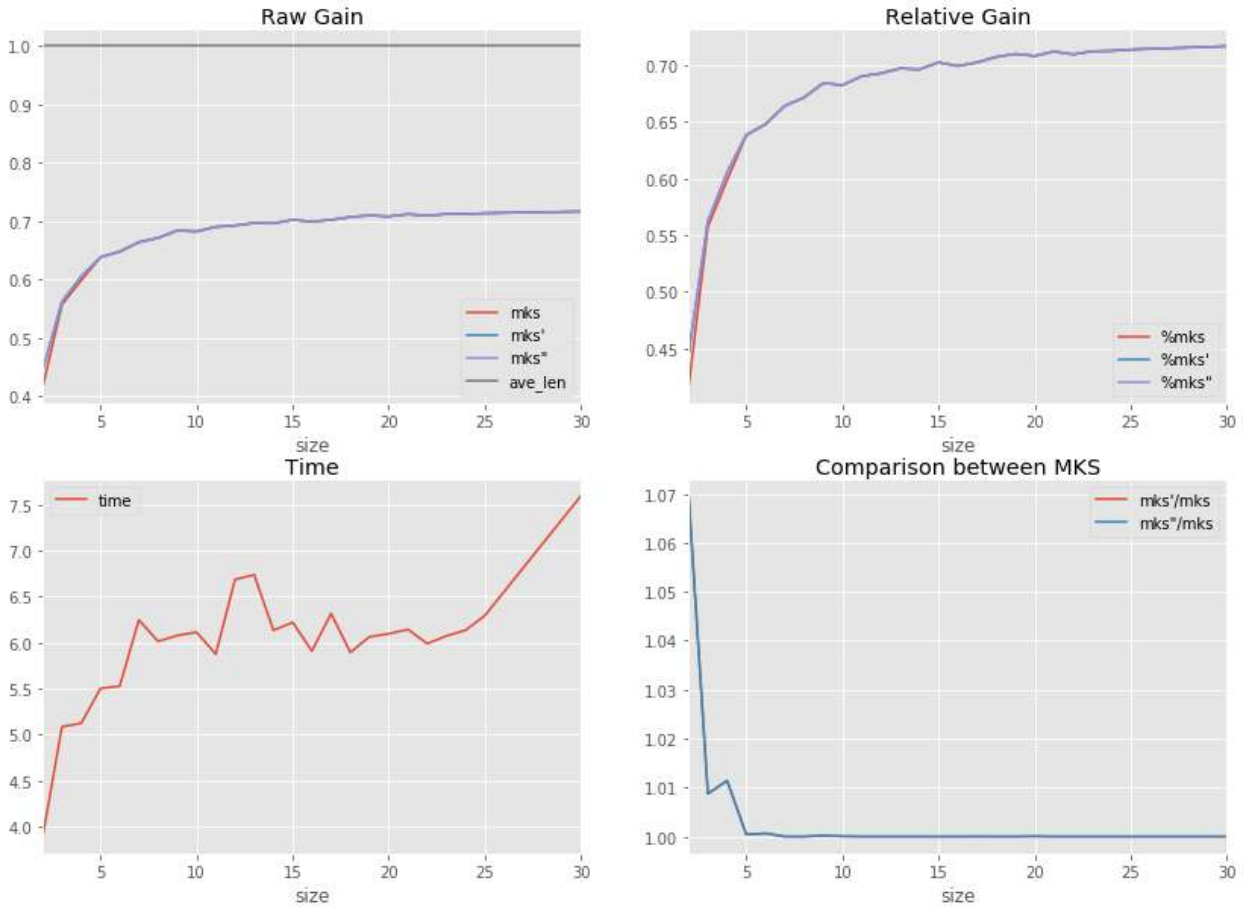
```
time 0
time: 7.59344921135289s - nb=5000 gain (0.716585290640898, 0.716585290640898, 0.
↪716585290640898, 1.0)
time: 3.8923985946166795s - nb=4581 gain (0.41594360086768417, 0.4448874994683378, 0.
↪4448874994683378, 1.0)
time: 5.085379287694195s - nb=4942 gain (0.5571683533987387, 0.5620376961406324, 0.
↪5620376961406324, 1.0)
time: 5.121866923020207s - nb=4974 gain (0.5983975448244626, 0.6052151883090817, 0.
↪6052151883090817, 1.0)
time: 5.501076360438674s - nb=4991 gain (0.6380275314306908, 0.6382847383691052, 0.
↪6382847383691052, 1.0)
time: 5.524899975880544s - nb=4988 gain (0.6475382003395598, 0.6479497864896859, 0.
↪6479497864896859, 1.0)
time: 6.245833967660474s - nb=4997 gain (0.6639308855291576, 0.6639308855291576, 0.
↪6639308855291576, 1.0)
time: 6.012760238038936s - nb=4997 gain (0.6712028636672216, 0.6712028636672216, 0.
↪6712028636672216, 1.0)
time: 6.076252674864918s - nb=4997 gain (0.6838256469329845, 0.6839490681696653, 0.
↪6839490681696653, 1.0)
time: 6.111897439143831s - nb=4999 gain (0.6822851853756178, 0.6823160384634976, 0.
↪6823160384634976, 1.0)
time: 5.873518026578495s - nb=4997 gain (0.6900718921309502, 0.6900718921309502, 0.
↪6900718921309502, 1.0)
```

(suite sur la page suivante)

(suite de la page précédente)

```
time: 6.684070891827105s - nb=4999 gain (0.6925798323648767, 0.6925798323648767, 0.
↪6925798323648767, 1.0)
time: 6.735858496876062s - nb=4997 gain (0.6969017445687994, 0.6969017445687994, 0.
↪6969017445687994, 1.0)
time: 6.1316901553000021s - nb=4999 gain (0.6960868000205542, 0.6960868000205542, 0.
↪6960868000205542, 1.0)
time: 6.2186773552921295s - nb=4999 gain (0.7022574175965309, 0.7022574175965309, 0.
↪7022574175965309, 1.0)
time: 5.907541621836572s - nb=4998 gain (0.6991010265166325, 0.6991010265166325, 0.
↪6991010265166325, 1.0)
time: 6.314328893328825s - nb=4999 gain (0.7022368488712789, 0.7022471332339055, 0.
↪7022471332339055, 1.0)
time: 5.892940837380593s - nb=4998 gain (0.7070717459272685, 0.7070717459272685, 0.
↪7070717459272685, 1.0)
time: 6.061792582734597s - nb=4999 gain (0.7097547179513399, 0.7097547179513399, 0.
↪7097547179513399, 1.0)
time: 6.094942944771901s - nb=4999 gain (0.7079858075795616, 0.7080166606674415, 0.
↪7080166606674415, 1.0)
time: 6.141645954818159s - nb=4999 gain (0.7118732966524257, 0.7118732966524257, 0.
↪7118732966524257, 1.0)
time: 5.9873731844709255s - nb=4999 gain (0.7094359027099135, 0.7094359027099135, 0.
↪7094359027099135, 1.0)
time: 6.0718454556808865s - nb=4999 gain (0.7120892682675833, 0.7120892682675833, 0.
↪7120892682675833, 1.0)
time: 6.133951068150054s - nb=4999 gain (0.7124903584100222, 0.7124903584100222, 0.
↪7124903584100222, 1.0)
time: 6.292655432947868s - nb=4999 gain (0.713611353936324, 0.713611353936324, 0.
↪713611353936324, 1.0)
```

```
<matplotlib.text.Text at 0x15a74bf40b8>
```

