Table des matières

Table des figures

Liste des tableaux

Abréviations

OAA	**O**ne **A**gainst **A**ll
OAO	**O**ne **A**gainst **O**ne
ECOC	**E**rror **C**orrecting **O**utput **C**ode
SVM	**S**upport **V**ector **M**achine

Symboles

x	exemple
D	dimension de l'espace des exemples : $x \in \mathbb{R}^D$
y	index de la classe
ℓ_y	classe
$\mathcal{D} = \{(x_i, y_i)\}_{i \in [\![1,n]\!]}$	ensemble d'apprentissage
$n = \lvert\mathcal{D}\rvert$	nombre d'exemples dans $\mathcal{D}$
$\mathcal{X}$	matrice des exemples d'apprentissage : $\mathcal{X} \in \mathbb{R}^{n \times D}$
$\mathcal{D}_t$	ensemble d'apprentissage de test
$n_t = \lvert\mathcal{D}_t\rvert$	nombre d'exemple dans $\mathcal{D}_t$
$\mathcal{L}$	ensemble des classes du problème de classification
K	nombre de classes : $K = \lvert\mathcal{L}\rvert$
$\mathcal{C}$	sous-ensemble de classe : $\mathcal{C} \subset \mathcal{L}$
$\mathcal{X}_\mathcal{C}$	sous-ensemble d'exemples de $\mathcal{D}$ de classe dans $\mathcal{C}$
f	classifieur binaire simple
$f_{\{\mathcal{C}^+,\mathcal{C}^-\}}$	classifieur ayant appris à séparer les exemples des classes de $\mathcal{C}^+$ des exemples de classes de $\mathcal{C}^-$
R	fonction de risque
	ECOC
T	nombre de classifieurs
$\mathbf{C}$	matrice de codage
	Hiérarchie
$\mathcal{E}_i$	ensemble des noeuds enfants du noeud d'index i

Schéma de classification

$\mathcal{S}$	schéma de classification
C_{x}	complexité d'un schéma
r_{x}	ratio de complexité d'un schéma
$f_{\mathcal{S}}$	processus de création des classifieurs d'un schéma $\mathcal{S}$
$b_{\mathcal{S}}$	processus de sélection de classifieur d'un schéma $\mathcal{S}$
$u_{\mathcal{S}}$	processus de mise à jour d'un schéma $\mathcal{S}$
$p_{\mathcal{S}}$	processus de prédiction finale d'un schéma $\mathcal{S}$

Chapitre 1

Introduction

1.1 Classification extrême et Big Data

1.1.1 Apprentissage automatique

Toute tâche d'apprentissage statistique est basée sur l'exploitation de données. Nous allons ainsi commencer par définir cette donnée qui constitue la brique de base sur laquelle s'appuient tous les algorithmes d'apprentissage. Une donnée représente une information. Cette information peut prendre diverses formes comme la valeur des pixels d'une image, les lettres et les mots formant un document textuel ou les caractéristiques telles que l'âge ou le sexe d'un être humain. Ces données sont produites constamment et en très grand nombre tous les jours. Un individu surfant sur internet laisse derrière lui un sillon d'informations par l'intermédiaire de cookies que stockent les sites visités dans le but de vendre des espaces publicitaires adaptés. Un capteur de pollution sur un toit parisien va enregistrer une information de pollution de l'air (dioxyde de carbone, ozone) à un certain moment de la journée et pour un endroit spécifique dans Paris. Ces données brutes ne sont pas utilisables directement, un traitement est requis pour extraire une information spécifique qui sera ensuite utilisable selon les besoins du problème.

Le fait de traiter ces données a un coût. Il est nécessaire d'utiliser du temps de processeur pour analyser, trier et extraire une information *utile* à partir d'une masse de données brutes. Une donnée brute venant d'un capteur d'appareil photo numérique ne sera qu'une succession de nombres venant de la mesure de la quantité de lumière qui arrive sur

chacun des pixels constituant le capteur. En l'état, cette succession de nombres n'est pas utilisable. Le simple fait d'afficher cette image sur un écran constitue un premier traitement simple de ces données où une correspondance entre la donnée brute du capteur et la valeur de l'intensité du pixel doit être déterminée. Un traitement plus complexe consisterait à *augmenter* l'information de l'image en annotant les différents éléments qui sont visibles sur la photo (visages, voitures, arbres).

Ce traitement permet d'ajouter de la valeur et de l'interêt à cette matrice de valeurs de pixels qui constitue une donnée. Grâce à ce processus d'annotation, un utilisateur pourra par exemple trier ses photos en fonction des personnes présentes sur chacune d'elles.

La famille des algorithmes d'apprentissage statistique supervisé regroupe un ensemble de méthodes permettant d'automatiser une tâche particulière de traitement de données en apprenant à généraliser à partir d'un ensemble d'exemples dits d'apprentissages pré-annotés (le plus souvent à la main). Il faut ensuite réussir à modéliser au mieux le processus d'inférence permettant d'*augmenter* l'information de la donnée. Une fois les paramètres du modèle appris grâce à l'ensemble d'apprentissage, le modèle est supposé être capable de reproduire l'inférence d'information pour de nouvelles données jamais vues auparavant.

1.1.2 Big Data

Il apparaît que la vitesse d'enregistrement de ces données connaît de nos jours un accroissement exponentiel. Il est reconnu partout aujourd'hui l'importance ainsi que la valeur de la donnée et il est devenu coutumier d'en enregistrer le plus possible même sans savoir exactement quoi en faire. De plus, avec la multiplication des smartphones et des objets connectés, chaque individu de nos sociétés informatisées crée un volume de données toujours plus important. Ces différents facteurs font que le différentiel grandit entre notre capacité à enregistrer de la donnée et notre capacité à la traiter. Cet écart va se creuser de plus en plus au point où il ne sera pas possible de juste doubler ou tripler la taille des clusters de calculs. L'idée d'adapter nos algorithmes de calcul pour faire face à ces volumes de données est un enjeu crucial pour les années à venir. Ainsi, notre capacité à traiter des données plus rapidement que leur vitesse de création est devenu un objectif fondamental actuel.

1.2 Temps d'inférence et complexité computationnelle

1.2.1 La tâche de classification

Parmi la multitude de tâches liées à l'apprentissage statistique, nous allons nous intéresser ici à la tâche de classification multi-classes, et plus particulièrement à la tâche de classification multi-classes avec un (très) grand nombre de classes. La tâche de classification consiste à catégoriser des éléments parmi plusieurs classes ou familles d'objets. La tâche qui consiste à trouver la catégorie d'un texte est un exemple de problème de classification. Il s'agit alors d'automatiser le processus permettant d'étiqueter le document textuel comme étant un texte parlant de voitures, du Siècle des Lumières où de la géographie de Paris. Une fois que le processus de traitement est appris, il est possible d'utiliser le modèle pour catégoriser automatiquement une nouvelle donnée.

1.2.2 Problèmes avec la Classification Extrême

La particularité de la tâche ici est de s'intéresser au cas où le nombre de catégories est beaucoup plus important que la normale. On parle ici de classification avec 1000, voire 10000 classes alors que les problèmes multi-classes classiques ne dépassent pas habituellement la centaine de classes. Aujourd'hui, il existe même des problèmes dépassant le million de catégories. Il n'est pas difficile d'imaginer des tâches pouvant atteindre le milliard de catégories dans le cas de reconnaissance de visages humains par exemple.

Cette particularité engendre de nouvelles problématiques spécifiques au grand nombre de classes. Tout d'abord la tâche est plus complexe étant donné le nombre de choix plus importants. En effet, un simple classifieur aléatoire pourra espérer obtenir un taux de bonne classification de 50% pour un problème à deux classes, mais seulement de 0.01% pour un problème à 10000 classes. Ensuite, le temps de classification est plus important pour un problème avec un plus grand nombre de classes. Avec les approches classiques, ce temps de classification évolue généralement de façon linéaire avec le nombre de classes. Ceci s'explique par le fait que le classifieur classique multi-classes à besoin d'établir un score par classe. S'il y a beaucoup de classes, cela nécessite de faire intervenir la fonction de score un grand nombre de fois ce qui peut être problématique dans les problèmes de classification extrême.

Ainsi, ce travail est motivé par le besoin de produire des algorithmes de classification capables de classifier avec une complexité sous-linéaire en fonction du nombre de classes. Nous parlons ici du temps de classification d'un exemple et non du temps d'apprentissage du modèle. La problématique du temps d'apprentissage n'est pas à mettre au second plan pour autant et il apparaît évident qu'un modèle se doit de pouvoir être appris et entraîné en un temps *raisonnable*. Ce *raisonnable* est directement dépendant du contexte et des contraintes liées au problème à résoudre. Le modèle a-t-il besoin d'être mis à jour régulièrement (et donc ré-appris) ? Le travail présenté ici s'est concentré sur des algorithmes avec un temps de classification sous-linéaire (en fonction du nombre de classes) mais avec des temps d'apprentissage similaires aux méthodes classiques (c'est-à-dire au pire linéaire avec le nombre de classes du problème).

1.2.3 Accélérer le processus de classification

Nous avons ainsi notre but clairement énoncé : réduire la complexité du processus de classification. Habituellement, le schéma de modèle multi-classes le plus connu est le Un-Contre-Tous (ou plus communément désigné par l'acronyme OAA pour *One-Against-All*). Il s'agit simplement d'apprendre une fonction de score pour chaque classe. Lors de la classification d'un nouvel exemple, il suffit de calculer le score de chaque classe pour cet exemple. L'algorithme prédit ensuite la classe ayant obtenu le meilleur score. Le choix de la fonction de score est libre.

Ainsi, le schéma de classification OAA nécessite d'utiliser K fois la fonction de score, où K représente le nombre de classes du problème. Comme expliqué précédemment, notre but est de faire mieux que cette complexité de classification en $O(K)$ et de viser des complexités en $O(\log K)$. Il est nécessaire alors de réduire le nombre de fonctions de score à utiliser.

Pour ce faire, il existe principalement deux types de modèle :

- Le modèle *hiérarchique*
- Le modèle *ensembliste*

Le modèle hiérarchique : ce type de modèle vise à organiser hiérarchiquement les fonctions de décision dans une structure arbre. Classiquement, les feuilles représentent les classes. Chaque nœud possède un classifieur. L'exemple à classifier commence au nœud

racine, puis descend dans l'arbre en suivant les résultats des classifieurs de chacun des nœuds rencontrés jusqu'à atteindre un nœud feuille. La classe prédite pour cet exemple correspond à la classe associée à ce nœud feuille. Ainsi, le nombre de classifieurs utilisé sera égal à la profondeur de l'arbre (en supposant l'arbre équilibré). De par la caractéristique naturelle des arbres, la profondeur d'un arbre équilibré est logarithmique en fonction du nombre de feuilles, et donc du nombre de classes. Une classification avec un tel modèle possède ainsi une complexité en $O(\log K)$.

Le modèle ensembliste. L'idée des méthodes ensemblistes est d'agréger les réponses d'une multitude de classifieurs et de prédire la classe avec une majorité de votes positifs. Une méthode de type OAA utilise des classifieurs spécialisés dans la reconnaissance d'une classe unique. À la différence d'une méthode OAA, une méthode de la famille des codes correcteurs d'erreurs (ou ECOC pour *Error Correcting Output Codes*) utilise un ensemble de classifieurs dont chacun d'entre eux est entraîné à reconnaître un nombre limité de classes largement inférieur à K, le nombre de classes à reconnaître. L'objectif est d'obtenir un nombre final de classifieurs (pour obtenir une prédiction), moindre qu'avec une méthode de type OAA. La difficulté est alors de bien contrôler le nombre de ces classifieurs car un trop petit nombre ne permettra pas d'obtenir une classification suffisamment précise.

1.3 Contributions

Nous allons présenter ici deux travaux. Le premier travail concerne une nouvelle méthode de création de hiérarchie de classes. Contrairement à ce qui se fait habituellement, la méthode proposée vise à découvrir la hiérarchie de classes par dichotomie : le but est d'inférer les étages intermédiaires de la hiérarchie de façon dynamique à chaque itération de l'algorithme. Cette approche est accompagnée d'une nouvelle méthode de partitionnement des classes. Ce partitionnement utilise une mesure de similarité entre un nœud et un cluster de nœuds basée sur la famille de classifieurs qui sera utilisée dans l'arbre. L'idée est d'utiliser ensuite un algorithme dérivé de l'algorithme des K-moyennes dans le but de former itérativement des partitions en accord avec le type de classifieurs qui seront utilisés par la suite dans les nœuds de la hiérarchie. Le second travail est une méthode active de sélection de classifieurs durant le processus de classification d'un exemple. L'idée

est de choisir optimalement les classifieurs apportant le plus d'information pour résoudre la tâche de classification pour un exemple.

En probabilisant l'appartenance de l'exemple aux différentes classes, notre algorithme propose d'éliminer dynamiquement les classes les moins vraisemblables au fur et à mesure des itérations de l'algorithme de façon à se concentrer sur les classes les plus dures à départager. Ainsi, l'utilité de chaque classifieur sélectionné est optimale selon notre mesure de gain d'information et la classification est plus rapide.

1.4 Plan

Nous verrons tout d'abord dans le Chapitre 2 l'état de l'art du domaine qui nous intéresse. Nous présenterons le plus exhaustivement possible les méthodes qui ont conduit à l'état actuel des méthodes permettant d'accélérer le processus de classification. Nous détaillerons les méthodes fondamentales nécessaires à la bonne compréhension du domaine. Nous allons ensuite faire une synthèse des méthodes de l'état de l'art par l'introduction d'un nouveau formalisme dans le Chapitre 3. Le Chapitre 4 présente les travaux et les contributions apportés au domaine des modèles de construction hiérarchique. Le Chapitre 5 correspond à notre contribution dans le domaine des méthodes actives de classification. Finalement, nous conclurons ce travail Chapitre 6.

Chapitre 2

Le problème de la classification multi-classes

2.1 Introduction

Nous allons présenter dans ce chapitre le problème de la Classification Multi-Classes ainsi que les principales méthodes qui ont été développées pour le résoudre. Dans un premier temps, nous allons formaliser le problème de classification et définir les notations que nous allons utiliser.

Dans un second temps, nous allons présenter l'état de l'art du domaine, et plus particulièrement les méthodes applicables dans un contexte de classification extrême avec beaucoup de catégories. Tout d'abord, nous verrons les modèles ensemblistes utilisant un ensemble de classifieurs. Ensuite, nous présenterons les travaux relatifs à l'apprentissage de hiérarchies de classifieurs.

Ces méthodes sont naturellement adaptées à la problématique de classification extrême. Nous verrons ensuite une troisième famille de méthodes cherchant à apprendre un espace latent pour les classes permettant de résoudre plus facilement le problème de classification dans un grand nombre de classes.

2.2 Formalisation de la tâche de classification

La tâche de classification est une tâche d'apprentissage statistique qui cherche à assigner une classe y à des exemples x représentés sous la forme d'un vecteur. Les exemples x_i sont de dimension D : $x_i \in \mathbb{R}^D$; et à chaque exemple x_i est associée une classe y_i tel que : $y_i \in \mathcal{L}$ avec $\mathcal{L} = \{\ell_k\}_{k\in[\![1,K]\!]}$ où K représente le nombre de classes différentes du problème. Le problème d'optimisation peut s'écrire d'une façon générale comme la minimisation du risque associé théorique de classification :

$$h^* = \arg\min_h R(h)$$

avec

$$R(h) = \int \mathbb{1}(h(x) \neq y)\mathrm{d}P(x,y)$$

où le but est de trouver la fonction h qui minimise le nombre d'erreurs sur l'ensemble des exemples x possibles. La distribution des données est en général inconnue et on travaille le plus souvent avec un sous-ensemble des exemples possibles : l'ensemble d'apprentissage $\mathcal{D} = \{(x_i, y_i)\}_{i\in[\![1,n]\!]}$. Le risque empirique associé est une mesure de la qualité de classification du modèle sur l'ensemble d'apprentissage. Tout comme le risque théorique, ce risque empirique est calculé en comptabilisant le nombre d'erreurs de classification, c'est à dire le nombre de fois où le modèle a prédit une classe différente de la classe correcte : $h(x_i) \neq y_i$ avec $h(x_i)$ la classe prédite pour le modèle : $h : \mathcal{X} \to \mathcal{L}$.

Le risque empirique s'écrit comme :

$$R_{\mathcal{D}}(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(h(x_i) \neq y_i)$$

Sous les bonnes hypothèses, le risque empirique converge vers le risque théorique. Les conditions de convergence pour une famille de classifieurs caractérisent les capacités de généralisation de cette famille. En pratique, pour évaluer cette capacité de généralisation du modèle de prédiction sur de nouvelles données, un ensemble de *test* $\mathcal{D}_t$ est formé avec un ensemble d'exemples disjoint de l'ensemble d'exemples d'entraînement $\mathcal{D}$.

2.2.1 Le modèle linéaire

La résolution de la tâche de classification multi-classes est principalement basée sur les travaux développés pour résoudre la tâche de classification binaire. Ce problème fondamental de l'apprentissage statistique a été très étudié et de nombreux travaux ont permis de le modéliser de façon précise. Les schémas de classification multi-classes classiques sont essentiellement constitués d'un ensemble de classifieurs binaires simples qui sont combinés de façon à former un classifieur plus complexe capable de classifier parmi plus que deux classes.