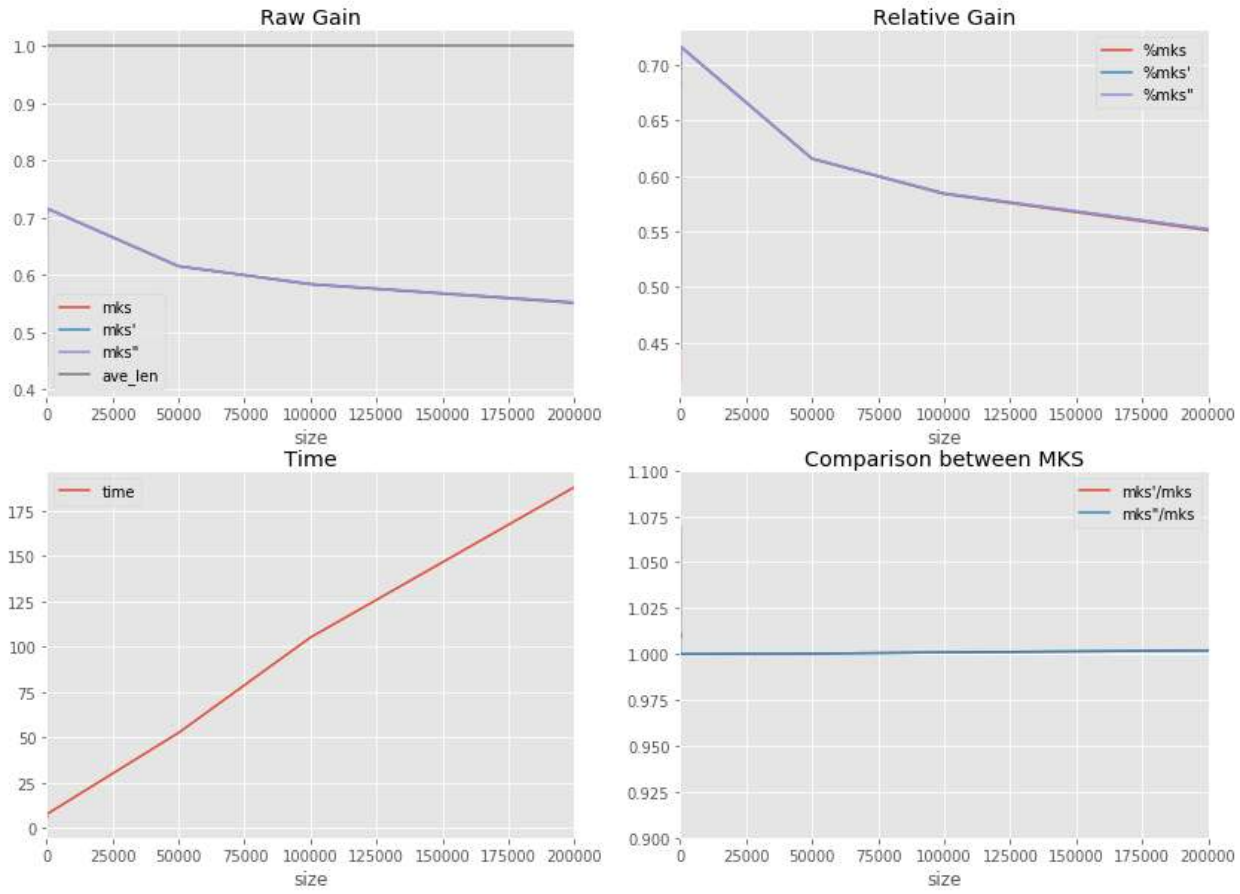


Wikipedia titles, uniform, longer test

```
time 0
time: 52.057980205573585s - nb=50000 gain (0.6162242515637921, 0.616305075104518, 0.
↪616305075104518, 1.0)
```

```
time 0
time: 52.51158252780897s - nb=50000 gain (0.615225173328998, 0.6153599275825006, 0.
↪6153599275825006, 1.0)
time: 105.0721302614229s - nb=100000 gain (0.5836043296652512, 0.5841384772496148, 0.
↪5841384772496148, 1.0)
time: 187.86111486480695s - nb=200000 gain (0.5507786166438062, 0.5518801462043321, 0.
↪5518801462043321, 1.0)
```

```
<matplotlib.text.Text at 0x15a132f8be0>
```



12.5.2 Completion profiling

Profiling avec cProfile, memory_profiler, line_profiler, pyinstrument, snakeviz.

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

— *vprof* (page 343)

Setup

Function to profile

```
from mlstatpy.nlp.completion import CompletionTrieNode

def gain_dynamique_moyen_par_mot(queries, weights):
    per = list(zip(weights, queries))
    total = sum(weights) * 1.0
    res = []
    trie = CompletionTrieNode.build([(None, q) for _, q in per])
    trie.precompute_stat()
    trie.update_stat_dynamic()
    wks = [(w, p, len(w) - trie.min_keystroke0(w)[0]) for p, w in per]
    wks_dyn = [(w, p, len(w) - trie.min_dynamic_keystroke(w)[0])
               for p, w in per]
    wks_dyn2 = [(w, p, len(w) - trie.min_dynamic_keystroke2(w)[0])
                for p, w in per]
    gain = sum(g * p / total for w, p, g in wks)
    gain_dyn = sum(g * p / total for w, p, g in wks_dyn)
    gain_dyn2 = sum(g * p / total for w, p, g in wks_dyn2)
    ave_length = sum(len(w) * p / total for p, w in per)
    return gain, gain_dyn, gain_dyn2, ave_length
```

Data

```
from mlstatpy.data.wikipedia import download_titles
file_titles = download_titles(country='fr')
```

```
len(file_titles)
```

```
33
```

```
from mlstatpy.data.wikipedia import enumerate_titles
list_titles = list(sorted(set(_ for _ in enumerate_titles(file_titles) if 'A' <= _[0]
↪<= 'Z')))
```

```
import random
sample1000 = random.sample(list_titles, 1000)
with open("sample1000.txt", "w", encoding="utf-8") as f:
    f.write("\n".join(sample1000))
```

Standard modules

cProfile

```
import cProfile, io, pstats, os

def toprofile0(lines):
    gain_dynamique_moyen_par_mot(lines, [1.0] * len(lines))

def doprofile(lines, filename):
    pr = cProfile.Profile()
    pr.enable()
    toprofile0(lines)
    pr.disable()
    s = io.StringIO()
    ps = pstats.Stats(pr, stream=s).sort_stats('cumulative')
    ps.print_stats()
    rem = os.path.normpath(os.path.join(os.getcwd(), "..", "..", ".."))
    res = s.getvalue().replace(rem, "")
    ps.dump_stats(filename)
    return res
```

```
r = doprofile(sample1000, "completion.prof")
print(r)
```

```
      1225828 function calls in 1.261 seconds
Ordered by: cumulative time
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000    1.261    1.261 <ipython-input-8-d2b6d910fdd7>
 ↪:3(toprofile0)
     1    0.000    0.000    1.261    1.261 <ipython-input-3-684f3e860cf5>
 ↪:3(gain_dynamique_moyen_par_mot)
     1    0.129    0.129    0.916    0.916 srcmlstatpynlpcompletion.
 ↪py:416(precompute_stat)
 15148    0.260    0.000    0.662    0.000 srcmlstatpynlpcompletion.
 ↪py:504(merge_completions)
 15148    0.277    0.000    0.281    0.000 {built-in method builtins.__build_
 ↪class__}
     1    0.051    0.051    0.216    0.216 srcmlstatpynlpcompletion.
 ↪py:451(update_stat_dynamic)
     1    0.087    0.087    0.107    0.107 srcmlstatpynlpcompletion.
 ↪py:203(build)
 16147    0.048    0.000    0.090    0.000 srcmlstatpynlpcompletion.
 ↪py:556(update_dynamic_minimum_keystroke)
 34403    0.061    0.000    0.070    0.000 srcmlstatpynlpcompletion.py:524(
 ↪<listcomp>)
 34127    0.019    0.000    0.037    0.000 {built-in method builtins.all}
 16147    0.026    0.000    0.037    0.000 srcmlstatpynlpcompletion.
 ↪py:625(init_dynamic_minimum_keystroke)
 16147    0.029    0.000    0.034    0.000 srcmlstatpynlpcompletion.
 ↪py:589(second_step)
298714    0.030    0.000    0.030    0.000 {built-in method builtins.len}
 15148    0.023    0.000    0.029    0.000 srcmlstatpynlpcompletion.
 ↪py:543(update_minimum_keystroke)
105060    0.028    0.000    0.028    0.000 {built-in method builtins.hasattr}
 15149    0.003    0.000    0.027    0.000 {method 'extend' of 'collections.
 ↪deque' objects}
 16148    0.018    0.000    0.027    0.000 srcmlstatpynlpcompletion.
 ↪py:97(unsorted_iter)
```

```
  1001    0.016    0.000    0.023    0.000 srcmlstatpynlpcompletion.
↪py:132(leaves)
 15148    0.018    0.000    0.023    0.000 {built-in method builtins.sorted}
 93541    0.022    0.000    0.022    0.000 srcmlstatpynlpcompletion.py:436(
↪<genexpr>)
  3000    0.013    0.000    0.014    0.000 srcmlstatpynlpcompletion.
↪py:258(find)
 16147    0.011    0.000    0.013    0.000 srcmlstatpynlpcompletion.py:20(__
↪init__)
109867    0.013    0.000    0.013    0.000 {method 'values' of 'dict' objects}
 22498    0.009    0.000    0.009    0.000 {built-in method builtins.min}
 45444    0.009    0.000    0.009    0.000 {method 'extend' of 'list' objects}
     1    0.001    0.001    0.009    0.009 <ipython-input-3-684f3e860cf5>:13(
↪<listcomp>)
  1000    0.001    0.000    0.008    0.000 srcmlstatpynlpcompletion.
↪py:383(min_dynamic_keystroke2)
 48441    0.007    0.000    0.007    0.000 {method 'pop' of 'list' objects}
 49552    0.007    0.000    0.007    0.000 {method 'append' of 'list' objects}
 19255    0.007    0.000    0.007    0.000 {built-in method builtins.max}
 52271    0.006    0.000    0.006    0.000 {method 'popleft' of 'collections.
↪deque' objects}
     1    0.001    0.001    0.006    0.006 <ipython-input-3-684f3e860cf5>:10(
↪<listcomp>)
 35125    0.005    0.000    0.005    0.000 {method 'append' of 'collections.
↪deque' objects}
     1    0.001    0.001    0.005    0.005 <ipython-input-3-684f3e860cf5>:11(
↪<listcomp>)
 16146    0.005    0.000    0.005    0.000 srcmlstatpynlpcompletion.py:54(_
↪add)
  1000    0.001    0.000    0.005    0.000 srcmlstatpynlpcompletion.
↪py:322(min_keystroke0)
  1000    0.001    0.000    0.005    0.000 srcmlstatpynlpcompletion.
↪py:353(min_dynamic_keystroke)
 16148    0.005    0.000    0.005    0.000 srcmlstatpynlpcompletion.py:518(
↪<genexpr>)
 15148    0.004    0.000    0.004    0.000 srcmlstatpynlpcompletion.
↪py:509(Fake)
 15148    0.004    0.000    0.004    0.000 srcmlstatpynlpcompletion.py:512(
↪<listcomp>)
 30296    0.003    0.000    0.003    0.000 {method 'items' of 'dict' objects}
 17147    0.002    0.000    0.002    0.000 {built-in method builtins.
↪isinstance}
     5    0.000    0.000    0.001    0.000 {built-in method builtins.sum}
     1    0.000    0.000    0.000    0.000 <ipython-input-3-684f3e860cf5>:7(
↪<listcomp>)
  1001    0.000    0.000    0.000    0.000 <ipython-input-3-684f3e860cf5>:18(
↪<genexpr>)
  1001    0.000    0.000    0.000    0.000 <ipython-input-3-684f3e860cf5>:15(
↪<genexpr>)
  1001    0.000    0.000    0.000    0.000 <ipython-input-3-684f3e860cf5>:16(
↪<genexpr>)
  1001    0.000    0.000    0.000    0.000 <ipython-input-3-684f3e860cf5>:17(
↪<genexpr>)
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.
```

```
↪Profiler' objects}
```