Le risque empirique R n'étant pas directement optimisable, on lui substitue une fonction de coût qui est une borne supérieur de ce risque. En pratique, cette fonction de coût associe un terme générique d'erreur (L) avec un terme de régularisation [Cortes and Vapnik, 1995] $(\lambda||h||)$:

$$R_{\mathcal{D}}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x_i), y_i) + \lambda||h||$$

Le problème d'optimisation recherchant le modèle optimal h^* parmi l'ensemble de la classe d'hypothèses $\mathcal{H}$ peut ainsi être formalisé de la façon suivante :

$$h^* = \arg\min_{h\in\mathcal{H}} R_{\mathcal{D}}(h)$$

La famille de classes d'hypothèses la plus simple est la classe des hypothèses linéaires. Une hypothèse de cette classe est paramétrée par un vecteur $\mathbf{w}$ tel que $h_{\mathbf{w}}(x) = \mathbf{w}^T x + b$ avec b le terme de biais.

Différentes fonctions de coût L peuvent être utilisées au lieu du classique *zero-one loss* décrit précédemment $\big(L_{0/1}(h(x_i, y_i)) = \mathbb{1}(h(x_i) \neq y_i)\big)$ qui n'amène pas à un problème d'optimisation convexes. On peut noter deux principales fonctions de coût permettant d'obtenir des problèmes d'optimisation convexe : le *Hinge loss* et le *Logistic loss* :

$$L_{Hinge}(h(x_i, y_i)) = \max(0, 1 - y \cdot h(x)) \tag{2.1}$$

$$L_{Logistic}(h(x_i, y_i)) = \log(1 + \exp(y \cdot h(x))) \tag{2.2}$$

Les SVM (pour *Support Vector Machine*) [Cortes and Vapnik, 1995] utilisent le *Hinge loss* comme fonction de coût de substitution. Le problème d'optimisation peut être résolu par descente de gradient stochastique [Bottou and Bousquet, 2008]. Les SVM linéaires et la regression logistique sont souvent utilisés comme briques de base pour les modèles plus complexes nécessaires à la classification multi-classe. Les deux briques ont des performances similaires sur les problèmes de classification binaires.

Les schémas de classification comme l'OAA (pour *One-Against-All* — Un-Contre-Tous), l'OAO (pour *One-Against-One* — Un-Contre-Un) ou les ECOC (pour *Error Correcting Output Code* — Codes Correcteurs d'Erreurs) sont des exemples courants de méthodes ensemblistes utilisant des classifieurs binaires simples dans le but de former un classifieur plus complexe multi-classe. Il a ainsi été montré [Rifkin and Klautau, 2004] comment l'utilisation de SVM comme classifieurs simples de ces méthodes permet d'établir les performances de l'état de l'art sur des problèmes variés. Nous verrons plus tard en détail ces schémas ensemblistes de classification multi-classe.

2.2.2 Limites dans le contexte d'extreme classification

2.2.2.1 Caractérisation du Big Data

À l'origine, les méthodes de classification multi-classes ont été développées dans le but de résoudre des problèmes avec un nombre de classes limité à quelques dizaines. Comme nous l'avons vu, les problèmes de classification extrême apportent des nouvelles problématiques pour l'apprentissage statistique. Pour la classification, il y a trois caractéristiques qui, lorsqu'elles sont significativement plus grandes que la plupart des autres problèmes, donnent un caractère massif aux données : le nombre d'exemples, la dimension de l'espace des exemples (le nombre de caractéristiques) et le nombre de classes.

Nombre d'exemples. Le nombre d'exemples n correspond à la quantité d'instances à notre disposition. Cela correspond par exemple au nombre de comptes qui sont enregistrés sur Facebook ou bien au nombre d'articles sur le site internet Wikipédia. Généralement, il est intéressant d'avoir plus de données pour la phase d'apprentissage car le modèle appris peut alors plus facilement généraliser s'il a eu un aperçu plus complet des exemples possibles qu'il sera amené à classifier plus tard. Cependant, il peut être plus

difficile de manipuler ces larges volumes de données ($n >> 1$) surtout pour des modèles nécessitant de charger tous les exemples dans une matrice pour la phase d'apprentissage. Les problèmes sont surtout en inférence pour le nombre d'exemples.

Espace dimensionnel. La dimension de l'espace des données correspond au nombre de caractéristiques de chaque exemple. L'âge d'un utilisateur de Facebook ou le décompte d'un mot particulier comme *voiture* dans un article de Wikipédia sont des caractéristiques. Plus il y a de caractéristiques, plus vaste est l'information que l'on possède pour chaque exemple. Si on reprend l'exemple de l'utilisateur Facebook, le nombre de caractéristiques différentes sera réduit. A l'inverse, le nombre de mots différents décrivant un document textuel comme un article Wikipédia est très important ($d >> 1$). Comme chaque mot possède son propre décompte, la taille de l'espace dimensionnel d'un document textuel peut dépasser facilement le million de dimensions. Pour un modèle linéaire, c'est autant de caractéristiques qu'il va falloir pondérer pour chacune des classes ce qui peut rendre l'apprentissage des paramètres du modèle plus complexe quand la représentation des exemples est dense dans cet espace de grande dimension.

Nombre de classes. La dernière caractéristique correspond au nombre de classes (K) associé aux données. C'est le nombre de catégories différentes qui caractérisent une donnée. Dans une classification binaire, il n'y a que deux classes. Pour la reconnaissance d'un chiffre, il y a 10 classes. Lorsque l'on parle du nombre de catégories dans Wikipédia, le nombre de classes peut atteindre plusieurs millions ($K >> 1$). C'est précisément ce point qui nous intéresse dans le travail présenté ici.

Au delà de l'aspect massif, un autre challenge concerne la complexité des données qui peut par exemple concerner une structure de dépendance sur les classes. On parle alors de classification structurée.

2.2.2.2 Exemple de bases de données

Ces bases de données de grandes dimensions sont de plus en plus présentes dans le paysage de l'apprentissage statistique. Par exemple, un site comme Wikipédia a à sa disposition un grand volume de données qui peuvent être associées à un grand nombre de catégories. De la même façon, un projet comme l'*Open Mozilla Directory* a pour but

de répertorier l'ensemble des sites internet. On imagine bien les volumes de données que cela représente ainsi que la complexité de la tâche de catégorisation quand le nombre de catégories est très élevé.

Le tableau 2.1 montre les caractéristiques de ces deux datasets, ainsi que d'un dataset nommé Sector [McCallum and Nigam, 1998; Choromanska and Langford, 2014] qui est d'une taille plus petite. Cela permet d'avoir une idée de ce qui était considéré comme un dataset avec beaucoup de classes il n'y a pas si longtemps. Ces données ont été utilisées dans les travaux présentés ici. Nous avons été amené à travailler avec des versions réduites de ces datasets pour accélérer les expériences qui pouvaient prendre trop de temps avec les datasets entier [1].

TABLE 2.1: Statistiques de bases de données.

	DMOZ	DMOZ (réduit)	Wikipedia (réduit)	Sector
# exemples d'entraînement	93805	9400	15000	6992
# exemples de validation	34905	3500	3000	1469
# exemples de test	34880	3500	3000	1158
# caractéristiques	347255	347255	857452	55198
# classes	12294	1000	1000	105

Habituellement, ce genre de base de données fait face à des problèmes de déséquilibre de représentativité entre les classes. Beaucoup de classes sont représentées par un petit nombre d'exemples. La Figure 2.1 montre le déséquilibre existant dans le dataset DMOZ. Ainsi, une grande partie des classes sont décrites avec très peu d'exemples ce qui peut poser des problèmes pour la reconnaissance de ces classes.

2.2.2.3 Un nouveau paradigme

Ce contexte particulier appelé *Classification Extrême* où le nombre de catégories peut dépasser un million définit un nouveau champ de problématiques. En effet, le fait de travailler avec autant de catégories amène de nouveaux problèmes comme l'augmentation du temps d'apprentissage et du temps de classification d'un nouvel exemple. De plus, la difficulté inhérente à la tâche de classification parmi des centaines de milliers ou des millions de classes rend certaines méthodes classiques non adaptées à ce nouveau paradigme.

1. Nous avons tiré au hasard une classe, puis nous avons calculé les 999 plus proches classes en utilisant une mesure de similarité cosinus entre les centroïdes des classes.

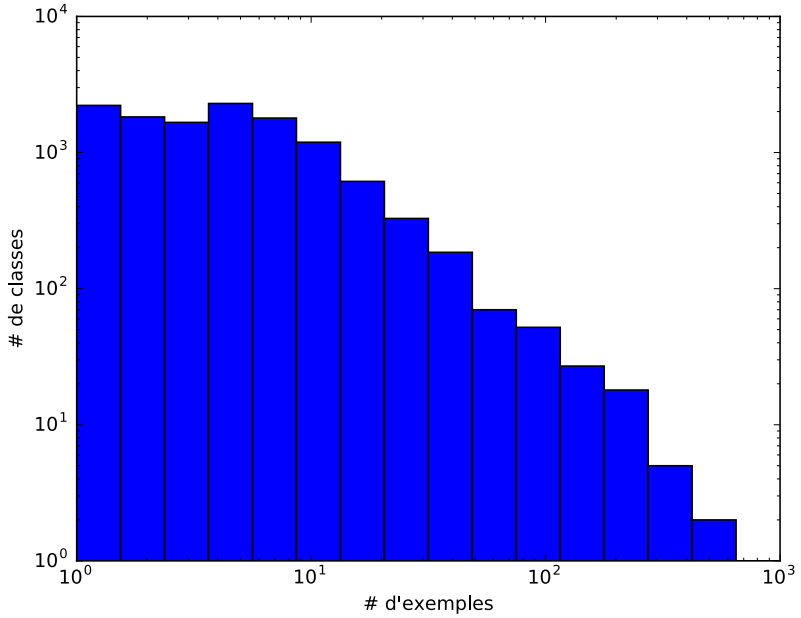


FIGURE 2.1: Histogramme de la répartition des classes en fonction du nombre d'exemples associé à chacune des classes pour le dataset DMOZ. Une grande majorité des classes est ainsi dotée de moins de 10 exemples. Ce déséquilibre fait partie intégrante de ce type de base de données avec un grand nombre de classes.

Illustrons ces nouvelles problématiques plus précisément. Un particulier navigue sur le site internet Facebook et il ne souhaite voir que du contenu adapté à ses centres d'intérêts. Pour cela, Facebook a besoin d'être capable de catégoriser les contenus produits par tous les autres utilisateurs. Autre exemple : le site encyclopédique participatif Wikipédia a un nouveau document à intégrer dans son encyclopédie. Il est nécessaire de catégoriser correctement le document pour faciliter son accessibilité. Dans le domaine du traitement d'images, le problème de reconnaissance de visage se heurte à ces problèmes de classification extrême. Apprendre un modèle capable de catégoriser rapidement n'importe quel visage parmi 7 milliards d'individus est un problème très complexe.

Ces tâches particulières demandent ainsi de nouvelles méthodes de classification capables de gérer la classification dans un très grand nombre de catégories.

2.2.2.4 Limites des méthodes usuelles

Une méthode classique comme le schéma de classification OAA que nous avons déjà évoqué a besoin d'apprendre un classifieur par classe. La concaténation des réponses de tous ces classifieurs binaires simples permet d'obtenir une classification plus complexe

capable de discriminer une classe parmi toutes les autres classes. Cela signifie que le modèle doit apprendre un nombre de classifieurs égal au nombre de classes ce qui est une opération qui peut être coûteuse lorsque le nombre de classes est important. De plus, pour l'utilisation du modèle, il faut calculer pour chaque nouvel exemple à classifier le résultat de la classification binaire de tous les classifieurs qui composent ce schéma de classification. Un schéma de classification de type OAA a une complexité de classification en $O(K)$ selon le nombre de classifieurs binaires. Certains schémas sont encore plus coûteux comme le OAO où un classifieur binaire est appris pour chaque paire de classe qu'il est possible de former. Le coût d'une classification dans ce cas est de l'ordre de $O(K^2)$, ce qui rend inutilisable ces méthodes dans ce cadre de classification extrême. Un des principaux buts du travail présenté ici est de développer des méthodes capables de prédire une catégorie en un temps sous linéaire avec comme cible $O(\log(K))$ qui est la complexité minimale que l'on peut espérer tout en assurant une certaine précision de la prédiction. Il va s'agir ainsi d'étudier le dilemme entre la précision de la classification et la complexité de cette classification.

2.2.3 Le classifieur simple

Le classifieur simple est une brique élémentaire pour la plupart des méthodes que nous allons présenter par la suite. Nous allons ainsi définir les notations relatives au classifieur simple dans le cadre d'une classification multi-classe.

On définit les exemples appartenant à la même classe ℓ par l'ensemble $\mathcal{X}_{\{\ell\}}$:

$$\mathcal{X}_{\{\ell\}} = \{x_i\}_{\{i=1\ldots n | y_i=\ell\}}$$

Ces ensembles $\mathcal{X}_{\{.\}}$ sont définis pour n'importe quel sous-ensemble de classes $\{\ell_i\}$. On note h la fonction de décision associée au classifieur simple. On note par $\mathcal{C}^+$ et $\mathcal{C}^-$ les deux sous-ensembles de classes ($\mathcal{C} \subset \mathcal{L}$) à séparer tel que $h_{\mathcal{C}^+,\mathcal{C}^-}$ correspond à la fonction de décision du classifieur ayant été entraînée à séparer les ensembles d'exemples de l'ensemble $\mathcal{X}_{\mathcal{C}^+}$ (annoté positivement) des exemples de l'ensemble $\mathcal{X}_{\mathcal{C}^-}$ (annoté négativement).

En notant $\mathcal{D}_{\mathcal{C}^+,\mathcal{C}^-}$ l'ensemble d'apprentissage issu de $\mathcal{D}$ tel que :

$$\mathcal{D}_{\mathcal{C}^+,\mathcal{C}^-} = \{(x_i, b(y_i)\}_{\{i=1\ldots n |\ y_i \in (\mathcal{C}^+ \cup \mathcal{C}^-)\}}$$

avec

$$b(y_i) = \begin{cases} +1 & \text{si } y_i \in \mathcal{C}^+ \\ -1 & \text{si } y_i \in \mathcal{C}^- \end{cases}$$

le risque empirique associé au classifieur simple séparant l'ensemble $\mathcal{C}^+$ de l'ensemble $\mathcal{C}^-$ est le suivant :

$$R_{\mathcal{D}}(h_{\mathcal{C}^+,\mathcal{C}^-}) = \frac{1}{n_{\mathcal{C}^+,\mathcal{C}^-}} \sum_{(x,y)\in\mathcal{D}_{\mathcal{C}^+,\mathcal{C}^-}} \mathbb{1}\left(h(x) \neq y\right)$$

avec $n_{\mathcal{C}^+,\mathcal{C}^-} = |\mathcal{D}_{\mathcal{C}^+,\mathcal{C}^-}|$ (la cardinalité de l'ensemble des exemples concernés).

2.3 Méthodes ensemblistes

Tout d'abord, intéressons-nous au cadre des méthodes ensemblistes. Ces méthodes se basent sur l'apprentissage d'un ensemble de classifieurs dont l'ensemble des réponses sera pris en compte pour prédire la classe d'un nouvel exemple.

2.3.1 One-Against-All

Le schéma de classification de référence est le OAA (pour *One-Against-All* — Un-Contre-Tous) [Scholkopf et al., 1995; Vapnik, 1998]. Ce schéma classique de classification multi-classes est reconnu comme une méthode robuste [Rifkin and Klautau, 2004], simple à mettre en place et capable de gérer des problèmes volumineux [Perronnin et al., 2012]. Comme vu précédemment, ce schéma utilise un ensemble de classifieurs, chacun cherchant à discriminer une seule classe. L'ensemble des dichotomies Φ_{OAA} constitué par ce schéma est le suivant :

$$\Phi_{OAA} = \{(\ell_k, \mathcal{L}\backslash\{\ell_k\})\}_{k\in[\![1,K]\!]}.$$

Dans un souci d'homogénéisation avec le formalisme présenté Chapitre 3, on définit une mémoire $\mathcal{M}$. Cette mémoire est composée d'un tableau $\mathbf{v} \in \mathbb{R}^K$ dont chaque valeur est associée à une classe. Durant le processus de classification, chaque case du tableau k est mise à jour avec le résultat de la fonction de décision du classifieur ℓ_k pour l'exemple en cours de classification : $\mathbf{v}[k] = h^*_{\Phi_{OAA}[k]}(x)$. Ces valeurs indiquent la confiance dans

chacune des prédictions. La prédiction p est obtenue en retournant la classe $\tilde{\ell}$ telle que :

$$p(\mathcal{M}) = \ell_{\tilde{k}} \text{ tel que } \tilde{k} = \arg\max_k \mathbf{v}[k].$$

Cette méthode a l'avantage d'être parallélisable car les apprentissages et utilisations des classifieurs sont indépendants les uns des autres.

2.3.2 One-Against-One

Le schéma de classification OAO (pour *One-Against-One* — un-contre-un) [Knerr et al., 1990] est assez semblable au schéma de classification OAA. Le principe est de constituer un ensemble de classifieurs correspondant à l'ensemble des paires de classes qu'il est possible de constituer dans $\mathcal{L}$. Ainsi :

$$\Phi_{OAO} = \{(\ell_k, \ell_{k'})\}_{\{(k,k')\in\mathcal{L}^2 \text{ et } k\neq k'\}}$$

Pour la mémoire $\mathcal{M}$, on utilise un vecteur $\mathbf{v} \in \mathbb{N}^K$ qui va comptabiliser le nombre de fois qu'une classe a *gagné* contre n'importe quelle autre classe. On a donc :

$$\mathbf{v}[k'] = \sum_{k\in\{\mathcal{L}\setminus k'\}} \mathbb{1}(h^*_{k' \, vs. \, k}(x) > 0)$$

Originellement, les réponses des classifieurs étaient agrégées par l'utilisation de l'opérateur ET [Knerr et al., 1990]. Friedman [1996] a proposé à la place un système de vote majoritaire : la classe qui a reçu le plus grand nombre de votes est celle qui est prédite : $\ell_{\tilde{k}}$ tel que $\tilde{k} = \arg\max_k \mathbf{v}[k]$.