Others informations when profiling

memory_profiler

See memory_profiler [594].

```
from memory_profiler import profile
%load_ext memory_profiler
```

```
%memit toprofile0(sample1000)
```

```
peak memory: 412.05 MiB, increment: 15.60 MiB
```

```
from io import StringIO
st = StringIO()
@profile(stream=st)
def toprofile(lines):
    gain_dynamique_moyen_par_mot(lines, [1.0] * len(lines))
toprofile(sample1000)
```

```
ERROR: Could not find file <ipython-input-12-7512ff5cdee3>
NOTE: %mprun can only be used on functions defined in physical files, and not in the␣
↪IPython environment.
```

```
%%file temp_mem_profile.py

from mlstatpy.nlp.completion import CompletionTrieNode
from memory_profiler import profile

@profile(precision=4)
def gain_dynamique_moyen_par_mot(queries, weights):
    per = list(zip(weights, queries))
    total = sum(weights) * 1.0
    res = []
    trie = CompletionTrieNode.build([(None, q) for _, q in per])
    trie.precompute_stat()
    trie.update_stat_dynamic()
    wks = [(w, p, len(w) - trie.min_keystroke0(w)[0]) for p, w in per]
    wks_dyn = [(w, p, len(w) - trie.min_dynamic_keystroke(w)[0])
               for p, w in per]
    wks_dyn2 = [(w, p, len(w) - trie.min_dynamic_keystroke2(w)[0])
                for p, w in per]
    gain = sum(g * p / total for w, p, g in wks)
    gain_dyn = sum(g * p / total for w, p, g in wks_dyn)
    gain_dyn2 = sum(g * p / total for w, p, g in wks_dyn2)
    ave_length = sum(len(w) * p / total for p, w in per)
    return gain, gain_dyn, gain_dyn2, ave_length

@profile(precision=4)
def toprofile():
    with open("sample1000.txt", "r", encoding="utf-8") as f:
```

(suite sur la page suivante)

594. https://pypi.python.org/pypi/memory_profiler/0.41

(suite de la page précédente)

```
        lines = [_.strip("\n\r ") for _ in f.readlines()]
    gain_dynamique_moyen_par_mot(lines, [1.0] * len(lines))
toprofile()
```

```
Overwriting temp_mem_profile.py
```

```
import sys
cmd = sys.executable
from pyquickhelper.loghelper import run_cmd
cmd += " -m memory_profiler temp_mem_profile.py"
out, err = run_cmd(cmd, wait=True)
print(out)
```

```
Filename: temp_mem_profile.py
Line #    Mem usage    Increment   Line Contents
================================================
     5  56.0625 MiB  56.0625 MiB   @profile(precision=4)
     6                             def gain_dynamique_moyen_par_mot(queries,␣
 ↪weights):
     7  56.0625 MiB   0.0000 MiB       per = list(zip(weights, queries))
     8  56.0625 MiB   0.0000 MiB       total = sum(weights) * 1.0
     9  56.0625 MiB   0.0000 MiB       res = []
    10  62.2500 MiB   6.1875 MiB       trie = CompletionTrieNode.build([(None,
 ↪ q) for _, q in per])
    11  69.5625 MiB   7.3125 MiB       trie.precompute_stat()
    12  78.7695 MiB   9.2070 MiB       trie.update_stat_dynamic()
    13  78.7734 MiB   0.0039 MiB       wks = [(w, p, len(w) - trie.min_
 ↪keystroke0(w)[0]) for p, w in per]
    14  78.7969 MiB   0.0234 MiB       wks_dyn = [(w, p, len(w) - trie.min_
 ↪dynamic_keystroke(w)[0])
    15  78.7969 MiB   0.0000 MiB                  for p, w in per]
    16  78.7969 MiB   0.0000 MiB       wks_dyn2 = [(w, p, len(w) - trie.min_
 ↪dynamic_keystroke2(w)[0])
    17  78.7969 MiB   0.0000 MiB                   for p, w in per]
    18  78.7969 MiB   0.0000 MiB       gain = sum(g * p / total for w, p, g␣
 ↪in wks)
    19  78.7969 MiB   0.0000 MiB       gain_dyn = sum(g * p / total for w, p,␣
 ↪g in wks_dyn)
    20  78.7969 MiB   0.0000 MiB       gain_dyn2 = sum(g * p / total for w, p,
 ↪ g in wks_dyn2)
    21  78.7969 MiB   0.0000 MiB       ave_length = sum(len(w) * p / total␣
 ↪for p, w in per)
    22  78.7969 MiB   0.0000 MiB       return gain, gain_dyn, gain_dyn2, ave_
 ↪length

Filename: temp_mem_profile.py
Line #    Mem usage    Increment   Line Contents
================================================
    24  55.9570 MiB  55.9570 MiB   @profile(precision=4)
    25                             def toprofile():
    26  55.9570 MiB   0.0000 MiB       with open("sample1000.txt", "r",␣
 ↪encoding="utf-8") as f:
    27  56.0625 MiB   0.1055 MiB           lines = [_.strip("nr ") for _ in f.
```

```
↪readlines()]
    28   78.7969 MiB   22.7344 MiB       gain_dynamique_moyen_par_mot(lines, [1.
↪0] * len(lines))
```

line_profiler

See line_profiler [595].

```
def lineprofile(lines):
    gain_dynamique_moyen_par_mot(lines, [1.0] * len(lines))
```

```
from mlstatpy.nlp.completion import CompletionTrieNode
```

```
from line_profiler import LineProfiler
prof = LineProfiler()
prof.add_function(gain_dynamique_moyen_par_mot)
prof.add_function(CompletionTrieNode.precompute_stat)
prof.run("lineprofile(sample1000)")
st = io.StringIO()
prof.print_stats(stream=st)
rem = os.path.normpath(os.path.join(os.getcwd(), "..", "..", ".."))
res = st.getvalue().replace(rem, "")
print(res)
```

```
Timer unit: 3.95062e-07 s
Total time: 3.3536 s
File: <ipython-input-3-684f3e860cf5>
Function: gain_dynamique_moyen_par_mot at line 3
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     3                                           def gain_dynamique_moyen_par_
↪mot(queries, weights):
     4         1        393.0    393.0      0.0      per = list(zip(weights,␣
↪queries))
     5         1         23.0     23.0      0.0      total = sum(weights) * 1.
↪0
     6         1          4.0      4.0      0.0      res = []
     7         1     379526.0 379526.0      4.5      trie =␣
↪CompletionTrieNode.build([(None, q) for _, q in per])
     8         1    6658020.0 6658020.0     78.4       trie.precompute_stat()
     9         1    1241489.0 1241489.0     14.6       trie.update_stat_
↪dynamic()
    10         1      64879.0  64879.0      0.8      wks = [(w, p, len(w) -␣
↪trie.min_keystroke0(w)[0]) for p, w in per]
    11         1         16.0     16.0      0.0      wks_dyn = [(w, p, len(w)␣
↪- trie.min_dynamic_keystroke(w)[0])
    12         1      58635.0  58635.0      0.7                 for p, w in␣
↪per]
    13         1         10.0     10.0      0.0      wks_dyn2 = [(w, p,␣
↪len(w) - trie.min_dynamic_keystroke2(w)[0])
    14         1      80850.0  80850.0      1.0                  for p, w in␣
↪per]
```

595. https://github.com/rkern/line_profiler

```
    15         1       1318.0   1318.0      0.0      gain = sum(g * p / total␣
↪for w, p, g in wks)
    16         1       1221.0   1221.0      0.0      gain_dyn = sum(g * p /␣
↪total for w, p, g in wks_dyn)
    17         1       1073.0   1073.0      0.0      gain_dyn2 = sum(g * p /␣
↪total for w, p, g in wks_dyn2)
    18         1       1349.0   1349.0      0.0      ave_length = sum(len(w)␣
↪* p / total for p, w in per)
    19         1          4.0      4.0      0.0      return gain, gain_dyn,␣
↪gain_dyn2, ave_length
Total time: 2.19801 s
File: srcmlstatpynlpcompletion.py
Function: precompute_stat at line 416
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   416                                           def precompute_
↪stat(self):
   417                                               """
   418                                               computes and stores␣
↪list of completions for each node,
   419                                               computes mks
   420
   421                                               @param      clean  ␣
↪clean stat
   422                                               """
   423         1          5.0      5.0      0.0          stack = deque()
   424         1      78759.0  78759.0      1.4          stack.extend(self.
↪leaves())
   425     52272     168548.0      3.2      3.0          while len(stack) > 0:
   426     52271     169512.0      3.2      3.0              pop = stack.
↪popleft()
   427     52271     148123.0      2.8      2.7              if pop.stat is␣
↪not None:
   428     17145      42365.0      2.5      0.8                  continue
   429     35126     105270.0      3.0      1.9              if not pop.
↪children:
   430       999       2545.0      2.5      0.0                  pop.stat =␣
↪CompletionTrieNode._Stat()
   431       999       5598.0      5.6      0.1                  pop.stat.
↪completions = []
   432       999       2812.0      2.8      0.1                  pop.stat.
↪mks0 = len(pop.value)
   433       999       2370.0      2.4      0.0                  pop.stat.
↪mks0_ = len(pop.value)
   434       999       2233.0      2.2      0.0                  if pop.
↪parent is not None:
   435       999       2339.0      2.3      0.0                      stack.
↪append(pop.parent)
   436     34127     397661.0     11.7      7.1              elif all(v.stat␣
↪is not None for v in pop.children.values()):
   437     15148      55784.0      3.7      1.0                  pop.stat =␣
↪CompletionTrieNode._Stat()
   438     15148      48358.0      3.2      0.9                  if pop.leave:
   439         1          5.0      5.0      0.0                      pop.stat.
```

```
↪mks0 = len(pop.value)
   440         1          5.0      5.0      0.0                        pop.stat.
↪mks0_ = len(pop.value)
   441     15148      76740.0      5.1      1.4                    stack.
↪extend(pop.children.values())
   442     15148    3717484.0    245.4     66.8                    pop.stat.
↪merge_completions(pop.value, pop.children.values())
   443     15148      62928.0      4.2      1.1                    pop.stat.
↪next_nodes = pop.children
   444     15148     292243.0     19.3      5.3                    pop.stat.
↪update_minimum_keystroke(len(pop.value))
   445     15148      55688.0      3.7      1.0                    if pop.
↪parent is not None:
   446     15147      64826.0      4.3      1.2                        stack.
↪append(pop.parent)
   447                                                          else:
   448                                                              # we'll do␣
↪it again later
   449     18979      61515.0      3.2      1.1                    stack.
↪append(pop)
```

Static Visualization

gprof2dot

See gprof2dot [596].