Le principal souci avec cette méthode est son coût. Elle est gourmande en mémoire car il faut stocker $\frac{K(K-1)}{2}$ classifieurs différents. De plus, elle n'est pas rapide : sa complexité de classification est en $O(K^2)$. Ainsi cette méthode n'est pas du tout adaptée pour des grandes valeurs de K. Nous verrons par la suite comment cette méthode a été adaptée par Platt et al. [2000] pour répondre aux contraintes de temps d'inférence en organisant hiérarchiquement ces classifieurs binaires.

2.3.3 Error Correcting Output Codes

Les codes correcteurs d'erreurs ou ECOC (pour *Error Correcting Output Codes*) [Dietterich and Bakiri, 1995] ont été introduits pour augmenter la robustesse des méthodes comme l'OAA. Au lieu d'utiliser un codage un parmi K comme OAA, chaque classe est codée par un mot de T bits avec $T < K$ (le nombre de classes). Chaque bit réalise une partition de l'ensemble des classes en deux groupes. Ce sont ces T classifieurs binaires que l'on apprend. En inférence, on calcule les sorties de ces T classifieurs et on utilise un plus proche voisin par rapport aux codes de classes.

Deux types de codes peuvent être constitués. Le premier type consiste, comme l'ensemble précédent, en des dichotomies *denses* où toutes les classes sont présentes dans un des deux ensembles de classe. Le second consiste à autoriser un codage *sparse* [Allwein et al., 2001] permettant de ne coder que pour un sous-ensemble de classes. Dans ce cas, les codes correspondent à des trichotomies (partition ternaire) et non plus à des dichotomies.

Le codage des dichotomies peut se visualiser comme une matrice $\mathcal{C} = \{-1, 0, 1\}^{K \times T}$ (dans le cadre d'un codage ternaire, i.e. *sparse*) où T est le nombre de trichotomies. La figure 2.2 montre un exemple de matrice de codage $\mathbf{C}$.

Chaque colonne code pour une partition ternaire des classes :

1 : la classe $\in \mathcal{C}^+$

-1 : la classe $\in \mathcal{C}^-$

0 : cette classe n'est pas prise en compte

où $\mathcal{C}^+$ et $\mathcal{C}^-$ correspondent aux ensembles des classes utilisé par le classifieur. Pour plus de clarté dans la suite de ce manuscrit, on continuera à parler de dichotomie $\{\mathcal{C}^+, \mathcal{C}^-\}$ qui définit complètement une trichotomie $\{\mathcal{C}^+, \mathcal{C}^-, \mathcal{C}^0\}$ car

$$\mathcal{C}^+ \cup \mathcal{C}^- \cup \mathcal{C}^0 = \mathcal{L}$$

$$\text{et} \quad \mathcal{C}^+ \cap \mathcal{C}^- = \emptyset, \;\; \mathcal{C}^+ \cap \mathcal{C}^0 = \emptyset, \;\; \mathcal{C}^- \cap \mathcal{C}^0 = \emptyset$$

$$\text{On a donc :} \;\; \mathcal{C}^0 = \mathcal{L} \backslash \{\mathcal{C}^+ \cup \mathcal{C}^-\}$$

Le processus de classification d'un ECOC va stocker dans sa mémoire $\mathcal{M}$ un *code* $v \in \{0, 1\}^T$ dans sa version classique (et $v \in \mathbb{R}^T$ dans une version plus évoluée). Ce code

	Dichotomie #1	Dichotomie #2	Dichotomie #3	...
ℓ_1 :	1	0	-1	.
ℓ_4 :	1	0	1	.
ℓ_2 :	-1	1	1	.
ℓ_3 :	0	0	-1	.
...	.	.	.	...

FIGURE 2.2: Exemple de matrice de codage $\mathbf{C}$ des dichotomies dans le cadre d'un modèle ECOC.

enregistre le résultat de tous les classifieurs de l'ensemble Φ des dichotomies qui ont été générées : $\mathbf{v}[k] = h^*_{\Phi[k]}(x)$.

La prédiction est obtenu en comparant le vecteur $\mathbf{v}$ obtenu avec le *code* de chaque classe (codes obtenus en isolant les lignes de la matrice de codage $\mathcal{C}$). En effet, chaque classe (ou label) ℓ_k possède un code propre $\mathcal{C}[k,:]$. Dans la suite de ce manuscrit, les termes *label* et *classe* seront indépendamment utilisés et dénoteront la même chose. Finalement, le label prédit est celui minimisant la distance de Hamming entre son code et celui obtenu par l'exemple à classifier : $\ell_{\tilde{k}}$ tel que $\tilde{k} = \arg\max_k \#\{i : \mathbf{v}[i] \neq \mathcal{C}[k,i]\}$

Étant donné la façon dont est décodé le signal obtenu pour la classification d'un nouvel exemple, il est important que la distance entre les codes des classes soit la plus grande possible de façon à éviter des erreurs de confusion entre deux classes (le code d'une classe correspond à une ligne de la matrice de codage $\mathcal{C}$). C'est un point essentiel du codage ECOC. Une façon simple pour générer cette matrice est de générer un grand nombre de matrices différentes et de sélectionner celles qui maximisent la distance de Hamming entre les codes des classes. Une autre façon consiste à faire en sorte d'avoir une matrice de Hadamard [Langford and Beygelzimer, 2005] ce qui garantit une distance minimale entre deux codes de $\frac{K}{2}$.

Les codes correcteurs d'erreurs garantissent une bonne prédiction finale tant que le nombre d'erreurs reste en dessous d'un certain seuil. En effet, en notant δ la distance minimale entre toutes les paires de codes de la matrice de codage $\mathcal{C}$, l'inférence de la classe d'un exemple sera correcte tant que le nombre d'erreurs sur l'ensemble des classifieurs sera inférieur à $\frac{\delta}{2}$.

Il faut aussi s'assurer que les codes des colonnes de la matrice $\mathcal{C}$ soient le plus éloigné les uns des autres. Si cela n'est pas respecté, il y a un risque de forte corrélation des erreurs des classifieurs dû à la proximité des dichotomies, ce qui n'aide pas à contenir le nombre

d'erreurs en dessous de $\frac{\delta}{2}$. Le dernier point clé est la qualité des classifieurs qui sont appris à partir des dichotomies Φ. Dans certains cas, les dichotomies sont tirées au hasard ce qui peut induire des problèmes de classification très complexes. L'utilisation de classifieurs trop simples dans ce cas abouti le plus souvent à un trop grand nombre d'erreurs de classification. Ainsi, il est recommandé d'adapter la famille $\mathcal{H}$ des classifieurs à utiliser selon la difficulté des dichotomies à résoudre. Les réseaux de neurones ou les machines à noyaux sont des familles de modèles [Dietterich and Bakiri, 1995] assez flexibles pour être adaptés à différents types de problèmes.

Allwein et al. [2001] à développé une méthode alternative qui utilise les confiances données par les classifieurs simples binaires en plus du résultat de la classification binaire +1/-1. Par exemple, avec l'utilisation de SVM, la distance de l'exemple à l'hyperplan de chaque classifieur est utilisée pour le décodage du code de l'exemple et la classification de ce dernier. Cette méthode alternative de décodage permet d'augmenter significativement la précision des ECOC tant que la valeur de confiance donnée par la classification est bien estimée et qu'elle peut être comparée avec les autres valeurs de confiance.

D'autres méthodes ont cherché à améliorer la phase de codage des classifieurs. Au lieu de générer aléatoirement des matrices, différents travaux ont cherché à coder automatiquement les dichotomies des classifieurs [Gao and Koller, 2011a; Schapire, 1997; Cissé et al., 2012; Zhao and Xing, 2013]. Schapire [1997] a eu l'idée de combiner l'approche ensembliste Boosting avec l'approche ECOC. Le Boosting fonctionne avec un ensemble de classifieurs faibles qui sont appris sur différentes distributions des exemples. Le but étant de spécialiser les classifieurs sur les exemples difficiles à classifier. L'idée est de combiner les deux approches : construire itérativement les dichotomies tout en agissant sur la distribution de l'ensemble d'apprentissage et sur la dichotomie de façon à minimiser l'erreur d'estimation de l'ensemble d'apprentissage. En reprenant cette idée de coopération des classifieurs faibles par boosting, Gao and Koller [2011a] introduisent une fonction de coût multi-classes [Crammer and Singer, 2002a] basée sur le *Hinge-Loss*. Quand une information hiérarchique est disponible, il peut être intéressant de s'en servir pour aider à la construction de problèmes de séparabilité de classes moins difficiles à résoudre. Ainsi, les classifieurs sont plus précis et la prédiction finale est meilleure. Cissé et al. [2012] à utilisé ainsi cette information de similarité entre classes $\mathbf{s}_i = [s(\ell_i, \ell_1), \ldots, s(\ell_i, \ell_K)]$ de façon à projeter ces vecteurs $\mathbf{s}_i \in \mathbb{R}^K$ dans un espace de dimension réduit $\mathbb{R}^T$ avec $T << K$.

L'espace de dimension réduit est appris grâce à un autoencodeur :

$$\arg\min_{\mathcal{W}} \sum_{i=1}^{K} ||\mathbf{s}_i - \mathcal{W}^T \times f(\mathcal{W} \times \mathbf{s}_i)||^2)$$

Le codage d'une classe donnée ℓ_i (correspondant à la ligne i de la matrice de codes : $\mathbf{C}_i$) est obtenu en calculant $\mathbf{C}_i = f(\mathcal{W} \times \mathbf{s}_i)$. Pour s'assurer d'une bonne séparation des codes de chacune des classes, une contrainte est ajoutée au problème de l'autoencodeur :

$$\forall (i,j), i \neq j : ||f(\mathcal{W} \times s_i) - f(\mathcal{W} \times s_j)|| \geq b$$

Un travail intéressant dans notre contexte de classification extrême a été proposé par Zhao and Xing [2013]. Ils ont proposé d'apprendre la matrice de codage en résolvant le problème suivant :

$$\arg\max_{\mathbf{C}} F_b(\mathbf{C}) - \lambda_r F_r(\mathbf{C}) - \lambda_c \sum_{i=1}^{T} ||\mathbf{C}_{(i)}||_2^2$$

avec F_b une fonction calculant la séparabilité de chaque dichotomie (et donc la précision résultante du classifieur correspondant), F_r une fonction calculant la séparabilité de codes des colonnes de la matrice de codage et $\mathbf{C}_{(i)}$ la colonne i de la matrice de codage $\mathbf{C}$. Cette fonction permet d'estimer le degré de corrélation entre les classifieurs. On apprend à minimiser l'erreur de classification F_b en minimisant la séparabilité des codages de classes F_r. La norme sur les colonnes $\mathbf{C}_{(i)}$ contrôle la complexité des classifieurs induits par la matrice de codage $\mathbf{C}$. La contrainte d'optimisation dans l'ensemble des entiers est relaxée pour autoriser à résoudre le problème dans l'espace des réels. Malgré la difficulté du problème d'optimisation avec une fonction non-convexe, cette solution a montré être capable de réduire l'erreur d'estimation (par opposition à l'erreur d'approximation).

2.3.4 Discussion

Les schémas de classification ensemblistes permettent d'agréger *à plat* un ensemble de classifieurs où chacun d'eux a un rôle similaire dans le processus de prédiction. Il n'y a ni ordre, ni hiérarchie dans cet ensemble de classifieurs simples. Comme il n'y a pas de processus séquentiel, le processus de décision n'est pas spécifique à l'exemple et tout les classifieurs sont utilisés à chaque fois pour tous les exemples. De là vient la principale

remarque que l'on peut faire pour ces méthodes : il arrive souvent que certains classifieurs n'aient rien à apporter à la décision.

Dans notre contexte de classification extrême, il est important d'éviter l'utilisation de classifieurs non informatifs.

Cette remarque met particulièrement en relief l'attrait des méthodes dites hiérarchiques : ces méthodes ont la particularité de proposer un *chemin* de classification qui va être spécifique à chaque exemple à classifier.

2.4 Approches hiérarchiques

Les approches hiérarchiques représentent le deuxième grand groupe des schémas de classification. A la différence du schéma de classification ensembliste, les approches hiérarchiques se basent sur l'organisation d'une hiérarchie de classifieurs [Cai and Hofmann, 2004; Liu et al., 2005a; Vural and Dy, 2004; Godbole et al., 2002; Chen et al., 2004; Babbar and Partalas, 2013; Liu et al., 2012]. Le processus de classification décrit un chemin dans l'arbre allant de la racine à un nœud feuille. À chaque nœud de l'arbre correspond un classifieur qui oriente l'exemple vers un de ses fils. Les feuilles correspondent aux étiquettes de classes. Différentes options sont possibles, une étiquette par feuille ou plusieurs étiquettes avec une redondance éventuelle par feuille. Il y a fondamentalement deux aspects dans le problème de la classification hiérarchique. Le premier problème vient de l'apprentissage de la hiérarchie de classes dans le cas où cette hiérarchie est inexistante ou incomplète. Le deuxième vient de l'apprentissage des fonctions de décision des nœuds de l'arbre qui utilisent la hiérarchie de classe.

Ce type de modèle basé sur l'organisation hiérarchique de classifieurs est apparu avec le besoin de catégoriser de nouveaux documents au sein d'ontologies existantes de documents [Koller and Sahami, 1997; Chakrabarti et al., 1997]. Initialement, ces ontologies n'étaient pas forcément très grandes ($K < 1000$), et l'aspect accélération du processus de classification n'était pas au cœur de la tâche.

Le but de la tâche était de classifier les exemples parmi l'ensemble de classes issues d'une hiérarchie de classes. Cette problématique a ainsi conduit naturellement vers l'utilisation de modèles hiérarchiques permettant d'imiter l'organisation des classes par l'utilisation

d'un arbre de classifieurs — bien que l'utilisation de cette information hiérarchique pré-existante n'ait pas toujours conduit à construire une hiérarchie de classifieurs [Bengio et al., 2010; Cissé et al., 2012; Wang, 1999]. Les modèles hiérarchiques ont rapidement convergé vers une structure à base de classifieurs binaires puissants correspondant aux séparations entre les classes à chaque nœud de la hiérarchie [Bengio et al., 2010; Liu et al., 2005a; Vural and Dy, 2004; Chen et al., 2004].

Nous allons commencer par formaliser le concept d'arbre de classifieurs en décrivant les 3 composantes d'un tel modèle. Nous décrirons ensuite l'état de l'art des méthodes hiérarchiques.

2.4.1 Formalisation du modèle hiérarchique

Un arbre T peut se formaliser par un quadruplet (N, E, A, F) :

- $N = \{n_0, n_1, \ldots, n_U\}$ correspond à l'ensemble des nœuds de l'arbre. Pour un arbre binaire parfait, le nombre de nœuds dans l'arbre est égal à $U = 2K - 1$.
- $E = \{(i, j), \ldots\}$ correspond à l'ensemble des arcs reliant deux nœuds (parent, enfant).
- $A = \{s_1, \ldots, s_U\}$ correspond à l'ensemble des labels présents dans chacun des nœuds. Par convention, le nœud 1 est le nœud racine : ainsi $s_1 = \mathcal{L}$. Chaque nœud feuille e est associé à un unique label : $|s_e| = 1$. Chaque ensemble de label d'un nœud fils est un sous-ensemble de l'ensemble de label du nœud père : $\forall (p, c) \in E$, $s_c \subseteq s_p$
- $F = \{f_2, \ldots, f_U\}$ correspond à l'ensemble des fonctions de décision présentes dans chaque nœud (sauf le nœud racine). Ces fonctions de décision sont celles qui sont utilisées pour *diriger* un exemple à classifier à travers l'arbre. Un nœud père calcule toutes les fonctions de décision des nœuds fils de façon à affecter l'exemple à un fils unique.

Le processus d'inférence hiérarchique classique d'un nouvel exemple commence au nœud racine qui contient tous les labels possibles. A chaque étape, l'exemple descend dans l'arc amenant au nœud ayant obtenu la plus grande valeur avec sa fonction de décision appliquée à l'exemple courant [Vural and Dy, 2004]. L'exemple descend à travers tous les niveaux de l'arbre jusqu'à ce qu'il atteigne une feuille. La classe associée à cette feuille

donnera la prédiction finale du processus de classification. L'Algorithme 1 détaille ce processus d'inférence d'un nouvel exemple.

Algorithm 1 Inférence dans un arbre de classifieurs

Entrée: Exemple : x
Entrée: Nœud racine : n_1 d'un arbre $T = (N, E, A, F)\}$
  $a \leftarrow rootNode$
  tant que a n'est pas une feuille **faire**
    $a \leftarrow \arg\max_{\{c\,:\,(a,c)\in E\}} f_c(x)$
  fin tant que
  retourner s_a

Un arbre de classifieurs peut être décomposé en 3 parties distinctes : la structure, l'affectation des labels aux nœuds et les fonctions de décision.

2.4.1.1 Structure d'un arbre

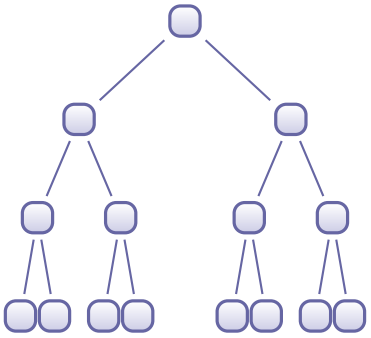
La structure de l'arbre correspond à l'organisation des nœuds et des branches qui constituent l'arbre : N et E.