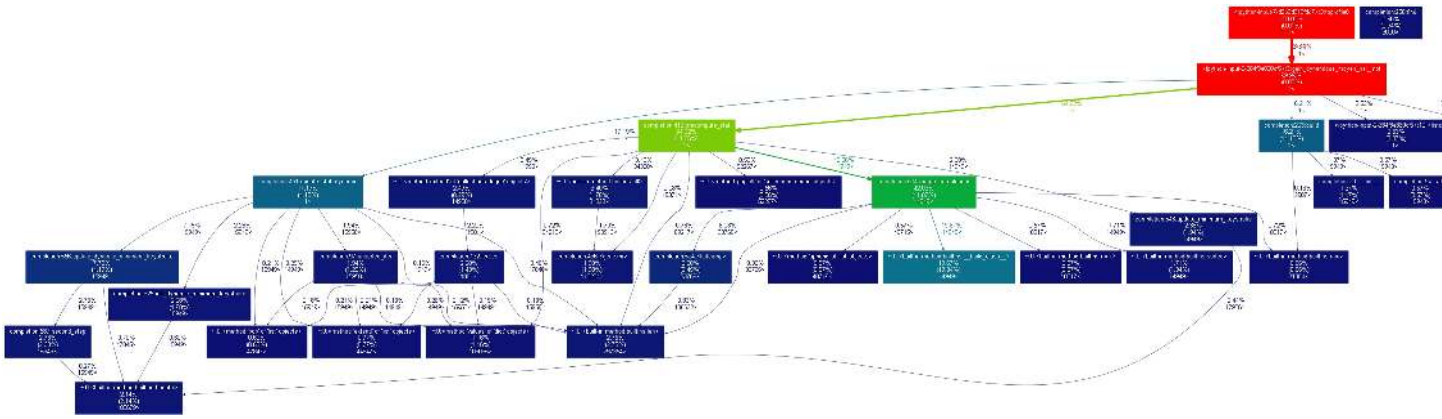
```
import gprof2dot
import sys
sys.argv=["", "-f", "pstats", "completion.prof", "-o", "completion.dot"]
gprof2dot.main()
```

```
from pyquickhelper.helpgen.conf_path_tools import find_graphviz_dot
dot = find_graphviz_dot()
```

```
from pyquickhelper.loghelper import run_cmd
out, err = run_cmd('"{0}" completion.dot -Tpng -ocompletion.png'.format(dot),␣
↪wait=True)
print(out)
```

```
from pyquickhelper.helpgen import NbImage
name = "completion.png"
if os.path.exists("images/completion.jpg"):
    # replacing by jpg because latex conversion does not like png with transparency
    name = "images/completion.jpg"
NbImage(name, width=800)
```

596. https://github.com/jrfonseca/gprof2dot



pyinstrument

See pyinstrument [597].

```
from pyinstrument import Profiler

profiler = Profiler(use_signal=False)
profiler.start()

toprofile0(sample1000)

profiler.stop()
out = profiler.output_text(unicode=False, color=False)
print(out.replace("\\", "/"))
```

```
__main__:3: DeprecationWarning: use_signal is deprecated and should no longer be used.
```

```
1.414 gain_dynamique_moyen_par_mot  <ipython-input-3-684f3e860cf5>:3
|- 1.050 precompute_stat  mlstatpy/nlp/completion.py:416
|  |- 0.846 merge_completions  mlstatpy/nlp/completion.py:504
|  |  `- 0.071 <listcomp>  mlstatpy/nlp/completion.py:524
|  |- 0.027 <genexpr>  mlstatpy/nlp/completion.py:436
|  |- 0.026 leaves  mlstatpy/nlp/completion.py:132
|  `- 0.015 update_minimum_keystroke  mlstatpy/nlp/completion.py:543
|- 0.235 update_stat_dynamic  mlstatpy/nlp/completion.py:451
|  |- 0.119 update_dynamic_minimum_keystroke  mlstatpy/nlp/completion.py:556
|  |  `- 0.037 second_step  mlstatpy/nlp/completion.py:589
|  |- 0.041 init_dynamic_minimum_keystroke  mlstatpy/nlp/completion.py:625
|  `- 0.020 unsorted_iter  mlstatpy/nlp/completion.py:97
`- 0.096 build  mlstatpy/nlp/completion.py:203
   `- 0.019 __init__  mlstatpy/nlp/completion.py:20
```

Javascript Visualization

SnakeViz

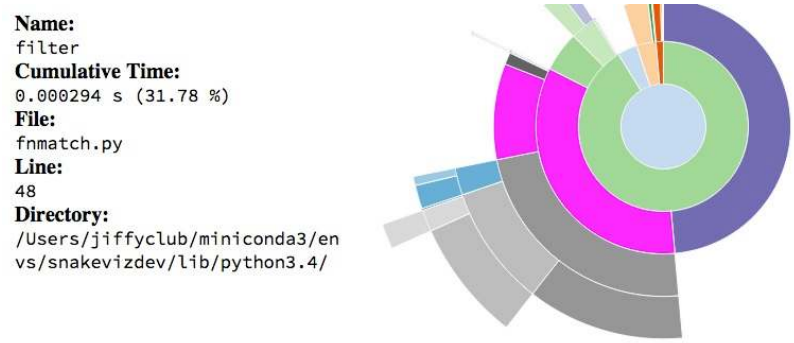
```
%load_ext snakeviz
```

597. https://github.com/joerick/pyinstrument

L'instruction qui suit lance l'explorateur par défaut avec les données du profilage.

```
# %snakeviz toprofile0(sample1000)
```

```
from pyquickhelper.helpgen import NbImage
NbImage("images/func_info.jpg", width=400)
```



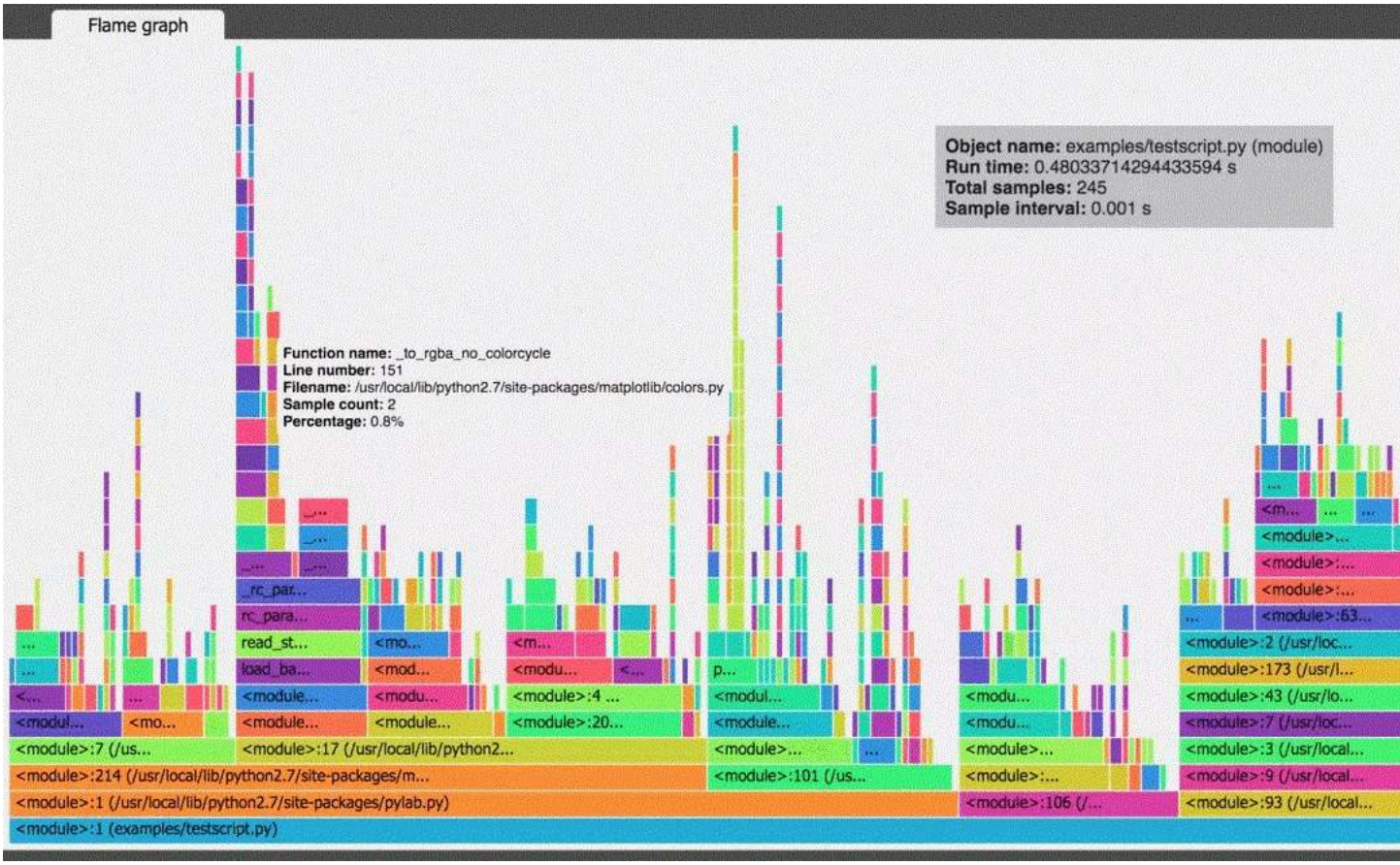
vprof

See vprof[598].

```
from vprof import profiler

# needs to be run from a file not from a notebook
# profiler.run(toprofile0, 'cmh', args=(sample1000,), host='localhost', port=8000)
```

```
from pyquickhelper.helpgen import NbImage
NbImage("images/vprof.jpg", width=800)
```

598. https://github.com/nvdv/vprof



12.5.3 Complétion

Comparaion de plusieurs algorithmes pour implémenter un système de complétion.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```


Tester des idées

Meilleur ordre pour a, ab, abc, abcd

```
from mlstatpy.nlp.completion import CompletionTrieNode
import itertools
queries = ['a', 'ab', 'abc', 'abcd']
for per in itertools.permutations(queries):
    trie = CompletionTrieNode.build([(None, w) for w in per])
    gain = sum( len(w) - trie.min_keystroke(w)[0] for w in per)
    print(gain, "ordre", per)
```

```
0 ordre ('a', 'ab', 'abc', 'abcd')
1 ordre ('a', 'ab', 'abcd', 'abc')
1 ordre ('a', 'abc', 'ab', 'abcd')
2 ordre ('a', 'abc', 'abcd', 'ab')
2 ordre ('a', 'abcd', 'ab', 'abc')
2 ordre ('a', 'abcd', 'abc', 'ab')
1 ordre ('ab', 'a', 'abc', 'abcd')
2 ordre ('ab', 'a', 'abcd', 'abc')
2 ordre ('ab', 'abc', 'a', 'abcd')
3 ordre ('ab', 'abc', 'abcd', 'a')
3 ordre ('ab', 'abcd', 'a', 'abc')
3 ordre ('ab', 'abcd', 'abc', 'a')
2 ordre ('abc', 'a', 'ab', 'abcd')
3 ordre ('abc', 'a', 'abcd', 'ab')
2 ordre ('abc', 'ab', 'a', 'abcd')
3 ordre ('abc', 'ab', 'abcd', 'a')
4 ordre ('abc', 'abcd', 'a', 'ab')
4 ordre ('abc', 'abcd', 'ab', 'a')
3 ordre ('abcd', 'a', 'ab', 'abc')
3 ordre ('abcd', 'a', 'abc', 'ab')
3 ordre ('abcd', 'ab', 'a', 'abc')
3 ordre ('abcd', 'ab', 'abc', 'a')
4 ordre ('abcd', 'abc', 'a', 'ab')
4 ordre ('abcd', 'abc', 'ab', 'a')
```

Meilleur ordre pour a, ab, abc, abcd, edf, edfh

```
queries = ['a', 'ab', 'abc', 'abcd', 'edf', 'edfh']
res = []
for per in itertools.permutations(queries):
    trie = CompletionTrieNode.build([(None, w) for w in per])
    gain = sum( len(w) - trie.min_keystroke(w)[0] for w in per)
    res.append((gain, "ordre", per))
res.sort(reverse=True)
for r in res[:30]:
    print(r)
```

```
(6, 'ordre', ('edfh', 'edf', 'abcd', 'abc', 'ab', 'a'))
(6, 'ordre', ('edfh', 'edf', 'abcd', 'abc', 'a', 'ab'))
(6, 'ordre', ('edfh', 'edf', 'abcd', 'ab', 'abc', 'a'))
(6, 'ordre', ('edfh', 'edf', 'abcd', 'ab', 'a', 'abc'))
(6, 'ordre', ('edfh', 'edf', 'abcd', 'a', 'abc', 'ab'))
(6, 'ordre', ('edfh', 'edf', 'abcd', 'a', 'ab', 'abc'))
(6, 'ordre', ('edfh', 'edf', 'abc', 'abcd', 'ab', 'a'))
```

(suite sur la page suivante)

(suite de la page précédente)

```
(6, 'ordre', ('edfh', 'edf', 'abc', 'abcd', 'a', 'ab'))
(6, 'ordre', ('edf', 'edfh', 'abcd', 'abc', 'ab', 'a'))
(6, 'ordre', ('edf', 'edfh', 'abcd', 'abc', 'a', 'ab'))
(6, 'ordre', ('edf', 'edfh', 'abcd', 'ab', 'abc', 'a'))
(6, 'ordre', ('edf', 'edfh', 'abcd', 'ab', 'a', 'abc'))
(6, 'ordre', ('edf', 'edfh', 'abcd', 'a', 'abc', 'ab'))
(6, 'ordre', ('edf', 'edfh', 'abcd', 'a', 'ab', 'abc'))
(6, 'ordre', ('edf', 'edfh', 'abc', 'abcd', 'ab', 'a'))
(6, 'ordre', ('edf', 'edfh', 'abc', 'abcd', 'a', 'ab'))
(6, 'ordre', ('abcd', 'abc', 'edfh', 'edf', 'ab', 'a'))
(6, 'ordre', ('abcd', 'abc', 'edfh', 'edf', 'a', 'ab'))
(6, 'ordre', ('abcd', 'abc', 'edfh', 'ab', 'edf', 'a'))
(6, 'ordre', ('abcd', 'abc', 'edfh', 'ab', 'a', 'edf'))
(6, 'ordre', ('abcd', 'abc', 'edfh', 'a', 'edf', 'ab'))
(6, 'ordre', ('abcd', 'abc', 'edfh', 'a', 'ab', 'edf'))
(6, 'ordre', ('abcd', 'abc', 'edf', 'edfh', 'ab', 'a'))
(6, 'ordre', ('abcd', 'abc', 'edf', 'edfh', 'a', 'ab'))
(6, 'ordre', ('abcd', 'abc', 'edf', 'ab', 'edfh', 'a'))
(6, 'ordre', ('abcd', 'abc', 'edf', 'ab', 'a', 'edfh'))
(6, 'ordre', ('abcd', 'abc', 'edf', 'a', 'edfh', 'ab'))
(6, 'ordre', ('abcd', 'abc', 'edf', 'a', 'ab', 'edfh'))
(6, 'ordre', ('abcd', 'abc', 'ab', 'edfh', 'edf', 'a'))
(6, 'ordre', ('abcd', 'abc', 'ab', 'edfh', 'a', 'edf'))
```

Influence du poids

```
queries = ['actuellement', 'actualité', 'acte', 'actes']
weights = [1, 1, 1, 2]
total = sum(weights) * 1.0 / len(queries)
res = []
for per in itertools.permutations(zip(queries, weights)):
    trie = CompletionTrieNode.build([(None, w) for w, p in per])
    wks = [(w, p, len(w)-trie.min_keystroke(w)[0]) for w, p in per]
    gain = sum( g*p/total for w, p, g in wks)
    res.append((gain, wks))
res.sort(reverse=True)
for r in res:
    print("{0:3.4} - {1}".format(r[0], " | ".join("%s p=%1.1f g=%1.1f" % _ for _ in␣
↪r[1])))
```

```
19.2 - actes p=2.0 g=4.0 | actuellement p=1.0 g=10.0 | acte p=1.0 g=1.0 | actualité␣
↪p=1.0 g=5.0
19.2 - actes p=2.0 g=4.0 | actualité p=1.0 g=7.0 | acte p=1.0 g=1.0 | actuellement␣
↪p=1.0 g=8.0
19.2 - actes p=2.0 g=4.0 | acte p=1.0 g=2.0 | actualité p=1.0 g=6.0 | actuellement␣
↪p=1.0 g=8.0
19.2 - actes p=2.0 g=4.0 | actuellement p=1.0 g=10.0 | actualité p=1.0 g=6.0 | acte␣
↪p=1.0 g=0.0
19.2 - actes p=2.0 g=4.0 | actualité p=1.0 g=7.0 | actuellement p=1.0 g=9.0 | acte␣
↪p=1.0 g=0.0
19.2 - actes p=2.0 g=4.0 | acte p=1.0 g=2.0 | actuellement p=1.0 g=9.0 | actualité␣
↪p=1.0 g=5.0
18.4 - actuellement p=1.0 g=11.0 | actes p=2.0 g=3.0 | actualité p=1.0 g=6.0 | acte␣
↪p=1.0 g=0.0
```

(suite sur la page suivante)

(suite de la page précédente)

```
18.4 - actuellement p=1.0 g=11.0 | actes p=2.0 g=3.0 | acte p=1.0 g=1.0 | actualité␣
↪p=1.0 g=5.0
18.4 - actualité p=1.0 g=8.0 | actes p=2.0 g=3.0 | actuellement p=1.0 g=9.0 | acte␣
↪p=1.0 g=0.0
18.4 - actualité p=1.0 g=8.0 | actes p=2.0 g=3.0 | acte p=1.0 g=1.0 | actuellement␣
↪p=1.0 g=8.0
18.4 - acte p=1.0 g=3.0 | actes p=2.0 g=3.0 | actuellement p=1.0 g=9.0 | actualité␣
↪p=1.0 g=5.0
18.4 - acte p=1.0 g=3.0 | actes p=2.0 g=3.0 | actualité p=1.0 g=6.0 | actuellement␣
↪p=1.0 g=8.0
17.6 - actuellement p=1.0 g=11.0 | actualité p=1.0 g=7.0 | actes p=2.0 g=2.0 | acte␣
↪p=1.0 g=0.0
17.6 - actuellement p=1.0 g=11.0 | acte p=1.0 g=2.0 | actes p=2.0 g=2.0 | actualité␣
↪p=1.0 g=5.0
17.6 - actualité p=1.0 g=8.0 | actuellement p=1.0 g=10.0 | actes p=2.0 g=2.0 | acte␣
↪p=1.0 g=0.0
17.6 - actualité p=1.0 g=8.0 | acte p=1.0 g=2.0 | actes p=2.0 g=2.0 | actuellement␣
↪p=1.0 g=8.0
17.6 - acte p=1.0 g=3.0 | actuellement p=1.0 g=10.0 | actes p=2.0 g=2.0 | actualité␣
↪p=1.0 g=5.0
17.6 - acte p=1.0 g=3.0 | actualité p=1.0 g=7.0 | actes p=2.0 g=2.0 | actuellement␣
↪p=1.0 g=8.0
16.8 - actuellement p=1.0 g=11.0 | actualité p=1.0 g=7.0 | acte p=1.0 g=1.0 | actes␣
↪p=2.0 g=1.0
16.8 - actuellement p=1.0 g=11.0 | acte p=1.0 g=2.0 | actualité p=1.0 g=6.0 | actes␣
↪p=2.0 g=1.0
16.8 - actualité p=1.0 g=8.0 | actuellement p=1.0 g=10.0 | acte p=1.0 g=1.0 | actes␣
↪p=2.0 g=1.0
16.8 - actualité p=1.0 g=8.0 | acte p=1.0 g=2.0 | actuellement p=1.0 g=9.0 | actes␣
↪p=2.0 g=1.0
16.8 - acte p=1.0 g=3.0 | actuellement p=1.0 g=10.0 | actualité p=1.0 g=6.0 | actes␣
↪p=2.0 g=1.0
16.8 - acte p=1.0 g=3.0 | actualité p=1.0 g=7.0 | actuellement p=1.0 g=9.0 | actes␣
↪p=2.0 g=1.0
```