Si on utilise un arbre binaire, la complexité d'inférence est en $O(\log_2(K))$, i.e $\log_2(K)$ classifieurs sont utilisés pour classer un exemple. En effet, un arbre n-aire équilibré sans redondance de classes peut effectuer une classification en utilisant $n \log_n(K)$ classifieurs. Si l'arbre n'est plus équilibré, la classification peut prendre jusqu'à K classifications dans le pire des cas (structure d'arbre en peigne, voir Figure 2.3). La structure de l'arbre est un paramètre important des méthodes hiérarchiques.

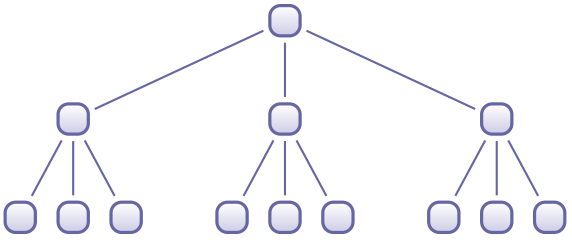
La plupart des méthodes hiérarchiques simplifient le problème d'apprentissage de structure en fixant le nombre exact d'enfants par nœud. Fixer un degré de liberté comme celui-là permet de simplifier l'apprentissage d'une hiérarchie mais réduit sa capacité à coller précisément à une ontologie latente particulière qui existerait entre les classes du problème. Cela peut se révéler problématique lorsque la structure inhérente aux relations entre les classes n'est pas constante sur l'ensemble de la hiérarchie.

2.4.1.2 Affectation des classes

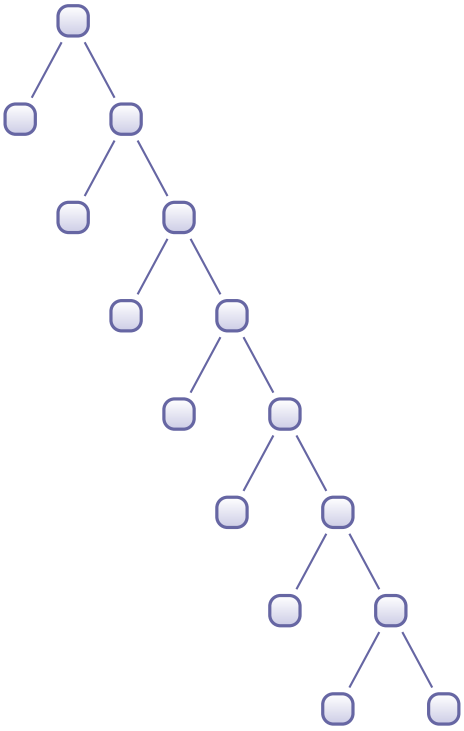
Le problème d'affectation des classes correspond au problème d'étiquetage des nœuds avec les labels de $\mathcal{L}$.



(A) Structure d'arbre binaire.



(B) Structure d'arbre ternaire.



(C) Structure d'arbre dit *en peigne*.

FIGURE 2.3: Différents types de structure d'arbre.

La résolution de ce problème est intrinsèquement liée à la structure de l'arbre. Suivant l'angle adopté pour résoudre le problème d'affectation des classes, le problème peut se réduire à un problème de partitionnement de classes si une approche *de haut vers le bas* (*top-down*) est utilisée. Alternativement, cela peut simplement être un problème d'agrégation de nœuds similaires, dans le cas d'une approche de *bas vers le haut* (*bottom-up*). Le choix de la bonne mesure de partitionnement et d'agrégation est primordial pour l'obtention d'une hiérarchie limitant le nombre d'erreurs.

Il faut distinguer deux types de hiérarchies suivant qu'elles autorisent ou non une certaine redondance dans les assignations de labels dans les nœuds. Un arbre est dit non-redondant si un label ne peut pas se retrouver parmi plusieurs nœuds frères : $\forall(p, c1, c2) \in N^3$et $(p, c1) \in E, (p, c2) \in E, s_{c1} \cap s_{c2} = \emptyset$. Inversement, un arbre non-redondant ne respecte pas cette condition et autorise un même label dans différents nœuds d'un même niveau de l'arbre. Ainsi, un tel arbre est moins sujet aux erreurs de classification car les labels sont redondants et donc de multiples feuilles peuvent se voir assigner la même classe. Cependant, la complexité du processus d'inférence est plus élevée que pour un arbre sans redondance.

2.4.1.3 Fonctions de décision

La dernière composante d'un arbre correspond aux fonctions de décisions de chacun des nœuds. Le moyen le plus utilisé pour apprendre ces classifieurs est de leur faire résoudre le problème de classification donné par le partitionnement des labels entre tous les nœuds fils d'un nœud père donné de l'arbre.

Tous ces classifieurs peuvent être appris indépendamment les uns des autres, ce qui permet de paralléliser l'apprentissage. Certaines méthodes proposent de faire une optimisation globale des paramètres de tous les classifieurs en même temps permettant de mieux prendre en compte la structure de l'arbre.

2.4.2 Apprentissage d'une hiérarchie de labels

De nombreux auteurs se sont intéressés à l'apprentissage de hiérarchie [Bengio et al., 2010; Marszalek and Schmid, 2008; Griffin and Perona, 2008; Deng et al., 2011; Gao and Koller, 2011b].

Le nombre de permutations de labels différentes pour une structure fixe est un problème combinatoire tel qu'il n'est tout simplement pas envisageable d'explorer toutes les combinaisons. Ce problème d'apprentissage d'une hiérarchie de labels regroupe généralement l'apprentissage d'une structure en même temps que l'apprentissage des partitions de labels. Cependant, le plus souvent, un unique paramètre contrôlant le nombre d'enfants par nœud décide de la structure de l'arbre.

2.4.2.1 Méthodes *Top-Down*

Il existe différentes méthodes *top-down* de création de hiérarchie. Les méthodes *top-down* sont dites *gourmandes* (*greedy*) car elles résolvent les problèmes de partitionnement de labels localement, en commençant par le haut de la hiérarchie, sans s'occuper de ce qu'il se passera les étapes suivantes. Les premières méthodes de construction d'arbre de classes s'appuient sur l'utilisation d'algorithmes de clustering comme celui des *K-moyennes* (K-means [Forgy, 1965]) dans le but d'organiser les ensembles de labels en clusters [Vural and Dy, 2004; Liu et al., 2005b; Marszalek and Schmid, 2008]. Ce clustering peut se faire en résolvant une série de problèmes de *max-cut* comme dans le travail de Chen et al. [2004]. Le fonctionnement général de ces méthodes est le suivant : on partitionne l'ensemble des classes au niveau du nœud racine, chaque groupe de classes étant attribué à un nouveau nœud fils de la racine. On répète l'opération en chaque nœud avec l'ensemble de classes qui lui a été attribué. On arête quand un critère est satisfait, par exemple la pureté des nœuds.

Pour l'utilisation de l'algorithme des K-moyennes par exemple, un centroïde est calculé pour chaque classe ℓ_k :

$$\mu_k = \frac{1}{|\mathcal{D}_k|} \sum_{x_i \in \mathcal{D}_k} x_i$$

Ensuite, chaque centroïde est utilisé comme un point dans l'algorithme des K-moyennes qui cherche à résoudre le problème suivant :

$$\arg\min_{\mathbf{s}} \sum_{i=1}^{k} \sum_{\mu_k \in s_i} ||\mu_k - m_i||^2$$

où les m_i représentent les moyennes des points affectés au cluster s_i.

Cette méthode ne garantit cependant pas un équilibre des classes dans les clusters. Il faut de plus donner le nombre de clusters à créer car cette méthode ne règle pas ce paramètre automatiquement. Certains travaux permettent d'aider à définir le nombre de clusters [Hamerly and Elkan, 2004]. Plus il y a de clusters en un nœud, plus il y a de nœuds fils, et plus le temps de classification est long. De plus, comme le nombre de classifieurs à utiliser est aussi proportionnel à la profondeur moyenne de l'arbre, un partitionnement trop déséquilibré entraînera une formation d'un arbre dit *en peigne*. Ce type d'arbre n'est pas intéressant car il rassemble une grande partie du flux d'exemples dans un seul chemin. On perd alors en spécificité des classifieurs pour un exemple donné. De plus, la classification nécessitera plus d'étapes. Il a été montré que pour certains problèmes difficiles de partitionnement, l'utilisation des K-moyennes ne donnait pas de résultats satisfaisant [Ng et al., 2001].

En partant d'un autre point de vue, Marszalek and Schmid [2007] à utilisé l'ontologie de catégories de Wordnet [Fellbaum, 1998] pour construire une hiérarchie de concepts visuels dans le cadre de la classification d'images.

Beygelzimer et al. [2009] à travaillé sur la conception d'un arbre de classes avec une contrainte supplémentaire. Le *Conditional Probability Tree* a un apprentissage d'une complexité logarithmique en fonction du nombre de classes. Le raisonnement derrière cette volonté porte sur le fait qu'il est important de considérer un temps limite pour l'apprentissage des modèles et qu'il ne faut pas seulement regarder le coût de la classification.

Partitionnement par clustering spectral. Bengio et al. [2010] a proposé d'utiliser des méthodes de *clustering spectral* [Ng et al., 2001; Planck and Luxburg, 2006] à la place de l'algorithme des K-moyennes. Le clustering spectral se base sur une matrice de distance entre les points à partitionner. Ici on veut partitionner les classes du nœud parent que l'on souhaite répartir dans les nœuds fils. Le choix de la mesure de distance entre classes est primordial car c'est l'unique information qui est utilisée pour effectuer le partitionnement. Dans le travail de Bengio et al. [2010], la mesure de distance est issue du calcul d'une matrice de confusion *classe à classe* [Godbole et al., 2002]. Un ensemble de classifieurs OAA est appris sur un ensemble d'entraînement. Puis, une matrice de confusion entre classes est complétée grâce aux erreurs de classification d'un ensemble de validation. Les raisons qui ont poussé Bengio et al. [2010] à choisir une matrice de

confusion sont motivées par le fait qu'il est préférable de mettre dans la même partition des classes qui sont difficiles à distinguer les unes des autres. La matrice de distance A, finalement utilisée par l'algorithme de spectral clustering, correspond à une version symétrique de la matrice de confusion C : $A = \frac{1}{2}(C + C^T)$. L'Algorithme 2 montre le mécanisme d'apprentissage du partitionnement de classes utilisé dans cette méthode :

Algorithm 2 Apprentissage de l'arbre de classe

Apprendre K classifieurs Un-Contre-Tous (OAA) : $f_1, \ldots, f_K$
Calculer la matrice de confusion C tel que : $C_{i,j} = |\{x, \ell_i\} \in \mathcal{D}_{validation} : \arg\max_k f_k(x) = j$
Pour chaque Nœud n de l'arbre, depuis la racine jusqu'aux feuilles, partitionner l'ensemble de classes s_n entre les ensembles de classes de ses enfants $L_n = \{s_c : c \in \mathcal{E}_n\}$, où $\mathcal{E}_n = \{c \in N : (n, c) \in E\}$ et $\cup_{c \in N_n} s_c = s_n$ en résolvant :

$$\arg\max_{L_n} = \sum_{c \in \mathcal{E}_n} \sum_{y_p, y_q \in s_c} A_{pq}$$

où

$$A = \frac{1}{2}(C + C^T)$$

Ensuite, le problème de maximisation peut être résolu en utilisant les techniques de spectral clustering. Le spectral clustering permet de résoudre efficacement un problème de *graph-cut*. La matrice de similarité A représente le poids des arêtes entre les nœuds d'un graphe qui symbolisent les classes.

Le problème de *graph-cut* que l'on cherche à résoudre ici est de produire des *coupes* maximisant le poids des arêtes internes à chaque cluster. Une *coupe* (P, Q) (représentant deux clusters de nœuds disjoints) est caractérisée par une valeur de coupe : $coupe(P, Q) = \sum_{i \in P, j \in Q} a_{ij}$ pour un exemple de partitionnement en 2 clusters. La fonction à minimiser est la suivante : $J_n = \frac{coupe(P,Q)}{Vol(P)} + \frac{coupe(P,Q)}{Vol(Q)}$ avec Vol la fonction calculant le volume d'un cluster : $Vol(P) = \sum_{i \in P, j \in N} a_{ij}$ avec N représentant l'ensemble des nœuds du problème de graph-cut, i.e l'ensemble des classes.

L'utilisation de méthodes spectrales permet de résoudre ce problème de *Coupes*. Cependant, il faut noter que cette méthode ne garantit pas toujours un équilibre des clusters ce qui peut occasionner la création d'arbres déséquilibrés.

Finalement, l'utilisation d'une matrice de confusion pour aider à regrouper les classes qui sont difficilement séparables permet d'améliorer significativement la qualité des partitions de classes [Bengio et al., 2010]. Néanmoins, la méthode est tributaire de la qualité de

l'information de la matrice de confusion. Dans un problème avec un grand nombre de classes, il peut être facile de séparer n'importe quelle paire de classes. Ainsi, l'information apportés par la matrice de confusion peut ne pas être suffisante pour garantir la formation de bonnes partitions.

Partitionnement redondant. Dans la même idée, Deng et al. [2011] construit une hiérarchie top-down, mais à la différence du travail précédent, la hiérarchie de classes construite ici est *redondante* ce qui implique qu'une classe peut se retrouver dans plusieurs nœuds de même profondeur dans la hiérarchie.

Lors de l'apprentissage, le partitionnement est effectué conjointement à l'apprentissage des fonctions de décision.

Ce partitionnement est effectué en minimisant une fonction d'*ambiguïté*. L'ambiguïté A caractérise la diminution du nombre de classes à désambiguïser à travers le parcours d'arbre pendant la classification de l'exemple. C'est une mesure locale qui se calcule au niveau d'un nœud et de ses enfants directs :

$$A(w, x, P) = \frac{1}{Z} \sum_{i=1}^{Z} P(\hat{q}, i)$$

où w correspond aux paramètres de la fonction de décision, P la matrice de partitionnement, Z le nombre de classes du nœud, et $\hat{q}$ l'index du nœud enfant vers lequel l'exemple x descend. Plus l'ambiguïté est grande, moins la classification sera rapide car un trop petit nombre de classes ont été écartées. Cette mesure est très importante dans le cas d'une hiérarchie redondante car sans elle une solution simple serait de garder le maximum de classes dans les nœuds, ce qui augmenterait considérablement la hauteur de l'arbre. Ainsi, cette mesure permet d'avoir une idée de la *vitesse* de classification de l'arbre en contrôlant la redondance de l'arbre. Le problème d'optimisation de la partition est le suivant :

$$\begin{aligned}
\underset{w,P}{\text{minimiser}} \quad & \tfrac{1}{m} \textstyle\sum_{i=1}^{m} A(w, x_i, P) && (2.3)\\
\text{t.q.} \quad & \tfrac{1}{m} \textstyle\sum_{i=1}^{m} L(w, x_i, y_i, P) \leq \epsilon && (2.4)\\
& P \in \{0,1\}^{Q \times Z} && (2.5)\\
& \text{avec } Q \text{ le nombre d'enfants.} && (2.6)
\end{aligned}$$

La résolution de ce problème n'est pas possible en pratique car l'optimisation d'une matrice P sur les entiers rend le problème NP-hard. Une façon de précéder consiste à couper le problème d'optimisation en deux pour, d'un côté optimiser w avec P fixé, et de l'autre optimiser P avec w fixé. Pour ce second problème, il a été montré par Deng et al. [2011] qu'il est possible de trouver une solution optimale approchée en un temps polynomial en passant par une résolution dans l'espace des réels.

Un des soucis avec ces méthodes est le manque d'automatisation quant au choix du nombre de partitions à faire. Ces méthodes fixent au tout début du processus le nombre d'enfants par nœud, ce qui fige la structure de l'arbre en création. Une ontologie peut au contraire revêtir des formes diverses, et le fait de limiter cette dimension peut nuire à l'optimalité du partitionnement trouvé. Une façon serait de chercher à optimiser aussi le nombre de clusters par nœud [Zelnik-manor et al., 2004] ce qui permettrait une flexibilité supplémentaire pour mieux coller à la hiérarchie latente des classes du problème.

2.4.2.2 Agrégation *Bottom-up*

L'autre approche *gourmande* possible est d'attaquer la construction de l'arbre par le bas. Ce type d'approche, dite *bottom-up*, se concentre sur l'agrégation des classes les plus similaires, selon une mesure d'agrégation particulière. Les nœuds à agréger sont de plus en plus gros et le processus continue jusqu'à ce que tous les nœuds fassent partie du même arbre.

Zhigang et al. à proposé une construction de hiérarchie en utilisant une mesure de similarité entre classes basée sur la méthode *SVDD* pour *Support Vector Data Description*. Cette méthode cherche à englober les points correspondant à une classe donnée dans une hypersphère de volume minimal. Par l'apprentissage d'une projection de l'espace des exemples en utilisant un noyau *RBF*, le problème de l'hypersphère se résume à trouver un hyperplan qui maximise l'écart entre les points d'une classe donnée et l'origine. La séparabilité entre deux classes est ensuite calculée en fonction des hypersphères correspondant à ces deux classes. L'algorithme d'agrégation itératif construit ensuite la hiérarchie par le bas : les deux clusters les plus proches, selon la mesure de séparabilité, sont agrégés à chaque itération. Deux techniques différentes permettent de construire, soit un arbre binaire, soit un arbre n-aire. Xia et al. [2007] a continué cette approche d'agrégation *gourmande* de clusters. Il utilise une mesure de séparabilité prenant en compte simultanément

les centroïdes des classes et les variances associées aux ensembles de points appartenant à une même classe. La justification étant que deux centroïdes éloignés n'impliquent pas forcément une bonne séparabilité entre deux classes.

Le principal problème avec ce genre de méthodes vient de la qualité de la mesure de séparabilité pour des problèmes avec un très grand nombre de classes [Marszalek and Schmid, 2008]. En effet, dans ce contexte, les espaces sont souvent très grands et permettent ainsi facilement de séparer n'importe quel ensemble de points avec un simple hyperplan. De ce fait, la mesure n'est pas suffisamment indicatrice de réelles proximités entre classes ce qui limite la qualité des partitionnements ainsi que des hiérarchies basées sur ces mesures.

De plus, Langford a très récemment avancé des idées pour expliquer en quoi le partitionnement *bottom-up* était une tâche par nature très compliquée à réaliser correctement [Choromanska and Langford, 2014].

2.4.3 Apprentissage des fonctions de décision dans les nœuds de la hiérarchie

Nous venons de voir des méthodes permettant de créer une hiérarchie de classes dans le cas où il n'y aurait pas d'information de hiérarchie disponible, ou bien que cette information de hiérarchie ne serait pas utile pour la tâche de classification.

Supposons maintenant que nous avons une hiérarchie à notre disposition. Il est ensuite nécessaire d'apprendre les fonctions de décision dans chacun des nœuds de la hiérarchie, de façon à pouvoir guider les exemples à classer à travers l'arbre de classes.

2.4.3.1 Apprentissage indépendant des modèles

La façon la plus simple d'apprendre ces classifieurs est de les apprendre indépendamment les uns des autres [Liu et al., 2005a; Yang et al., 2003]. L'idée est de regarder le problème local de classification en fonction des labels présents dans chacun des nœuds fils. La méthode la plus usitée consiste à construire un classifieur par nœud fils en utilisant un schéma OAA.

Pour l'ensemble de la tâche, le but est de minimiser l'erreur de classification (*coût 0/1*) sur l'ensemble des exemples m à disposition :

$$R_{\mathcal{D}}(h) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}(h(x_i) \neq y_i)$$

En remarquant que, dès que l'exemple dévie et prend un chemin ne lui permettant plus d'arriver à une feuille portant la bonne classe, une erreur peut être comptabilisée par exemple et par chemin, on peut écrire la fonction d'erreur hiérarchique comme suivant :

$$R_{\mathcal{D}}(\mathcal{F}) = \frac{1}{m}\sum_{i=1}^{m} \max_{j \in B(x)} \mathbb{1}(y_i \notin s_j)$$

où $B(x) = \{b_1(x), b_2(x), \ldots\}$ correspond à l'ensemble des nœuds formant la trajectoire de l'exemple x dans l'arbre, et s_j correspond à l'ensemble des classes présentes au nœud n_j. Ainsi, $b_1(x)$ est l'index du nœud fils sous le nœud racine qui correspond au mieux à l'exemple x, $b_2(x)$ correspond à l'index du meilleur nœud au second niveau de l'arbre, etc. Ce calcul de l'erreur ne comptabilise qu'une seule erreur pour un exemple qui prendrait un mauvais chemin.

L'optimisation directe de $R_{\mathcal{D}}$ n'est pas possible à cause du max. On définit alors $G_{\mathcal{D}}(\mathcal{F})$ la fonction d'erreur approximative qui comptabilise les erreurs sur l'ensemble des nœuds :

$$G_{\mathcal{D}}(\mathcal{F}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{n \in N} \mathbb{1}(sgn(f_n(x_i)) = C_n(y_i))$$

avec $C_n(y) = +1$ si $y \in s_n$ et -1 dans le cas inverse.

On a bien évidemment $R_{\mathcal{D}}(\mathcal{F}) < G_{\mathcal{D}}(\mathcal{F})$ étant donné que l'un somme toutes les erreurs des nœuds alors que l'autre prend un maximum sur le chemin emprunté par l'exemple dans l'arbre.

La solution à l'optimisation de $G_{\mathcal{D}}$ peut être très différente de celle qui optimiserait le risque empirique $R_{\mathcal{D}}$.

Pour son problème d'optimisation, Bengio et al. [2010] propose de remplacer la fonction indicatrice $\mathbb{1}$ de $G_{\mathcal{D}}$ par un *hinge loss* de la forme $f_n(x) = w_n\phi(x)$. Ainsi, le problème

de minimisation pour apprendre l'ensemble des fonctions de décision $\mathcal{F}$ s'écrit :

$$\sum_{n \in N} \left(\gamma ||w_n||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_{i\,n} \right) \text{ tel que } \forall i, \forall n \begin{cases} C_n(y_i) f_n(x_i) \geq 1 - \xi_{i\,n} \\ \xi_{i\,n} \geq 0 \end{cases}$$

où les $\xi_{i\,n}$ agissent comme des marges et permettent de contrôler l'erreur de classification. Comme les paramètres des classifieurs locaux n'interfèrent pas avec la fonction objectif, les processus d'apprentissage des classifieurs peuvent être parallélisés. Cette propriété est très intéressante dans le cas du contexte de classification dans un grand nombre de catégories car l'apprentissage peut être plutôt long. Le fait de ne pas pouvoir paralléliser cet apprentissage peut être rédhibitoire dans le choix d'une méthode plutôt qu'une autre.

D'autres travaux ont cherché à améliorer les performances de classification en faisant de la réduction de dimension locale, de façon à limiter les erreurs de sur-apprentissage [Koller and Sahami, 1997; Dumais and Chen, 2000].

2.4.3.2 Apprentissage joint des modèles

Une dernière famille d'approches pour l'apprentissage de ces fonctions de décision correspond aux modèles apprenant ces fonctions de décision de façon globale. Ces méthodes sont fondamentalement plus complexes, étant donnée la quantité plus importante de paramètres qu'il faut apprendre en même temps.

Cependant, cet apprentissage global des classifieurs permet de répondre au problème du manque d'exemples pendant l'apprentissage des paramètres des classifieurs. En effet, vers le bas de la hiérarchie, les classifieurs sont appris sur un sous-ensemble très réduit de l'ensemble d'apprentissage. L'erreur d'estimation peut ainsi prendre des proportions non négligeables suivant la rareté des données d'apprentissage. Le fait d'apprendre globalement ces classifieurs permet d'aider le processus d'apprentissage en forçant un apprentissage conjoint des paramètres des classifieurs. De plus, cet apprentissage joint permet une meilleure approximation du *coût 0/1*. En effet, le problème d'optimisation opère sur une fonction d'erreur prenant en compte l'ensemble de l'arbre de classifieurs, ce qui permet d'optimiser une fonction d'erreur plus proche du *coût 0/1*.

Voilà le problème formalisé le plus simplement possible :

$$\arg\min_{w} \mathcal{R}(w) + C \times \mathcal{L}_{emp}(w)$$

où w correspond aux paramètres à apprendre, $\mathcal{R}$ est le terme de régularisation, $\mathcal{L}_{emp}$ correspond à la fonction d'erreur et C correspond à l'hyper-paramètre contrôlant le compromis entre les deux termes. Pour faire simple, dans le cas d'une optimisation indépendante, les termes $\mathcal{R}$ et $\mathcal{L}_{emp}$ peuvent être décomposés et l'apprentissage se fera localement à travers la hiérarchie de classifieurs. Dans le cas d'une optimisation jointe, il y a deux possibilités :

1. Le terme d'erreur empirique $\mathcal{L}_{emp}$ englobe une erreur sur tous les classifieurs en même temps.
2. Le terme de régularisation $\mathcal{R}$ inclut des contraintes croisées entre classifieurs.

Dans un cas comme dans l'autre, l'optimisation doit se faire sur l'ensemble des classifieurs. Nous détaillons ces deux aspects ci-dessous.

Erreur Empirique Jointe Les premiers travaux cherchant à apprendre une hiérarchie de façon jointe [Cai and Hofmann, 2004] ont cherché à généraliser les travaux sur la classification multi-classes SVM développée par Crammer and Singer [2002b]. Le problème d'optimisation cherchant à optimiser les paramètres de poids w_k pour chaque nœud s'écrit ainsi :

$$\text{minimiser } \gamma \sum_{n \in N} ||w_n||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \tag{2.7}$$

$$\text{tel que} \quad \forall j \neq i,\ F(x_i, y_i; w_{y_i}) - F(x_i, y_j; w_{y_j}) \geq 1 - \xi_i \tag{2.8}$$

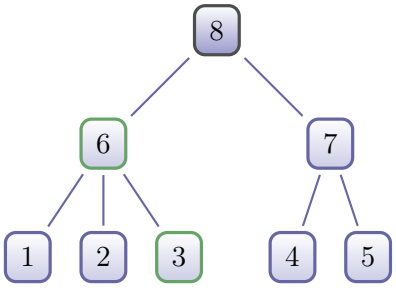
$$i \in \{1, \ldots, m\}, \xi_i \geq 0 \tag{2.9}$$

avec $F(x, y_i; w_i)$ une fonction de décision, qui s'écrit le plus souvent comme un simple produit scalaire. En associant les classes à un vecteur d'attributs $\Gamma(y) \in \mathbb{R}^s$, on peut écrire de façon plus générale :

$$F(x, y_i; w) = \langle w, \Phi(x, y) \rangle = \sum_{n \in N} \lambda(y) \langle w_n, x \rangle$$

$$\text{avec} \quad \Phi(x, y) = \Gamma(y) \otimes x = \begin{pmatrix} \lambda_1(y) \cdot x \\ \lambda_2(y) \cdot x \\ \dots \\ \lambda_n(y) \cdot x \end{pmatrix}$$

où la fonction λ contrôle la façon dont sont formés les vecteurs attributs correspondant aux classes. La Figure 2.4 montre comment les fonctions Γ et Φ s'utilisent. La fonction Γ prend un label en entrée et retourne le vecteur d'attributs correspondant à cette classe en utilisant l'information hiérarchique des classes.



$$\Gamma(3) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \text{ et } \Phi(x, 3) = \begin{pmatrix} 0 \\ 0 \\ x \\ 0 \\ 0 \\ x \\ 0 \end{pmatrix}$$

FIGURE 2.4: Exemple d'un problème de classification avec 5 classes. Une organisation hiérarchique des classes est donnée à gauche. Les vecteurs d'attributs $\Gamma(.)$ et $\Phi(.,.)$ sont de la même longueur que le nombre de nœuds moins un (le nœud racine ne compte pas). Un exemple est donné pour montrer comment le vecteur d'attributs du nœud '3' est construit : pour chaque nœud présent sur le chemin entre le nœud racine et le nœud feuille '3', la valeur du vecteur d'attributs correspondant est instanciée à 1. Pour tous les autres nœuds, la valeur est 0. Dans notre exemple, les lignes 3 et 6 ont une valeur de 1 étant donné la hiérarchie donnée à gauche.

Ainsi, l'information hiérarchique de classes est incluse dans la fonction d'erreur à optimiser. L'idée est que deux classes proches dans la hiérarchie vont avoir des vecteurs d'attributs similaires. Cette formulation du problème d'optimisation permet d'approximer de façon plus précise la fonction d'erreur hiérarchique vue précédemment plutôt que le *coût 1/0*. Ainsi, en remplaçant la contrainte de violation de la marge ξ_i (dans l'équation 2.8) par $\frac{\xi_i}{\gamma(i,j)}$, il a été montré par Cai and Hofmann [2004] que la solution à ce problème d'optimisation spécifique donne une bonne solution pour le problème avec une fonction d'erreur hiérarchique : $\hat{\gamma} \equiv \frac{1}{m} \sum_{i=1}^{m} \gamma(y_i, h(x_i))$.

Cette méthode d'optimisation a montré un meilleur contrôle de l'erreur d'estimation par l'introduction de contraintes prenant en compte l'organisation des classes en une hiérarchie. Ces contraintes permettent de forcer une optimisation jointe des paramètres de tous les nœuds présents sur le chemin pour rejoindre un nœud feuille. Il faut noter cependant que dans le cadre de problèmes de classification dans un très grand nombre

de classes, les vecteurs attributs peuvent atteindre des tailles conséquentes ce qui peut rendre le problème d'optimisation difficile à résoudre à cause du nombre de contraintes (la taille du vecteur d'attributs est égal au nombre de nœuds dans l'arbre). De plus, dans l'optique de minimiser le *zero-one loss*, il n'est pas nécessaire de satisfaire toutes les contraintes de l'équation 2.8.

Bengio et al. [2010] a cherché à organiser ces contraintes suivant le chemin emprunté par les exemples plutôt que de chercher à satisfaire un ensemble plus large de contraintes (comme en 2.8). Ces contraintes visent à contrôler que pour un nœud père à un étage donné de la hiérarchie, seul le bon nœud enfant ait un score plus élevé que le score des autres nœuds :

$$\text{minimiser} \qquad \gamma \sum_{j=1}^{n} ||w_j||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \tag{2.10}$$

$$\text{tel que} \qquad f_r(x_i) \geq f_s(x_i) - \xi_i, \forall r, \forall s : (\ell_i \in s_r \wedge \ell_i \notin s_s), \exists p : r, s \in \mathcal{E}(p) \tag{2.11}$$

$$\xi_i \geq 0, i = 1, \ldots, m \tag{2.12}$$

où r et s représentent des nœuds frères. En pratique, cela signifie qu'il n'y a au plus qu'une seule contrainte active étant donné que seuls les nœuds frères rentrent dans le problème d'optimisation des paramètres d'un nœud. La résolution de ce problème d'optimisation est ainsi facilité par rapport à la résolution du problème d'optimisation posé par Cai and Hofmann [2004].

Régularisation Jointe L'autre façon d'introduire des contraintes liées à une information d'organisation hiérarchique de classes est d'utiliser le terme de régularisation. L'idée est par exemple de se servir des classifieurs voisins pour contrôler leurs paramètres, de façon à les forcer à être plus similaires. Ce type d'approche vient du domaine du *transfer learning* [Argyriou et al., 2008; Caruana, 1997; Evgeniou and Pontil, 2004] qui vise à transférer ce que l'on a appris durant une tâche vers une autre tâche similaire. Dans le cadre de l'apprentissage de classifieurs organisés hiérarchiquement, l'idée est de forcer les classifieurs proches hiérarchiquement à ne pas être trop différents [Cai and Hofmann, 2004]. Mais à la différence des travaux présentés précédemment, c'est le terme de régularisation qui va être utilisé pour inclure ces nouvelles contraintes [Dekel et al., 2004; Gopal and Yang, 2012, 2013]. Par exemple, Gopal and Yang [2012] à cherché à modéliser le problème avec une approche Bayésienne. Le but étant de modéliser les dépendances

entre les nœuds en utilisant une régression logistique multi-variées. Le souci avec ce type d'approche peut venir du temps d'inférence qui peut limiter son utilisation dans le cas de très grandes bases de données (comportement propre à l'approche Bayésienne). Dans le prolongement de son travail précédent, Gopal and Yang [2013] à proposé une méthode récursive de régularisation capable de modéliser n'importe quel ensemble de dépendances issues d'un graphe :

$$\text{minimiser } \frac{1}{2}\sum_{n\in N}||w_n - w_{\mathcal{P}(n)}|| + \gamma\sum_{j\in\mathcal{L}}\sum_{i=1}^{m} L(y_{ij}, x_i, w_j)$$

avec $\mathcal{P}(n)$ le nœud parent du nœud n et $L(y, x, w)$ une fonction de coût dépendant des classifieurs locaux (SVM ou régression logistique). Dans ce problème d'optimisation, les paramètres peuvent être appris en partie en parallèle, étant donné qu'il y a uniquement des dépendances entre les paramètres d'un nœud père avec ses nœuds fils.

Une autre idée, qui va un peu à l'encontre de la première, consiste à forcer une certaine dissimilarité entre le classifieur d'un nœud père et les classifieurs des nœuds fils. Zhou and Xiao [2011] ont développé cette idée en forçant l'orthogonalité entre les hyperplans des nœuds père et fils. Cette idée est motivée par le besoin de spécificité quand on descend dans la hiérarchie pendant la classification d'un nouvel exemple.

Plus on descend dans la hiérarchie, plus les nœuds ont besoin de spécificité. L'idée de l'orthogonalité ajoute une dimension à ce raisonnement. Imaginons qu'un nœud père représente l'ensemble des mammifères et qu'il se soit aidé de la caractéristique *poil* pour discriminer les exemples appartenant à son ensemble de catégories. Ensuite, les enfants vont avoir à différencier les mammifères entre eux. La caractéristique *poil* n'est plus utile à ce moment là (étant donné que les mammifères sont poilus), et il est intéressant de forcer les classifieurs à se concentrer sur des caractéristiques qui vont être plus utiles dans la tâche visant à séparer les mammifères entre eux.

$$\begin{aligned}
\text{minimiser} \quad & \tfrac{1}{2}\textstyle\sum_{v\in N}\mathbf{K}_{vv}||w_v||^2 + \sum_{v\in N}\sum_{u\in\mathcal{B}(v)}\mathbf{K}_{uv}|w_u^T w_v| + \frac{C}{m}\sum_{i=1}^{m}\xi_i && (2.13)\\
\text{tel que} \quad & \forall i \in [\![1,\dots m]\!], \forall v \in \mathcal{B}^+(\ell_i), \forall u \in \mathcal{S}_v : \langle w_v, x_i\rangle - \langle w_u, x_i\rangle \geq 1 - \xi_i && (2.14)\\
& \forall i \in [\![1, m]\!] : \xi_i \geq 0 && (2.15)
\end{aligned}$$

où $\mathcal{B}(v)$ correspond à l'ensemble des nœuds du chemin menant du nœud racine au nœud feuille v , $\mathcal{B}^+(\ell)$ correspond à l'ensemble des nœuds du chemin menant du nœud racine

au nœud feuille associé à la classe ℓ , $\mathcal{S}_v$ correspond à l'ensemble des nœuds frères de v , $K \in \mathbb{R}^{K \times K}$. Dans ce problème d'optimisation, $\mathbf{K}_{uv}|w_u^T w_v|$ va forcer les classifieurs appartenant au même chemin [racine $\rightarrow$ feuille] à être orthogonaux.