Nouvelle métrique

Intuition

```
def gain_moyen_par_mot(queries, weights):
    total = sum(weights) * 1.0
    res = []
    for per in itertools.permutations(zip(queries, weights)):
        trie = CompletionTrieNode.build([(None, w) for w, p in per])
        wks = [(w, p, len(w)-trie.min_keystroke(w)[0]) for w, p in per]
        gain = sum( g*p/total for w, p, g in wks)
        res.append((gain, wks))
    res.sort(reverse=True)
    for i, r in enumerate(res):
        print("{0:3.4} - {1}".format(r[0], " | ".join("%s p=%1.1f g=%1.1f" % _ for _␣
↪in r[1])))
        if i > 10:
            print("...")
            break
```

```
queries = ['actuellement', 'actualité', 'actuel']
weights = [1, 1, 1]
gain_moyen_par_mot(queries, weights)
```

```
7.0 - actuellement p=1.0 g=11.0 | actuel p=1.0 g=4.0 | actualité p=1.0 g=6.0
7.0 - actuellement p=1.0 g=11.0 | actualité p=1.0 g=7.0 | actuel p=1.0 g=3.0
7.0 - actuel p=1.0 g=5.0 | actuellement p=1.0 g=10.0 | actualité p=1.0 g=6.0
7.0 - actuel p=1.0 g=5.0 | actualité p=1.0 g=7.0 | actuellement p=1.0 g=9.0
7.0 - actualité p=1.0 g=8.0 | actuellement p=1.0 g=10.0 | actuel p=1.0 g=3.0
7.0 - actualité p=1.0 g=8.0 | actuel p=1.0 g=4.0 | actuellement p=1.0 g=9.0
```

```
queries = ['actuellement', 'actualité', 'actuel']
weights = [1, 1, 0]
gain_moyen_par_mot(queries, weights)
```

```
9.0 - actuellement p=1.0 g=11.0 | actualité p=1.0 g=7.0 | actuel p=0.0 g=3.0
9.0 - actualité p=1.0 g=8.0 | actuellement p=1.0 g=10.0 | actuel p=0.0 g=3.0
8.5 - actuellement p=1.0 g=11.0 | actuel p=0.0 g=4.0 | actualité p=1.0 g=6.0
8.5 - actualité p=1.0 g=8.0 | actuel p=0.0 g=4.0 | actuellement p=1.0 g=9.0
8.0 - actuel p=0.0 g=5.0 | actuellement p=1.0 g=10.0 | actualité p=1.0 g=6.0
8.0 - actuel p=0.0 g=5.0 | actualité p=1.0 g=7.0 | actuellement p=1.0 g=9.0
```

```
queries = ['actuellement', 'actualité']
weights = [1, 1]
gain_moyen_par_mot(queries, weights)
```

```
9.0 - actuellement p=1.0 g=11.0 | actualité p=1.0 g=7.0
9.0 - actualité p=1.0 g=8.0 | actuellement p=1.0 g=10.0
```

Vérification

```
def gain_dynamique_moyen_par_mot(queries, weights, permutation=True):
    total = sum(weights) * 1.0
    res = []
    for per in itertools.permutations(zip(queries, weights)):
        trie = CompletionTrieNode.build([(None, w) for w, p in per])
        trie.precompute_stat()
        trie.update_stat_dynamic()
        wks = [(w, p, len(w)-trie.min_dynamic_keystroke(w)[0]) for w, p in per]
        gain = sum( g*p/total for w, p, g in wks)
        res.append((gain, wks))
        if not permutation:
            break
    res.sort(reverse=True)
    for i, r in enumerate(res):
        print("{0:3.4} - {1}".format(r[0], " | ".join("%s p=%1.1f g=%1.1f" % _ for _
↪in r[1])))
        if i > 10:
            print("...")
            break
```

Pas de changement :

```
queries = ['actuellement', 'actualité', 'actuel']
weights = [1, 1, 0]
gain_dynamique_moyen_par_mot(queries, weights)
```

```
9.0 - actuellement p=1.0 g=11.0 | actualité p=1.0 g=7.0 | actuel p=0.0 g=3.0
9.0 - actualité p=1.0 g=8.0 | actuellement p=1.0 g=10.0 | actuel p=0.0 g=3.0
8.5 - actuellement p=1.0 g=11.0 | actuel p=0.0 g=4.0 | actualité p=1.0 g=6.0
8.5 - actuel p=0.0 g=5.0 | actualité p=1.0 g=7.0 | actuellement p=1.0 g=10.0
8.5 - actualité p=1.0 g=8.0 | actuel p=0.0 g=4.0 | actuellement p=1.0 g=9.0
8.0 - actuel p=0.0 g=5.0 | actuellement p=1.0 g=10.0 | actualité p=1.0 g=6.0
```

Changements :

```
queries = ['actuellement', 'actualité', 'actuel']
weights = [1, 1, 1]
gain_dynamique_moyen_par_mot(queries, weights)
```

```
7.333 - actuel p=1.0 g=5.0 | actualité p=1.0 g=7.0 | actuellement p=1.0 g=10.0
7.0 - actuellement p=1.0 g=11.0 | actuel p=1.0 g=4.0 | actualité p=1.0 g=6.0
7.0 - actuellement p=1.0 g=11.0 | actualité p=1.0 g=7.0 | actuel p=1.0 g=3.0
7.0 - actuel p=1.0 g=5.0 | actuellement p=1.0 g=10.0 | actualité p=1.0 g=6.0
7.0 - actualité p=1.0 g=8.0 | actuellement p=1.0 g=10.0 | actuel p=1.0 g=3.0
7.0 - actualité p=1.0 g=8.0 | actuel p=1.0 g=4.0 | actuellement p=1.0 g=9.0
```

```
gain_moyen_par_mot(queries, weights)
```

```
7.0 - actuellement p=1.0 g=11.0 | actuel p=1.0 g=4.0 | actualité p=1.0 g=6.0
7.0 - actuellement p=1.0 g=11.0 | actualité p=1.0 g=7.0 | actuel p=1.0 g=3.0
7.0 - actuel p=1.0 g=5.0 | actuellement p=1.0 g=10.0 | actualité p=1.0 g=6.0
7.0 - actuel p=1.0 g=5.0 | actualité p=1.0 g=7.0 | actuellement p=1.0 g=9.0
7.0 - actualité p=1.0 g=8.0 | actuellement p=1.0 g=10.0 | actuel p=1.0 g=3.0
7.0 - actualité p=1.0 g=8.0 | actuel p=1.0 g=4.0 | actuellement p=1.0 g=9.0
```

Ajouter une complétion

```
queries = ['macérer', 'maline', 'machinerie', 'machinerie infernale', 'machinerie␣
↪infernalissime',
           'machine artistique', 'machine automatique',
           'machine chaplin', 'machine intelligente', 'machine learning']
weights = [1] * len(queries)
gain_dynamique_moyen_par_mot(queries, weights, permutation=False)
```

```
10.1 - macérer p=1.0 g=6.0 | maline p=1.0 g=4.0 | machinerie p=1.0 g=7.0 | machinerie␣
↪infernale p=1.0 g=16.0 | machinerie infernalissime p=1.0 g=20.0 | machine␣
↪artistique p=1.0 g=12.0 | machine automatique p=1.0 g=12.0 | machine chaplin p=1.0␣
↪g=7.0 | machine intelligente p=1.0 g=11.0 | machine learning p=1.0 g=6.0
```

```
queries = ['machine'] + queries
weights = [1] * len(queries)
weights[queries.index('machine')] = 0.0
gain_dynamique_moyen_par_mot(queries, weights, permutation=False)
```

```
12.3 - machine p=0.0 g=6.0 | macérer p=1.0 g=5.0 | maline p=1.0 g=3.0 | machinerie␣
↪p=1.0 g=8.0 | machinerie infernale p=1.0 g=17.0 | machinerie infernalissime p=1.0␣
↪g=21.0 | machine artistique p=1.0 g=15.0 | machine automatique p=1.0 g=15.0 |␣
↪machine chaplin p=1.0 g=11.0 | machine intelligente p=1.0 g=16.0 | machine learning␣
↪p=1.0 g=12.0
```

Wikipedia

— PageCount [599]
— dump [600]

12.5.4 Complétion Simple

Evaluation d'une métrique pour un système de complétion sur quelques cas simples.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

— *Métrique M'* (page 350)

Métrique M'

```
from mlstatpy.nlp import CompletionSystem
mots = ["po", "po rouge", "po vert", "po orange", "port", "port blanc", "port bleu",
↪"port rouge"]
ens = CompletionSystem(mots)
ens.compute_metrics()
for el in ens:
    print("n={1} : M'={0} | {2}".format(el.mks1, el.weight, el.value))
```

```
n=0 : M'=1 | po
n=1 : M'=2 | po rouge
n=2 : M'=3 | po vert
n=3 : M'=4 | po orange
n=4 : M'=3 | port
n=5 : M'=4 | port blanc
n=6 : M'=5 | port bleu
n=7 : M'=6 | port rouge
```

```
mots_rev = mots.copy()
mots_rev[4], mots_rev[-1] = mots_rev[-1], mots_rev[4]
ens = CompletionSystem(mots_rev)
ens.compute_metrics()
for el in ens:
    print("n={1} : M'={0} | {2}".format(el.mks1, el.weight, el.value))
```