2.4.3.3 Apprentissage itératif des modèles

Nous venons de voir comment les classifieurs des nœuds de l'arbre pouvaient être appris tous en même temps, en incorporant dans les termes de régularisation ou d'erreur des contraintes basées sur la hiérarchie de classes. Il est possible d'aborder différemment ce problème en apprenant les classifieurs de façon séquentielle.

Lors de l'apprentissage de classifieurs organisés hiérarchiquement dans le contexte de classification dans un grand nombre de classes, on observe un phénomène appelé propagation des erreurs. Il s'agit simplement du fait qu'un exemple suit généralement un chemin spécifique dans l'arbre et qu'il arrive fréquemment qu'un exemple dévie des chemins qui lui permettent d'atteindre la bonne classe. Une autre conséquence de ce phénomène correspond à la qualité des ensembles d'entraînement. Lors de l'apprentissage des classifieurs, un ensemble d'entraînement est utilisé. Cet ensemble est supposé être un sous-ensemble représentatif de l'ensemble des exemples qu'il va falloir classifier ensuite (sous-ensemble de test). Dans le cadre de classification hiérarchique, il peut arriver que l'ensemble de test dévie fortement de la distribution de l'ensemble d'entraînement, étant donné que chaque exemple de l'ensemble de test va suivre un chemin particulier. Beygelzimer et al. [2009] à cherché à corriger ce biais en proposant de filtrer l'ensemble d'apprentissage au fur et à mesure de l'apprentissage des classifieurs. En pratique, l'auteur propage les exemples du bas de la hiérarchie vers le haut. Seuls les exemples ayant été bien classifiés sont autorisés à être utilisés pour l'apprentissage des nœuds plus haut. De cette façon, l'erreur de propagation est contrôlée.

Cependant, cette méthode peut entraîner un autre problème : le manque de données d'entraînement. Vu que l'ensemble d'apprentissage est filtré, il peut être fortement réduit après plusieurs niveaux de hiérarchie. Bennett and Nguyen [2009] ont cherché aussi à améliorer la capacité de généralisation des classifieurs en sélectionnant l'ensemble d'entraînement à chaque nœud. Ils ont décidé de commencer à apprendre les classifieurs en haut de la hiérarchie et de propager uniquement vers les nœuds fils les exemples ayant été bien classifiés. Par l'utilisation de validation croisée, ces ensemble sont sélectionnés

précisément de façon à éviter de trop faire dévier les distributions des ensembles d'entraînement des ensembles de tests. Pour éviter de se retrouver avec le problème du manque de données, on ajoute systématiquement à l'ensemble d'entraînement les exemples correspondant aux classes présentes dans le nœud (ces exemples s'ajoutent donc aux exemples descendant du nœud père). Ceci permet de ne pas se retrouver avec trop peu d'exemples une fois arrivé à des étages inférieurs de la hiérarchie.

Une autre façon pour affiner l'ensemble d'apprentissage consiste à concaténer à l'exemple x les sorties des fonctions de décisions des classifieurs des nœuds enfants. Ainsi, la réponse des classifieurs des étages inférieurs est directement utilisée dans l'apprentissage des fonctions de décision des nœuds supérieurs. Ces méthodes peuvent être combinées pour donner un processus global appelé *refinement experts* [Bennett and Nguyen, 2009] qui se déroule en deux étapes : 1/ un processus de bas en haut visant à réduire les faux négatifs via l'utilisation des informations des classifieurs fils et 2/ un processus de haut en bas visant à affiner les ensembles d'apprentissage en faisant propager les exemples depuis le nœud racine. Il a été montré que ces méthodes permettent d'améliorer de façon significative la qualité des fonctions de décision apprises.

2.4.4 Apprentissage global

Nous venons de voir comment les deux briques *apprentissage de hiérarchie de classes* et *apprentissage des fonctions de décision* pouvaient être apprises indépendamment. Or, il apparaît que ces méthodes peuvent donner des résultats sous-optimaux étant donnée que l'un des deux paramètres est figé pendant l'apprentissage de l'autre. Certains travaux ont donc essayé de développer des processus d'apprentissage capables de travailler conjointement sur la création d'une hiérarchie et sur l'apprentissage des fonctions de décisions.

Nous avons déjà vu en détail la méthode de Deng et al. [2011] qui construit une hiérarchie de classes en même temps qu'il apprend les paramètres de ses classifieurs linéaires dans les nœuds (sous-section 2.4.2.1). En commençant par le nœud racine, l'algorithme d'apprentissage optimise deux problèmes itérativement : le premier problème consiste à trouver une partition optimale des classes, étant donnée une répartition des exemples dans les nœuds fils ; l'autre problème consiste à apprendre les poids des fonctions de décision linéaires étant donné un partitionnement des classes dans les nœuds fils.

De façon un peu similaire, Gao and Koller [2011b] propose d'apprendre une hiérarchie de classifieurs en résolvant un problème de coloration des nœuds. Ici aussi, l'apprentissage commence par le nœud racine et propose de construire un arbre binaire de classifieurs. Le problème d'optimisation propose d'apprendre en même temps les poids des classifieurs linéaires ainsi que la coloration des nœuds. Ce qu'on appelle coloration correspond au problème de partitionnement avec une différence notable. A chaque nœud, il est décidé de ne pas tenir compte de certaines classes difficiles à distinguer des autres. Ainsi, le problème de séparation induit est plus simple étant donné le nombre réduit de classes à prendre en compte. Toutes les classes mises de côté pour ce nœud seront présentes dans tous les nœuds fils. Le côté sensible dans cette optimisation est de réussir à trouver un compromis entre la vitesse de classification et la précision de la classification. Plus il y a de *non-décision* en laissant de côté un grand nombre de classes, plus l'arbre sera haut.

Dans une toute autre approche, Weston et al. [2013] construit une hiérarchie de deux niveaux uniquement dans le but de réduire le nombre de classifieurs à utiliser pour faire une prédiction. En effet, l'auteur a K *scorers* à sa disposition : un par classe. Son but est de limiter le nombre de *scorers* à utiliser pour classifier un nouvel exemple. Pour l'apprentissage, l'algorithme des K-moyennes est utilisé pour fractionner les exemples d'apprentissage en K clusters. Puis, une coloration redondante des nœuds est appliquée aux clusters. Pendant la classification d'un nouvel exemple, le cluster le plus proche est assigné à l'exemple. Finalement, uniquement les *scorers* des classes qui ont été assignées au cluster choisi sont utilisés.

2.5 Méthodes d'*Embedding*

Les méthodes dites d'embedding sont des méthodes qui cherche à créer une projection de l'espace des données initiales vers un espace latent avec des caractéristiques particulières. Ces méthodes ne permettent pas en elles même une accélération du temps d'inférence computationnelle au niveau de l'algorithme de décision. L'embedding peut servir à réduire la dimension des exemples ce qui permet d'accélérer les opérations faites sur les données.

Une autre utilisation possible est de se servir de l'espace latent pour faire la prédiction. Dans cet exemple la, la classification d'un exemple se fait en choisissant la classe correspondant au prototype le plus proche de l'exemple dans l'espace latent. La complexité de ce type de prédiction se fait donc en $O(K)$ ce qui implique une dépendance linéaire vis-à-vis du nombre de classes de la complexité de prédiction. Ces méthodes ne permettent pas de réduire algorithmiquement le processus de classification mais il est néanmoins important de les passer en revue.

La méthode d'embedding la plus directe dans le cadre de classifications hiérarchiques consiste à chercher une projection telle que $\mathcal{Z}(y) = \mathbf{V}\phi(y)$ avec $\phi(y)$ le vecteur de dimensions K dont toutes les composantes sont nulles sauf la $y^{\text{ème}}$, et $\mathbf{V} \in \mathbb{R}^{\mathcal{D}_{lat} \times K}$ ($\mathcal{D}_{lat}$ la dimension de l'espace latent). La classification est donnée ensuite par :

$$h(x) = \underset{k \in [\![1,K]\!]}{\arg\max}\, S(\mathbf{W}x, \mathbf{V}\phi(k)))$$

où $\mathbf{W} \in \mathbb{R}^{\mathcal{D}_{lat} \times \mathcal{D}}$ est une matrice de paramètre à apprendre et $S(\cdot, \cdot)$ est une mesure de similarité entre deux vecteurs. Les méthodes de *compressed sensing* arrivent à des problèmes d'optimisation à peu près similaires. La matrice $\mathbf{V}$ est alors générée aléatoirement et seule la matrice $\mathbf{W}$ est apprise. Bengio et al. [2010] a proposé une méthode pour apprendre $\mathbf{V}$ et $\mathbf{W}$ de sorte que les exemples soient proches de leur prototypes dans l'espace latent. L'idée est d'optimiser le problème suivant :

$$\text{minimiser} \qquad \gamma||\mathbf{W}||_{FRO} + \tfrac{1}{m}\textstyle\sum_{i=1}^{m} \xi_i \tag{2.16}$$

$$\text{tel que} \qquad (\mathbf{W}x_i)^T \mathbf{V}\phi(i) \geq (\mathbf{W}x_i)^T \mathbf{V}\phi(j) - \xi_i, \forall j \neq i \tag{2.17}$$

Ce problème non-convexe peut être résolu approximativement par une descente de gradient stochastique en tirant au hasard les poids initiaux.

Dans le cas où une hiérarchie de classes est disponible, il est possible de chercher un espace commun aux classes et aux exemples en utilisant la méthode de Weinberger and Chapelle [2008]. L'idée est d'utiliser une matrice de coût entre les classes $\mathbf{C}$ contenant les informations d'organisation hiérarchique des classes. Ensuite, il faut apprendre les paramètres de l'espace latent de façon à ce que les distances entre les prototypes des classes reflètent les distances de la matrice de coût $\mathbf{C}$.

Chapitre 3

Caractérisation des Schémas de Classification

3.1 Introduction

Ce chapitre a pour but de présenter et d'analyser un nouveau formalisme qui permet de regrouper les méta-algorithmes présentés au chapitre précédent et qui vont servir de références tout au long de ce travail de thèse. Ce formalisme ainsi que le travail préliminaire d'étude des modèles classiques permettront de positionner clairement les contributions qui seront présentées dans les chapitres suivants.

Le travail de thèse qui a été conduit ici porte principalement sur les méta-algorithmes cherchant à agréger les réponses d'une multitude de classifieurs locaux simples de façon à construire une réponse globale au problème de classification. Ces méthodes permettent d'étudier et de contrôler le compromis entre la complexité et la précision d'une classification. En effet, en supposant que les temps de calcul des classifieurs simples sont tous du même ordre de grandeur, la complexité des algorithmes de prédiction peut se calculer en comptabilisant le nombre de classifieurs simples utilisés pour la prédiction globale du modèle pour un exemple. Ainsi, le compromis complexité-précision peut être facilement étudié.

Nous allons dans la suite de ce chapitre présenté le formalisme que nous proposons, appelé *schéma de classification*, qui permettra de situer précisément les différentes méthodes qui nous intéressent ici. Ce formalisme modélise la classification multi-classes

comme un processus de décision séquentiel. Nous étudierons ensuite plus en détail les principales méthodes relevant de ce formalisme. L'étude de leur fonctionnement ainsi que de leurs limites permettra de mieux situer les modèles présentés dans les chapitres de contributions de ce manuscrit de thèse.

3.2 Schéma de classification

Le formalisme des schémas de classification que nous proposons permet d'étudier dans un cadre unifié les méthodes qui utilisent un ensemble de classifieurs simples binaire dans le but de produire une classification multi-classe. C'est le cas en particulier des heuristiques simples du type *un contre tous*, des méthodes ensemblistes de type ECOC, qui utilisent un ensemble de réponses de classifieurs pour obtenir une prédiction globale, des méthodes hiérarchiques qui utilisent un ensemble de classifieurs organisés dans une hiérarchie.

Notre formalisme se focalise sur la question de l'inférence et plus exactement sur la manière de sélectionner les classifieurs simples et d'agréger leur réponse. Pour cela, nous proposons une modélisation sous la forme d'un processus séquentiel de décision, que nous appelons *schéma de classification*. Les classifieurs simples sont en général des modèles de classification binaire spécialisés dans la séparation de deux sous-ensembles d'exemples. Le schéma de classification détaille les processus d'apprentissage et d'utilisation des classifieurs pour la classification d'un nouvel exemple : quel est l'objectif de chaque classifieur, combien faut-il en apprendre, et comment faut-il les utiliser pour agréger les réponses et obtenir une classification.

3.2.1 Formalisme du schéma de classification

Un schéma de classification $\mathcal{S}$ est défini par 1/ la façon dont les classifieurs simples h sont constitués, c'est à dire quels sous-ensembles de classes chacun d'eux séparent, et 2/ comment ces classifieurs sont utilisés et comment leur réponses sont agrégées pour aboutir à une classification. Il est défini ainsi par un couple de 2 processus ($\mathcal{S} = \{f_{\mathcal{S}}, b_{\mathcal{S}}\}$) que nous allons détailler par la suite :

- $f_{\mathcal{S}}$, le processus de création des dichotomies et des classifieurs, spécifie :

  - le sous-ensemble des dichotomies qui seront utilisées,
  - comment sont appris ces classifieurs ;
- $b_{\mathcal{S}}$: le processus de classification d'un exemple, modélisé comme un processus de décision séquentiel ; il est constitué de
  - $\mathcal{M}$, un ensemble d'état qui représenteront dans notre contexte la mémoire du processus, le plus souvent un vecteur de $\mathbb{R}^K$ représentant le score de chaque classe ;
  - π, une politique de sélection du prochain classifieur simple à utiliser en fonction de l'état - mémoire actuel ;
  - $u_{\mathcal{S}}$, le mécanisme de mise à jour de la mémoire ;
  - $p_{\mathcal{S}}$, le mécanisme de classification finale en fonction de la mémoire.

Le processus $f_{\mathcal{S}}$ sera utilisé pendant la phase d'apprentissage du modèle pour constituer l'ensemble de classifieurs disponibles pour l'inférence. Pendant l'inférence d'un exemple x, la politique π spécifie le prochain classifieur à utiliser en fonction du résultat des classifications précédentes. L'évaluation de l'exemple x par le classifieur sélectionné induit un changement d'état/mémoire $\mathcal{M}^t$ à l'itération t commandé par le mécanisme $u_{\mathcal{S}}$. Finalement, la classification finale est obtenue par $p_{\mathcal{S}}$ qui va utiliser la mémoire $\mathcal{M}^{T_{max}}$ de l'état final T_{max} pour attribuer une classe à l'exemple x.

3.2.1.1 Création de l'ensemble de classifieurs simples

Notons Φ l'ensemble des dichotomies possibles $(\mathcal{C}^+, \mathcal{C}^-)$ de l'espace $\mathcal{L}$ tel que $\mathcal{C}^+ \cap \mathcal{C}^- = \emptyset$, $\mathcal{C}^+ \cup \mathcal{C}^- \subseteq \mathcal{L}$, $|\mathcal{C}^+| > 0$ et $|\mathcal{C}^-| > 0$. Un élément $\mathbf{c} \in \Phi$ définit ainsi deux sous-ensembles de classes. $\mathbf{c}$ peut être représenté par un vecteur dont chaque élément c_i est relatif à une classe ℓ_i. Chaque c_i peut se voir assigné 3 valeurs différentes : $\{-1, 0, 1\}$ suivant que la classe appartienne au premier groupe de classes $\{+\}$, au second groupe de classes $\{-\}$, ou a aucun des deux groupes. Un schéma de classification possède un processus $f_{\mathcal{S}}$ chargé de la création de cet ensemble :

$$f_{\mathcal{S}} : (\mathcal{D}, \Phi) \mapsto \Phi_{\mathcal{S}}$$

La création de cet ensemble permet d'entraîner un ensemble de classifieurs. Pour chaque dichotomie $\mathbf{c}$, il est possible d'apprendre un classifieur binaire qui va apprendre à séparer les exemples des classes de l'ensemble $\{+\}$ des exemples des classes de l'ensemble $\{-\}$.

Le choix du type de classifieur local est libre. Avec un contexte comme le notre où le nombre de classes peut être très large, les dichotomies peuvent correspondre à des problèmes de séparation de deux ensembles de classes difficilement séparables. Il est de coutume alors d'utiliser des modèles de classification puissants comme les machines à vecteurs de supports (ou SVM) pour chacun des classifieurs.

Pour illustrer cette création de classifieur : considérons le schéma de classification OAA. Ce schéma va sélectionner un ensemble de dichotomies Φ_{OAA} où chacune de ces dichotomies correspond à la séparation d'une classe de toutes les autres classes. Ainsi : $\Phi_{OAA} = \{(\{\ell_k\}, \{\mathcal{L} \backslash \ell_k\})\}_{k \in \{1,\ldots,K\}}$.

La création de cet ensemble de classifieurs correspond à la partie *apprentissage* des schémas de classification (bien que dans certains cas comme celui du OAA, les partitions sont fixées). La phase de classification qui va être détaillée dans la partie suivante correspond à l'utilisation de l'ensemble de classifieurs appris.

3.2.1.2 Processus de classification

Le processus de classification décrit le mécanisme pour la prédiction de la classe d'un nouvel exemple que l'on souhaite classifier. Le processus de classification est modélisé comme un processus de décision séquentiel défini par le quadruplet $(\mathcal{M}_\mathcal{S}, \pi_\mathcal{S}, u_\mathcal{S}, p_\mathcal{S})$:

- un ensemble d'états mémoire $\mathcal{M}_\mathcal{S}$, en général un vecteur de $\mathbb{R}^K$ de taille le nombre de classes et dénotant le score de chaque classe ; nous noteront $\mathcal{M}^t$ l'état de la mémoire à chaque itération t du processus.
- La politique $\pi_\mathcal{S} : \mathcal{M}_\mathcal{S} \to \Phi_\mathcal{S}$ sélectionne une dichotomie parmi celles possibles.
- Le mécanisme de transition $u_\mathcal{S} : \mathcal{M}_\mathcal{S} \times \Phi_\mathcal{S} \times \mathbb{R} \to \mathcal{M}_\mathcal{S}$ qui à partir d'un état mémoire, d'une dichotomie $\mathbf{c}$ et du résultat du classifieur $h_\mathbf{c}$ correspondant met à jour la mémoire.
- le mécanisme de décision $p_\mathcal{S} : \mathcal{M}_\mathcal{S} \to \mathcal{L}$ qui à partir de l'état mémoire final donne la classification de l'exemple.

Le schéma de classification $\mathcal{S}$ est doté d'une politique qui à partir d'un état mémoire sélectionne la prochaine dichotomie à utiliser. Cette dichotomie $\mathbf{c}$ et son classifieur associé $h_\mathbf{c}$ sont utilisés pour mettre à jour la mémoire selon $u_\mathcal{S}$ en fonction de la réponse $h_\mathbf{c}(x)$ du classifieur sur x (typiquement mise à jour des scores des classes). A la fin du processus

de classification, la processus retourne le label prédit en fonction de la mémoire à l'état final T_{max} et du mécanisme de décision $p_{\mathcal{S}}$ (la classe avec le plus grand score en général).

Bien entendu, bien que le processus de classification soit modélisé par un processus séquentiel dans notre formalisme, la tâche peut être parallélisée dans le cas de certaines méthodes.

3.2.2 Complexité du processus de classification

Nous allons définir ici une mesure qui nous intéresse particulièrement dans ce travail : la complexité du processus de classification (ou d'*inférence*). Pour plus de clarté, le terme *inférence* décrira uniquement l'action de classifier un exemple.

Cette mesure se base sur le nombre de classifieurs utilisés pour obtenir une prédiction pour un nouvel exemple. Nous considérons un cadre où tous les classifieurs simples issus d'une même famille de classifieurs $\mathcal{H}$ sont considérés comme coûtant le même temps (à un ordre de grandeur près) lors de l'inférence d'un nouvel exemple. Par exemple, si on considère la famille des classifieurs linéaires, le processus de décision de chacun des classifieurs est en $O(D)$ (avec D la dimension de l'espace des données).

Dans toute la suite de ce travail, on ne s'attardera pas sur les différentes familles de classifieurs possibles et on se limitera à l'utilisation de classifieurs linéaires.

Dans notre contexte , le coût de calcul d'une classification d'un nouvel exemple d'un schéma de classification est proportionnel au nombre de classifieurs utilisés lors du processus de classification ainsi qu'au coût du processus de décision séquentiel. Le plus souvent, ce coût est négligeable par rapport au coût d'utilisation d'un classifieur (par exemple, dans le cas d'un schéma hiérarchique de classification, le prochain classifieur est donné par la hiérarchie des classifieurs et son calcul est constant : $O(1)$) On peut ainsi réduire le problème de coût d'une classification au problème de complexité du processus de classification en nombre de classifieurs simples utilisés. Un schéma de classification OAA a une complexité de classification d'un nouvel exemple linéaire selon le nombre de classes du problème : $O(K)$. Un arbre binaire équilibré a une complexité de classification logarithmique selon le nombre de classes : $O(\log(K))$.

Lorsque l'arbre n'est pas équilibré, le mesure de la complexité doit être plus précise pour prendre en compte que certaines branches de l'arbre seront plus utilisées que d'autres. La

complexité d'un arbre non-équilibré est calculée en sommant le *coût* de classification de chaque nœud de l'arbre (le *coût* de classification d'un nœud est obtenu en comptabilisant le nombre de fils du nœud), pondérés par la probabilité d'utilisation des nœuds noté w_n. En effet, le nœud racine sera utilisé 100% du temps alors qu'un nœud juste au dessus de deux feuilles sera beaucoup moins utilisé. Cette probabilité est estimée en calculant le ratio entre le nombre d'exemples (dans l'ensemble d'apprentissage) des classes présentes dans le nœud avec le nombre total d'exemples. Ainsi, le calcul de la complexité d'un arbre non-équilibré peut s'écrire comme suivant :

$$Complexity(T) = \sum_{n \in \mathcal{A}} w_n |\mathcal{C}_n|$$

avec T un arbre, w_n la probabilité d'utilisation du nœud n, $|\mathcal{C}_n|$ le nombre de fils du nœud n.

On définit un *ratio de complexité* r_{x} qui calcule pour un schéma de classification $\mathcal{S}$ le ratio entre la complexité de classification du modèle en question et la complexité de classification du schéma de référence OAA. Le ratio de complexité s'écrit :

$$r_{\mathrm{x}}(\mathcal{S}) = \frac{Complexity(\mathcal{S})}{K}$$

où K (le nombre de classes) correspond à la complexité du modèle de référence OAA. Ainsi, un modèle avec un ratio r_{x} plus grand que 1 sera plus lent pour classifier un exemple que notre schéma de référence (OAA) alors qu'un modèle avec un ratio r_{x} plus petit que 1 sera plus rapide pour la tâche de classification d'un nouvel exemple.

Cette mesure servira essentiellement à comparer les différents modèles en mettant en regard leur précision de classification avec la complexité de leur processus de classification.

3.3 Étude des schémas de classification

Nous allons comparer et étudier les modèles de classification par agrégation de classifieurs dans notre formalisme. Nous proposons dans les chapitres suivants deux schémas de classification novateurs. Pour bien contextualiser nos travaux nous allons commencer

par proposer une étude approfondie des schémas de classification classiques pour mieux rendre compte de leur force ainsi que de leurs limites. Pour cela, nous avons conduit des expériences sur des ensembles de données réelles issues de récents challenges de classification en grand nombre de classes. Une étude plus détaillée des schémas de classification classiques basée sur le résultat d'expérimentations suivra.

3.3.1 Étude des schémas de classification classiques

Commençons par situer les méthodes classiques de classification dans ce nouveau formalisme.

La grande différence entre les schémas de classification ensemblistes et les schémas de classification hiérarchiques réside dans la manière dont les classifieurs sont appris et utilisés. Pour connecter ces deux types de méthodes avec le formalisme décrit précédemment, l'ensemble des schémas de classification ensemblistes peut être caractérisé par l'absence de politique spécifique $\pi_{\mathcal{S}}$ décrivant le choix du prochain classifieur (ou dichotomie) à utiliser. En effet, ces schémas utilisent une même séquence de dichotomies en parcourant une à une toutes les dichotomies disponibles. Cela signifie qu'il n'y a pas de choix *actif* des classifieurs pendant le processus de classification. Les schémas de classification hiérarchiques sont caractérisés par une politique spécifique à l'organisation hiérarchique des classifieurs qui ont été constitués préalablement. Le processus de décision du prochain classifieur à utiliser est dicté par le dernier classifieur utilisé et par la réponse qu'il a donnée.

Les schémas de classification hiérarchiques sont de plus caractérisés par une fonction de prédiction finale prenant uniquement en compte le dernier classifieur utilisé pour prendre sa décision. En effet, ce dernier classifieur amène l'état courant du processus de classification à un nœud feuille associé à une seule classe (dans le cas où il n'y a pas de redondance).

Nous avons résumé dans le Tableau 3.3.2 les méthodes principales qui rentrent dans notre formalisme de Schémas de Classification. Ce tableau est un condensé de tout ce qui se fait relativement à notre tâche de réduction du nombre de classifieurs utilisés pour une classification. Nous avons divisé les différents schémas de classification en trois familles : *classique*, *ensemble* et *hiérarchique*. Pour chacune des méthodes présentées

dans ce tableau, nous avons détaillé la famille de classifieurs utilisés ($f_{\mathcal{S}}$), la politique de sélection du prochain classifieur à utiliser ($\pi_{\mathcal{S}}$), la mémoire du système ($\mathcal{M}_{\mathcal{S}}$), la fonction de mise à jour de la mémoire en fonction de la réponse du classifieur utilisé ($u_{\mathcal{S}}$) et finalement la fonction de prédiction finale qui prédit la classe en fonction de la mémoire du système ($p_{\mathcal{S}}$).

Les méthodes de la famille classique sont les méthodes de l'état de l'art utilisées pour résoudre des problèmes de classification multi-classe. On retrouve ainsi les méthodes OAA et OAO. Ces méthodes n'ont pas de politique active et parcourent l'ensemble des classifieurs correspondant à $\Phi_{\mathcal{S}}$. Seules les méthodes hiérarchiques vont bénéficier d'une politique active quant au choix du prochain classifieur et il y aura une séquence de classifieurs spécifique à l'exemple en cours de classification qui sera utilisé.

La Figure 3.1 correspond au diagramme décrivant le processus de classification des schémas de classification de type méthode ensembliste. La Figure 3.2 correspond à celui des schémas de classification de type hiérarchique. On peut voir que les deux familles de schémas diffèrent en quelques points importants : les méthodes ensemblistes attendent d'avoir récolté toute l'information des classifieurs avant de prendre des décisions sur la classe à prédire. Pour les schémas hiérarchiques, la décision se fait tout au long du processus de classification. En effet, chaque mise à jour du nœud actif réduit l'ensemble des classes qu'il va être possible d'attribuer pour l'exemple en cours de classification étant donné que toutes les classes ne sont pas accessibles depuis n'importe quel nœud de l'arbre.

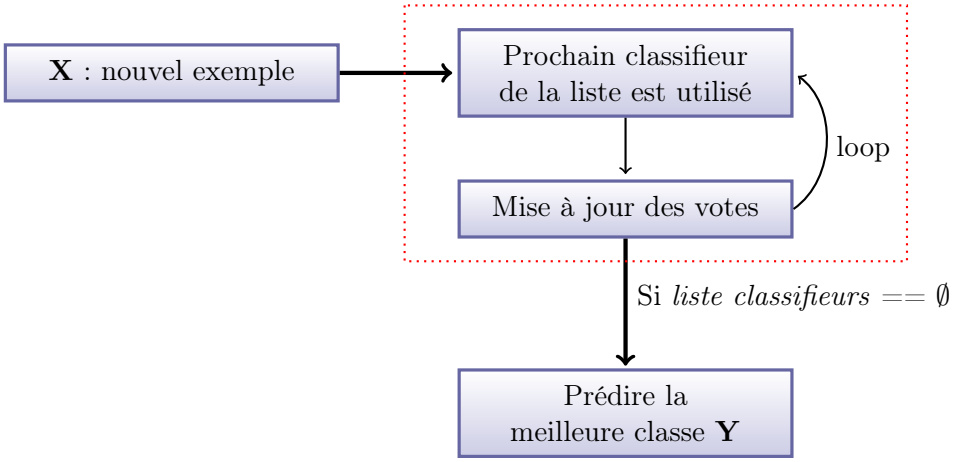


FIGURE 3.1: Processus de classification d'une méthode ensembliste.

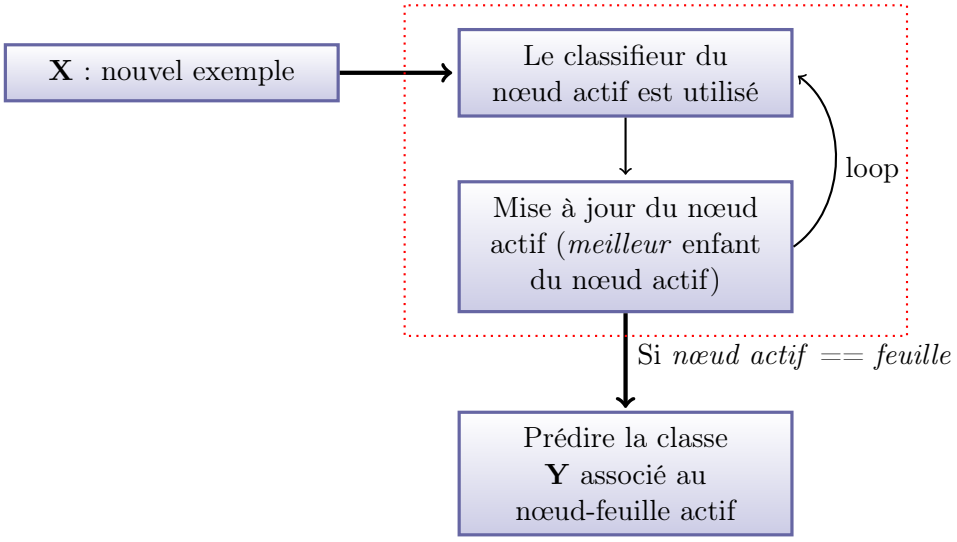


FIGURE 3.2: Processus de classification d'une méthode ensembliste.

	$f_{\mathcal{S}}$	$\pi_{\mathcal{S}}$	$\mathcal{M}_{\mathcal{S}}$	$u_{\mathcal{S}}$	$p_{\mathcal{S}}$
Classic :					
OAA	$\{(\ell_k, \{\mathcal{L}\backslash\ell_k\})\}_{k\in[\![1,K]\!]}$	*prochain* **c** *inutilis*	$\mathbb{R}^K$	$\mathbf{v}[k] = h^*_{\Phi_k}(x)$	$\arg\max_k \mathbf{v}[k]$
OAO	$\{(\ell_k, \ell_{k'})\}_{\{(k,k')\in\mathcal{L}^2 \text{ et } k\neq k'\}}$	*prochain* **c** *inutilis*	$\mathbb{N}^K$	$\rightarrow$ ajouter un vote aux deux classes	$\arg\max_k \mathbf{v}[k]$
Ensemble :					
ECOC avec *hamming decoding*	dichotomies aléatoires $\Phi_i \in \{-1,1\}^T$	*prochain* **c** *inutilis*	$\{-1,1\}^T$	$\mathbf{v}[t] = \begin{cases} -1 \text{ si } h^*_{\Phi_t}(x) < 0 \\ +1 \text{ si } h^*_{\Phi_t}(x) > 0 \end{cases}$	$\arg\max_k \#\{i : \mathbf{v}[i] = \Phi_i[k]\}$
ECOC avec coding ternaire	dichotomies aléatoires $\Phi_i \in \{-1,0,1\}^T$	*prochain* **c** *inutilis*	$\{-1,1\}^T$	$\mathbf{v}[t] = \begin{cases} -1 \text{ si } h^*_{\Phi_t}(x) < 0 \\ +1 \text{ si } h^*_{\Phi_t}(x) > 0 \end{cases}$	$\arg\max_k \#\{i : \mathbf{v}[i] = \mathbf{c}_i[k]\}$
ECOC with loss decoding	dichotomies aléatoires $\Phi_i \in \{-1,0,1\}^T$	*prochain* **c** *inutilis*	$\mathbb{R}^T$	$\mathbf{v}[t] = h^*_{\Phi_t}(x)$	$\arg\max_k \sum_i \mathbf{v}[i]\mathbf{c}_i[k]$
ECOC with Bloom Filters	dichotomies avec rejet par utilisation d'espaces latents	*prochain* **c** *inutilis*	$\{-1,1\}^T$	$\mathbf{v}[t] = \begin{cases} -1 \text{ si } h^*_{\Phi_t}(x) < 0 \\ +1 \text{ si } h^*_{\Phi_t}(x) > 0 \end{cases}$	$\arg\max_k \#\{i : \mathbf{v}[i] = \mathbf{c}_i[k]\}$
Hiérarchique :					
Clustering Spectral (*top-down*)	dichotomies avec rejet par clustering spectral	meilleur enfant	noeud courant	$\emptyset$	classe du noeud feuille atteint
Agregation (*bottom-up*)	Ternary dichotomies from class tree obtained	meilleur enfant	n oeud courant	$\emptyset$	classe du noeud feuille atteint

3.3.2 Expérimentations

Pour les expériences, nous avons utilisé les données qui ont été décrites en détail sous-section 2.2.2.2. En particulier, nous avons principalement utilisé la base de données DMOZ ainsi que les échantillons issus de la base de données Wikipédia. Nous nous sommes intéressés dans ce travail au compromis entre la précision et le temps alloué au calcul d'une prédiction. Nous avons précédemment défini pour cela un ratio de complexité qui correspond au nombre de classifieurs utilisés pour la prédiction d'un nouvel exemple. Nous avons fait varier la complexité des modèles testés et nous avons noté pour chacune de ces complexités la précision du modèle. Par exemple, pour les modèles de type ECOC, il suffit de changer le longueur des codes des classes pour changer la complexité.

La qualité de la classification est calculée par le taux de bonne classification sur un ensemble de test de n exemples :

$$R_{\mathcal{D}}(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(h(x_i) \neq y_i)$$

Il existe d'autres mesures permettant de rendre compte de la qualité de prédiction d'un modèle de classification multi-classe. Par exemple, il existe des mesures calculant un score relatif à l'éloignement de la catégorie prédite avec la catégorie réelle dans une hiérarchie de classe donnée [Silla and Freitas, 2011]. Nous avons fait le choix ici de rester au plus près de la tâche de classification en comptabilisant en erreur toute mauvaise classification.