599. https://dumps.wikimedia.org/other/pagecounts-raw/
600. https://dumps.wikimedia.org/backup-index.html

```
n=0 : M'=1 | po
n=1 : M'=2 | po rouge
n=2 : M'=3 | po vert
n=3 : M'=4 | po orange
n=4 : M'=3 | port rouge
n=5 : M'=4 | port blanc
n=6 : M'=5 | port bleu
n=7 : M'=3 | port
```

```
mots_court = [m[4:] for m in mots if m.startswith("port") and len(m) > 4]
ens = CompletionSystem(mots_court)
ens.compute_metrics()
for el in ens:
    print("n={1} : M'={0} | {2}".format(el.mks1, el.weight, el.value))
```

```
n=0 : M'=1 |  blanc
n=1 : M'=2 |  bleu
n=2 : M'=3 |  rouge
```

```
mots_court = [m for m in mots if m != "port"]
ens = CompletionSystem(mots_court)
ens.compute_metrics()
for el in ens:
    print("n={1} : M'={0} | {2}".format(el.mks1, el.weight, el.value))
```

```
n=0 : M'=1 | po
n=1 : M'=2 | po rouge
n=2 : M'=3 | po vert
n=3 : M'=4 | po orange
n=4 : M'=3 | port blanc
n=5 : M'=4 | port bleu
n=6 : M'=5 | port rouge
```

```
couleur = ["blanc", "vert", "orange", "rouge", "noir", "noire", "blanche"]
key = "portes"
mots = ["port", "port rouge", "port vert", "port orange", "pore", "pour"]
mots.append(key)
mots += [key + " " + c for c in couleur]
ens = CompletionSystem(mots)
ens.compute_metrics()
for el in ens:
    print("n={1} : M'={0} | {2}".format(el.mks1, el.weight, el.value))
```

```
n=0 : M'=1 | port
n=1 : M'=2 | port rouge
n=2 : M'=3 | port vert
n=3 : M'=4 | port orange
n=4 : M'=4 | pore
n=5 : M'=4 | pour
n=6 : M'=3 | portes
n=7 : M'=4 | portes blanc
n=8 : M'=5 | portes vert
n=9 : M'=6 | portes orange
n=10 : M'=6 | portes rouge
n=11 : M'=6 | portes noir
```

(suite sur la page suivante)

(suite de la page précédente)

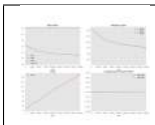
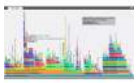
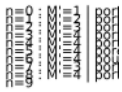
```
n=12 : M'=7 | portes noire
n=13 : M'=5 | portes blanche
```

```
mots2 = [m for m in mots if m != "portes"]
ens = CompletionSystem(mots2)
ens.compute_metrics()
for el in ens:
    print("n={1} : M'={0} | {2}".format(el.mks1, el.weight, el.value))
```

```
n=0 : M'=1 | port
n=1 : M'=2 | port rouge
n=2 : M'=3 | port vert
n=3 : M'=4 | port orange
n=4 : M'=4 | pore
n=5 : M'=4 | pour
n=6 : M'=3 | portes blanc
n=7 : M'=4 | portes vert
n=8 : M'=5 | portes orange
n=9 : M'=6 | portes rouge
n=10 : M'=6 | portes noir
n=11 : M'=7 | portes noire
n=12 : M'=4 | portes blanche
```

```
mots3 = mots2.copy()
mots3.insert(1, "portes")
ens = CompletionSystem(mots3)
ens.compute_metrics()
for el in ens:
    print("n={1} : M'={0} | {2}".format(el.mks1, el.weight, el.value))
```

```
n=0 : M'=1 | port
n=1 : M'=2 | portes
n=2 : M'=3 | port rouge
n=3 : M'=4 | port vert
n=4 : M'=4 | port orange
n=5 : M'=4 | pore
n=6 : M'=4 | pour
n=7 : M'=3 | portes blanc
n=8 : M'=4 | portes vert
n=9 : M'=5 | portes orange
n=10 : M'=5 | portes rouge
n=11 : M'=5 | portes noir
n=12 : M'=6 | portes noire
n=13 : M'=4 | portes blanche
```

	*Completion Trie and metrics* (page 328)	Evaluation of a completion system on wikpedia pages.
	*Completion profiling* (page 333)	Profiling avec cProfile, memory_profiler, line_profiler, pyinstrument, snakeviz.
	Complétion (page 344)	Comparaion de plusieurs algorithmes pour implémenter un système de complétion.
	*Complétion Simple* (page 350)	Evaluation d'une métrique pour un système de complétion sur quelques cas simples.

12.5.5 Notebooks Coverage

Report on last executions.

95% 2018-10-27

index	coverage	exe time	last execution	name	title	success	time	nb cells	nb runs	nb valid
0	100%	0.131	2018-10-27	*dsgarden/classification_multiple.ipynb* (page 299)	*Classification multiple* (page 299)	True	3.371	2	2	2
1	100%	117.432	2018-10-27	*dsgarden/correlation_non_lineaire.ipynb* (page 192)	*Corrélations non linéaires* (page 192)	True	119.714	41	41	41
2	100%	20.465	2018-10-27	*dsgarden/discret_gradient.ipynb* (page 234)	*Le gradient et le discret* (page 234)	True	22.603	22	22	22
3	100%	10.081	2018-10-27	*dsgarden/file_dattente_ex.ipynb* (page 301)	*File d'attente, un exemple simple* (page 301)	True	13.138	8	8	8
4	100%	8.255	2018-10-27	*dsgarden/quantile_regression_example.ipynb* (page 246)	*Régression quantile illustration* (page 246)	True	11.146	10	10	10
5	100%	29.741	2018-10-27	*dsgarden/regression_lineaire.ipynb* (page 221)	*Régression linéaire et résultats numériques* (page 221)	True	32.185	41	41	41
6	100%	377.498	2018-10-27	*dsgarden/split_train_test.ipynb* (page 179)	*Répartir en base d'apprentissage et de test* (page 179)	True	379.688	29	29	29
7	100%	62.067	2018-10-27	*image/segment_detection.ipynb* (page 172)	*Détection de segments dans une image* (page 172)	True	64.169	17	17	17
8	100%	23.260	2018-10-27	*metric/pvalues_examples.ipynb* (page 145)	*p-values* (page 145)	True	25.468	10	10	10
9	100%	14.582	2018-10-27	*metric/roc_example.ipynb* (page 304)	*ROC* (page 304)	True	17.163	21	21	21
10	100%	20.822	2018-10-27	*ml/benchmark.ipynb* (page 311)	*Benchmark* (page 311)	True	24.173	9	9	9
11	100%	102.562	2018-10-27	*ml/logreg_voronoi.ipynb* (page 88)	*Voronoï et régression logistique* (page 88)	True	105.366	64	64	64
12	100%	7.357	2018-10-27	*ml/mf_acp.ipynb* (page 314)	*Factorisation et matrice et ACP* (page 314)	True	10.143	14	14	14
13	100%	4.203	2018-10-27	*ml/reseau_neurones.ipynb* (page 318)	*Réseaux de neurones* (page 318)	True	7.134	5	5	5
14	100%	12.851	2018-10-27	*ml/valeurs_manquantes_mfpyx.ipynb* (page 320)	*Valeurs manquantes et factorisation de matrices* (page 320)	True	15.157	35	35	35
15	100%	101.508	2018-10-27	*nlp/completion_profiling.ipynb* (page 333)	*Completion profiling* (page 333)	True	104.542	27	27	27
16	100%	1.129	2018-10-27	*nlp/completion_simple.ipynb* (page 350)	*Complétion Simple* (page 350)	True	4.123	9	9	9
17	100%	2.955	2018-10-27	*nlp/completion_trie.ipynb* (page 344)	*Complétion* (page 344)	True	5.139	15	15	15
18	0%	nan		*nlp/completion_trie_long.ipynb* (page 328)	*Completion Trie and metrics* (page 328)		nan	20	0	


NB: 94%

Bibliographie

[Arthur2007] k-means++ : the advantages of careful seeding (2007), *Arthur, D.; Vassilvitskii, S.*, Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. pp. 1027–1035. PDF [6].

[Balakrishnan1996] Comparative performance of the FSCL neural net and K-means algorithm for market segmentation (1996), P. V. Sundar Balakrishnan, Martha Cooper, Varghese S. Jacob, Phillip A. Lewis, *European Journal of Operation Research*, volume 93, pages 346-357

[Bahmani2012] Scalable K-Means++ (2012), *Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, Sergei Vassilvitskii*, Proceedings of the VLDB Endowment (PVLDB), Vol. 5, No. 7, pp. 622-633 (2012) PDF [7], arXiv [8]

[Cheung2003] k^*-Means : A new generalized k-means clustering algorithm (2003), Yiu-Ming Cheung, *Pattern Recognition Letters*, volume 24, 2883-2893

[Davies1979] A cluster Separation Measure (1979), D. L. Davies, D. W. Bouldin, *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, volume 1(2)

[Goodman1954] Measures of associations for cross-validations (1954), L. Goodman, W. Kruskal, *J. Am. Stat. Assoc.*, volume 49, pages 732-764

[Herbin2001] Estimation of the number of clusters and influence zones (2001), M. Herbin, N. Bonnet, P. Vautrot, *Pattern Recognition Letters*, volume 22, pages 1557-1568

[Kothari1999] On finding the number of clusters (1999), Ravi Kothari, Dax Pitts, *Pattern Recognition Letters*, volume 20, pages 405-416

[Liu2003] Strip line detection and thinning by RPCL-based local PCA (2003), Zhi-Yong Liu, Kai-Chun Chiu, Lei Xu, *Pattern Recognition Letters* volume 24, pages 2335-2344

[Silverman1986] Density Estimation for Statistics and Data Analysis (1986), B. W. Silverman, *Monographs on Statistics and Applied Probability, Chapman and Hall, London*, volume 26

[Xu1993] Rival penalized competitive learning for clustering analysis, rbf net and curve detection (1993), L. Xu, A. Krzyzak, E. Oja, *IEEE Trans. Neural Networks*, volume (4), pages 636-649

[Biernacki2001] {Assessing a Mixture Model for Clustering with the Integrated Completed Likelihood (2001), C. Biernacki, G. Deleux, G. Govaert, *IEEE Transactions on Image Analysis and Machine Intelligence*, volume {22(7), pages 719-725

[Celeux1985] The SEM algorithm : a probabilistic teacher algorithm derived from the EM algorithm for the mixture problem (1985), G. Celeux, J. Diebolt, *Computational Statistics Quarterly*, Volume 2(1), pages 73-82

6. http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf
7. http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf
8. https://arxiv.org/abs/1203.6402

[Celeux1985b] On stochastic version of the EM algorithm (1985), Gilles Celeux, Didier Chauveau, Jean Diebolt, Rapport de recherche de l'INRIA*, n 2514

[Dempster1977] Maximum-Likelihood from incomplete data via the EM algorithm (1977), A. P. Dempster, N. M. Laird, D. B. Rubin, *Journal of Royal Statistical Society B*, volume 39, pages 1-38

[Figueiredo2002] Unsupervised learning of finite mixture models (2002), M. A. T. Figueiredo, A. K. Jain, IEEE Transactions on Pattern Analysis and Machine Intelligence, volume 24(3), pages 381-396

[ZhangB2004] Competitive EM algorithm for finite mixture models (2004), Baibo Zhang, Changshui Zhang, Xing Yi, *Pattern Recognition*, volume 37, pages 131-144

[Kohonen1982] Self-organized formation of topologically correct feature maps (1982), T. Kohonen, *Biol. Cybern.*, volume (43), pages 59-69

[Kohonen1997] Self-Organizing Map (1997) T. Kohonen, *Springer*

[Lo1991] On the rate of convergence in topology preserving neural networks (1991), Z. Lo, B. Bavarian, *Biological Cybernetics*, volume 63, pages 55-63

[Rougier] Dynamic Self-Organising Map [15], Nicolas P. Rougier and Yann Boniface

[Wu2004] Clustering of the self-organizing map using a clustering validity index based on inter-cluster and intra-cluster density (2004), Sitao Wu, Tommy W. S. Chow, *Pattern Recognition*, volume (37), pages 175-188

[Bottou1991] **Une approche théorique de l'apprentissage connexionniste,** Application à la reconnaissance de la parole, Léon Bottou,

Thèse de l'Université de Paris Sud, Centre d'Orsay.

[Broyden1967] Quasi-Newton methods and their application to function minimization (1967), C. G. Broyden, *Math. Comput pages 21-368*

[Bishop1995] Neural networks for pattern recognition (1995), C. M. Bishop, *Oxford University Press*

[Cottrel1995] Neural modeling for time series : a statistical stepwise methode for weight elimination (1995), M. Cottrel, B. Girard, M. Mangeas, C. Muller, *IEEE Transaction On Neural Networks*

[Cybenko1989] Approximation by superpositions of a sigmoidal function (1989), G. Cybenko, *Mathematics of Controls, Signals, and Systems*, p 303-314

[Davidon1959] Variable metric method for minimization (1959), C. W. Davidon, *A.E.C. Research and Development Report, ANL-5990*

[Driancourt1996] Optimisation par descente de gradient stochastique de systèmes modulaires combinant réseaux de neurones et programmation dynamique, Application à la reconnaissance de la parole (1996), X. Driancourt, *Thèse de l'Université de Paris Sud, Centre d'Orsay*.

[Fletcher1963] A rapidly convergent descent method for minimization (1963), R. Fletcher, M. J. D. Powell, *Computer Journal 6, pages 163-168*

[Fletcher1993] An overview of Unconstrained Optimization (1993), R. Fletcher, *Numerical Analysis Report NA/149*

[Kullback1951] On information and sufficiency (1951), S. Kullback, R. A. Leibler, *Ann. Math. Stat. 22, pages 79-86*

[LeCun1985] Une procédure d'apprentissage pour réseaux à seuil asymétrique (1985), Yann Le Cun, *Cognita*, p 599-604

[Moré1977] The Levenberg-Marquardt algorithm : Implementation and theory (1977), J. J. Moré, *Proceedings of the 1977 Dundee Conference on Numerical Analysis, G. A. Watson, ed., Lecture Notes in Mathematics, vol. 630, Springer-Verlag, Berlin, pages 105-116*

[Rumelhart1986] Learning internal representations by error propagation (1986), D. E. Rumelhart, G. E. Hinton, R. J. Williams in *Parallel distributed processing : explorations in the microstructures of cohniyionn MIT Press, Cambridge*

[Saporta1990] Probabilités, analyse des données et statistique (1990), Gilbert Saporta, *Editions Technip*

[Song1997] Self-organizing algorithm of robust PCA based on single layer NN (1997) Song Wang, Shaowei Xia, *Proceedings of the 4th International Conference Document Analysis and Recognition*

15. http://www.labri.fr/perso/nrougier/coding/article/article.html

[Beckmann1990] The R*-tree : an efficient and robust access method for points and rectangles, N. Beckmann, H. P. Kriegel, P. Schneider, B. Seeger, Proceedings of SIGMOD conference, Atlantic City, pages 322-331

[Berchtold1996] The X-Tree : An index structure for high dimension data, S. Berchtold, D. A. Keim, H. P. Kriegel, Proceedings of the 22nd Internation Conference on Very Large Databases, Bombay, India

[Farago1993] Fast Nearest-Neighbor Search in Dissimilarity Spaces, A. Farago, T. Linder, G. Lugosi, IEEE Transactions on Pattern Analysis and Machine Intelligence, volume 15(9), pages 957-962

[Guttman1984] R-Trees : A Dynamic Index Structure for Spatial Searching, A. Guttman, Proceedings ACM SIGMOD, pages 47-57

[Moreno2003] A modification of the LAESA algorithm for approximated k-NN classification, Francisco Moreno-Seco, Luisa Mico, Jose Oncina, Pattern Recognition Letters, volumne 24, pages 47-53

[Rico-Juan2003] Comparison of AESA and LAESA search algorithms using string and tree-edit-distances, J. R. Rico-Juan, L. Mico, Pattern Recognition Letters, volume 24, pages 1417-1426

[Sellis1987] The R+tree - a Dynamic Index for Multi-Dimensional Objects, T. Sellis, N. Roussopoulos, C. Faloutos, Proceedings of the 13th VLDB conference, pages 507-518

[Acara2011] Scalable tensorfactorizations for incomplete data, *Evrim Acara Daniel, M.Dunlavyc, Tamara G.Koldab. Morten Mørupd*, Chemometrics and Intelligent Laboratory Systems, Volume 106, Issue 1, 15 March 2011, Pages 41-56, or ArXiv 1005.2197 [83]

[Boutsidis2008] SVD-based initialization : A head start for nonnegative matrix factorization. *Christos Boutsidis and Efstratios Gallopoulos* Pattern Recognition, 41(4) : 1350-1362, 2008.

[Gilles2014] The Why and How of Nonnegative Matrix Factorization, *Nicolas Gillis*, ArXiv 1401.5226 [84]

[Gupta2010] Additive Non-negative Matrix Factorization for Missing Data, *Mithun Das Gupta*, ArXiv 1007.0380 [85]

[Bampoulidis2017] Does Online Evaluation Correspond to Offline Evaluation in Query Auto Completion ? (2017) Alexandros Bampoulidis, João PalottiMihai LupuJon BrasseyAllan Hanbury *ECIR 2017 : Advances in Information Retrieval*

[Sevenster2013] Algorithmic and user study of an autocompletion algorithm on a large medical vocabulary (2013), Merlijn Sevenster, Rob van Ommering, Yuechen Qian *Journal of Biomedical Informatics* 45, pages 107-119

[Agarwal2005] Generalization Bounds for the Area Under the ROC Curve (2005), Shivani Agarwal, Thore Graepel, Ralf Herbich, Sariel Har-Peled, Dan Roth *Journal of Machine Learning Research, volume 6, pages 393-425*

[Saporta1990] Probabilités, analyse des données et statistique (1990), Gilbert Saporta, *Editions Technip*

[Damerau1964] A technique for computer detection and correction of spelling errors (1964), *F. J. Damerau*, Commun. ACM, volume 7(3), pages 171-176

[Kripasundar1996] Generating edit distance to incorporate domain information (1996), *V. Kripasunder, G. Seni, R. K. Srihari*, CEDAR/SUNY

[Levenstein1966] Binary codes capables of correctiong deletions, insertions, and reversals (1966), *V. I. Levenstein*, Soviet Physics Doklady, volume 10(8), pages 707-710

[Seni1996] Generalizing edit distance to incorporate domain information : handwritten text recognition as a case study (1996), *Giovanni Seni, V. Kripasundar, Rohini K. Srihari*, Pattern Recognition volume 29, pages 405-414

[Waard1995] An optimised minimal edit distance for hand-written word recognition (1995), *W. P. de Waard*, Pattern Recognition Letters volume 1995, pages 1091-1096

[Wagner1974] The string-to-string correction problem (1974), *R. A. Wagner, M. Fisher*, Journal of the ACM, volume 21, pages 168-178

[Blondel2004] A measure of similarity between graph vertices *Vincent Blondel, Anahi Gajardo, Maureen Heymans, Pierre Senellart, Paul Van Dooren*, arxiv/0407061 [141]

83. https://arxiv.org/pdf/1005.2197.pdf
84. https://arxiv.org/abs/1401.5226
85. https://arxiv.org/abs/1007.0380
141. https://arxiv.org/abs/cs/0407061

[Faure2000] Précis de recherche opérationnelle, 5ième édition, *Robert Faure, Bernard Lemaire, Christophe Picouleau*, Dunod

[Gouriéroux1983] Analyse des séries temporelles, Christian Gouriéroux, Alain Monfort, Editions Economica

[Saporta2006] Probabilités, analyse des données et statistique, Gilbert Saporta, Editions Technip

[Chen2014] Fast Iteratively Reweighted Least Squares Algorithms for Analysis-Based Sparsity Reconstruction [177] Chen Chen, Junzhou Huang, Lei He, Hongsheng Li

177. https://arxiv.org/abs/1411.5057