Pour un modèle hiérarchique, il peut y avoir différentes façons de gérer la complexité. La plus simple est de jouer sur le nombre d'enfants par nœuds. En effet, plus le nombre d'enfants par nœud est grand, plus les classifieurs utilisés dans les nœuds sont complexes, et donc coûteux. Un nœud qui possède Q enfants ($Q \in [\![2, \infty]\!]$) doit apprendre Q classifieurs : un pour chaque ensemble de classe associé à chacun des enfants. Le passage d'un exemple d'un étage à l'étage inférieur coûte donc Q classifications. La profondeur d'un arbre Q-aire est approximativement égal à $\log_Q(K)$ pour un nombre total de classes de K et pour un arbre équilibré. Ainsi, le coût de classification C_{x} d'un tel arbre est égal à :

$$C_{\mathrm{x}} = Q \log_Q(K)$$

On obtient trivialement que plus Q est grand, plus la classification est coûteuse. De plus, on voit que le coût minimal de classification est obtenu pour $Q = 2$.

On remarque aussi que pour des valeurs de Q qui tendent vers K, on observe $C_{\mathrm{x}} \to K$. Ceci correspond au cas où l'arbre est complètement plat (i.e un schéma de classification OAA).

3.3.3 Étude de l'erreur

Étudions tout d'abord les performances des méthodes usuelles. Pour cela, nous avons implémenté les méthodes classiques entrant dans notre formalisme des schémas de classifications. Nous avons ainsi comparé des modèles hiérarchiques ainsi qu'un modèle ensembliste de type ECOC. Parmi les méthodes hiérarchiques, la première méthode utilise une hiérarchie originale pré-existante avec la base de donnée.

Chaque feuille est associé à une classe. La seconde méthode hiérarchique va créer sa propre hiérarchie de classes. Pour cela, on partitionne les ensembles de classes, en commençant par le nœud racine, via une méthode de clustering spectral. Il faut choisir le nombre d'enfants par nœud. Comme vu précédemment, ce paramètre permet de régler le compromis entre la complexité et la précision de la classification. Il est choisi ici de façon à obtenir une vitesse de classification similaire à la vitesse de classification du modèle hiérarchique utilisant la hiérarchie originale. Finalement, la méthode ensembliste de type ECOC utilise une matrice de coding ternaire et une méthode de decoding [1] utilisant les fonctions de décisions des classifieurs simples plutôt que juste la classification binaire (voir sous-section 2.3.3). Le tableau 3.1 montre ainsi les performances de ces trois approches pour des temps de classifications similaires. Le premier constat que l'on peut faire est que ces méthodes ne permettent pas d'obtenir des performances similaires à ceux d'une méthode utilisant l'information de hiérarchie de classes donnée avec les données DMOZ.

La différence de performance peut s'expliquer par des différences de séparabilité des dichotomies de classes mises en jeu.

1. Le decoding correspond au processus de classification d'un exemple grâce au "code" obtenu pour cet exemple constitué par les réponses des classifieurs utilisés.

TABLE 3.1: Score de modèles classiques sur le dataset DMOZ pour une même complexité de classification.

	% de bonne classification
Modèle Hiérarchique Originale	38.2
Modèle Hiérarchique 'Clustering Spectral'	32.7
Modèle Hiérarchique Aléatoire	28.3
Modèle ECOC 'Ternaire'	32.3

Attardons-nous donc un peu sur l'étude de l'erreur des classifieurs simples. Il y a différents facteurs qui peuvent amener un classifieur simple à faire une erreur de classification. Tout d'abord, il y a les erreurs que l'on caractérise comme étant des erreurs d'estimation. Cela correspond aux erreurs qui sont dues à une mauvaise généralisation. Ces erreurs peuvent être dues à un manque d'exemples d'entraînement, ou bien à un bruit présent dans les caractéristiques des données.

Ensuite, l'erreur peut être une erreur d'approximation qui correspond au fait que la famille de classifieurs du classifieur simple est trop restreinte. Il n'est pas possible de modéliser n'importe quel problème de classification binaire avec un simple hyperplan par exemple. Si le problème de classification est trop complexe à résoudre par rapport à la famille de classifieurs, cela générera des erreurs d'approximation.

Commençons par étudier ces différents types d'erreurs pour les schémas de classification hiérarchique. Nous avons affiché les % de bonne classification à différents niveaux de la hiérarchie pour un modèle hiérarchique utilisant la hiérarchie originale de classes et pour un modèle hiérarchique générant aléatoirement une hiérarchie de classes. Les résultats de ces expériences peuvent être visualisés Figure 3.3.

Ce qui diffère entre les deux modèles est la difficulté des problèmes de séparation que les classifieurs simples doivent résoudre. Dans le cas de la hiérarchie originale, les ensembles de classes dans chacun des nœuds sont plus homogènes que dans les hiérarchies aléatoires. Ainsi, les problèmes de séparation sont très probablement plus faciles. Si on regarde en particulier le nœud racine (qui correspond au *1er* niveau des hiérarchies), il y a un grand écart entre la précision obtenue par les deux classifieurs. Pourtant, c'est le même ensemble d'exemples qui est utilisé dans les deux cas, ainsi que le même simple modèle de classification (SVM linéaire). Cet écart correspond vraisemblablement à un très grand nombre d'erreurs d'approximation qui peut être expliqué par la difficulté du classifieur simple à bien séparer des ensembles difficiles à séparer. En effet, dans les niveaux élevés de

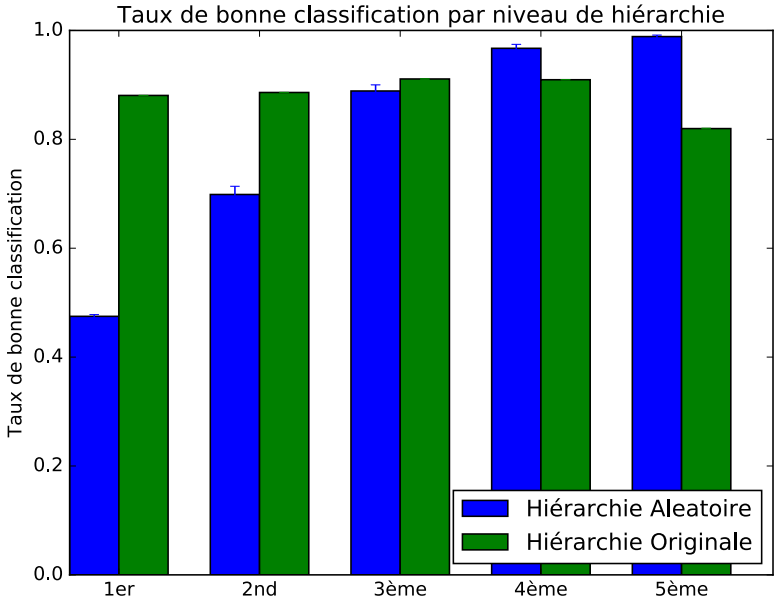


FIGURE 3.3: Score de précision pour chaque niveau de hiérarchie et pour 2 modèles hiérarchiques (Hiérarchie Aléatoire et Hiérarchie Originale) sur le dataset DMOZ. Pour la génération aléatoire, les scores ont été moyennés sur 80 expériences et les écarts-types ont été affichés directement sur les barres.

la hiérarchie, les problèmes de séparation peuvent être extrêmement complexes. Quand il s'agit de séparer des groupes de plus de 1000 concepts différents, il arrive fréquemment que l'erreur d'approximation explose étant donnée la relative simplicité des classifieurs utilisés dans ce type de modèle hiérarchique. Plus on va descendre dans la hiérarchie, plus on va avoir une erreur d'estimation liée à un mauvais apprentissage des paramètres à cause du manque de données. Il faut savoir que dans ce type de problèmes, le nombre d'exemples peut devenir excessivement petit pour certaines classes ce qui implique une trop grande variance des distributions des ensembles d'exemples de l'ensemble d'entraînement. Les bonnes performances en bas de la hiérarchie (à partir du *4ème* niveau) du modèle avec hiérarchie aléatoire s'expliquent par le fait que les exemples ambiguës et durs à classifier ont déjà été éliminés par les étages plus haut, ainsi que par le fait que la hiérarchie aléatoire se retrouve avec des problèmes de séparation de classes plus faciles en bas de la hiérarchie. En effet, une hiérarchie bien construite aura tendance à avoir des classes semblables proches en bas de la hiérarchie. Ainsi, les problèmes de séparation seront plus difficiles à résoudre. Inversement, le fait de tirer au hasard le partitionnement des classes construit naturellement des problèmes de séparation plus faciles en bas de la hiérarchie (deux classes prises au hasard ont plus de chance d'être facilement séparables

comparées à deux classes partageant beaucoup de similarités).

En pratique, ce sont les erreurs en haut de la hiérarchie qui prédominent. Le taux d'erreur est significativement plus grand pour les nœuds ayant à gérer une séparation avec beaucoup de classes. Il est ainsi primordial d'aider au mieux la tâche des classifieurs en haut de la hiérarchie en leur proposant de résoudre des problèmes de classification les plus simples possibles. On peut voir sur la Figure 3.4 les performances des méthodes de l'état de l'art de création de hiérarchies de classes. Nous avons comparé ici les performances à chaque niveau de hiérarchie pour un modèle utilisant du clustering spectral pour organiser les classes dans une hiérarchie et pour un modèle créant une hiérarchie aléatoirement. On peut noter un saut dans l'amélioration de la précision des classifieurs à partir du second niveau (juste en dessous du nœud racine) des hiérarchies. Le problème de partitionnement initial au niveau du nœud racine est très difficile étant donné le nombre de possibilités de partitionner un grand ensemble de classe. La méthode de clustering spectral ne parvient pas à faire mieux que la méthode aléatoire quand le nombre de classes est trop élevé. On peut noter aussi que les valeurs de précision par niveau sont similaires à celles du modèle avec hiérarchie originale à partir du second niveau et plus. On a la même perte de précision pour les niveaux les plus bas (5ème niveau) qui s'explique par le fait que la méthode a réussi d'une certaine façon à construire des partitions de classes cohérentes, donc plus difficiles à séparer vers les nœuds feuilles de l'arbre de classifieurs.

Les taux d'erreurs en bas d'une hiérarchie sont souvent des erreurs d'estimation dues à un manque d'exemples pour bien estimer les paramètres des classifieurs. En effet, on observe un grand nombre de classes assez peu représenté en terme de nombre d'exemples. Pour observer l'influence de ce manque de données conduisant à des erreurs d'estimation, nous avons étudié les précisions par rapport à la taille des classes (qui correspond au nombre d'exemples de cette classe présents dans l'ensemble d'entraînement). La Figure 3.5 montre ainsi ces précisions pour différentes tailles de classes pour un modèle hiérarchique à base de clustering spectral. On peut voir clairement un plus grand nombre d'erreurs pour des classes sous représentées.

L'utilisation d'un modèle ECOC permet de réduire ces erreurs.

Les modèles ECOC créent de nouvelles partitions des données en combinant plusieurs classes. Comme ils ne sont pas hiérarchiques, on peut éviter le phénomène d'attrition des

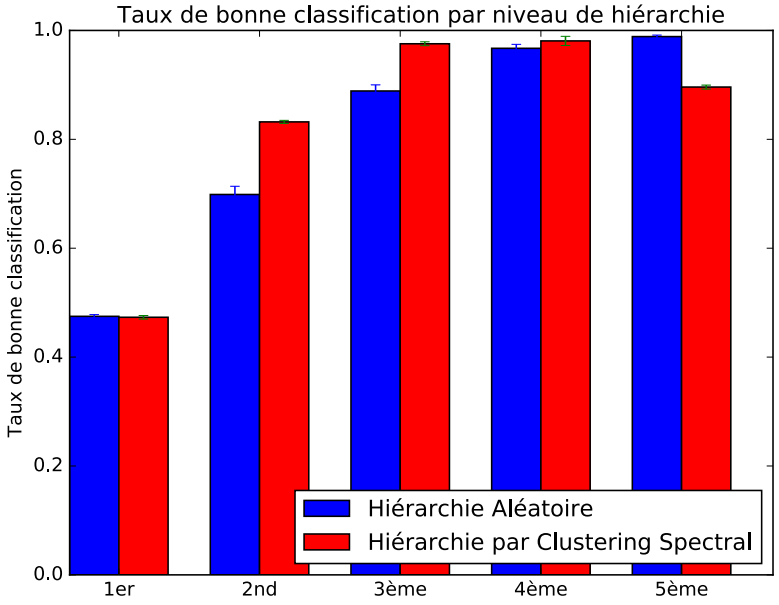


FIGURE 3.4: Score de précision de la classification pour chaque niveau de hiérarchie et pour 2 modèles hiérarchiques (Hiérarchie Aléatoire et Hiérarchie par clustering spectral) sur le dataset DMOZ. Les scores ont été moyennés sur 80 expériences et les écarts types ont été affiché directement sur les barres.

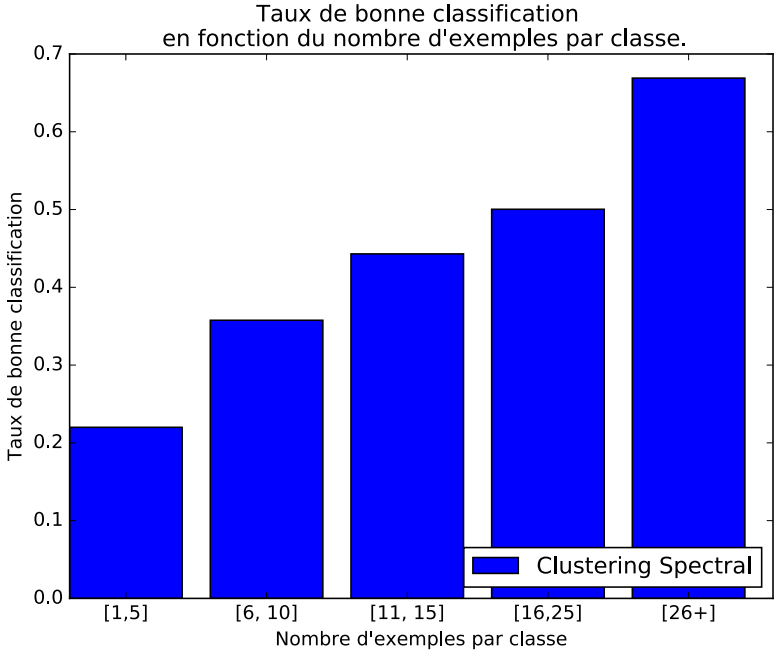


FIGURE 3.5: Taux de bonne classification d'un modèle hiérarchique pour différents ensembles de test découpés en fonction de la taille de chaque classe en terme de nombre d'exemples (résultats moyennés sur 80 expériences, sur le Dataset DMOZ).

données. On peut voir Figure 3.6 comment les modèles ECOC sont capables de corriger les erreurs des classes les moins fournies en exemples par rapport au modèle hiérarchique

utilisant du clustering spectral pour construire une hiérarchie.

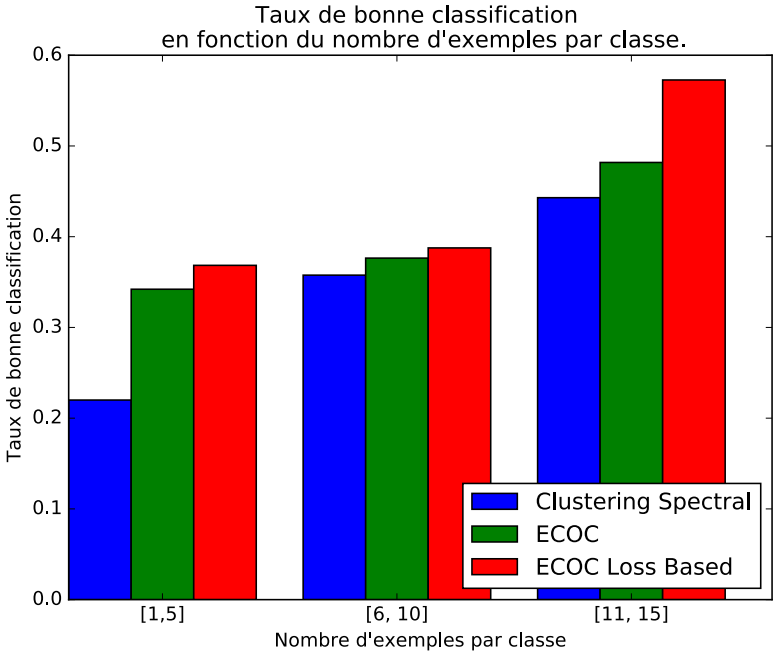


FIGURE 3.6: Taux de bonne classification pour différents modèles pour différents ensembles de test découpés en fonction de la taille de chaque classe en terme de nombre d'exemples (résultats moyennés sur 80 expériences, sur le Dataset DMOZ).

3.3.4 Étude du compromis Complexité-Précision

On peut voir sur la Figure 3.7 l'évolution de la précision de la méthode ECOC sur la base de données DMOZ en fonction du ratio de complexité (ratio entre la complexité du modèle et la complexité d'un modèle OAA). Ce ratio de complexité dépend directement de la longueur du code. Pour un ratio de complexité de 1, la longueur des codes pour chacune des classes est de K. On observe ainsi une perte significative de précision des modèles dès lors que les contraintes sur la longueur des codes sont trop fortes. A ce moment là, le nombre de classifieurs est trop réduit pour que le modèle conserve son taux de bonne classification.

La Figure 3.8 montre la précision d'un modèle hiérarchique utilisant une méthode de clustering spectral pour construire une hiérarchie. On peut voir que la méthode hiérarchique est capable d'obtenir des meilleurs performances que la méthode ECOC lorsque les contraintes sur le temps maximal alloué à la classification sont les plus fortes. Ceci s'explique par la capacité naturelle des méthodes hiérarchiques à atteindre rapidement une forte réduction du nombre de classifieurs à utiliser pour effectuer une prédiction.

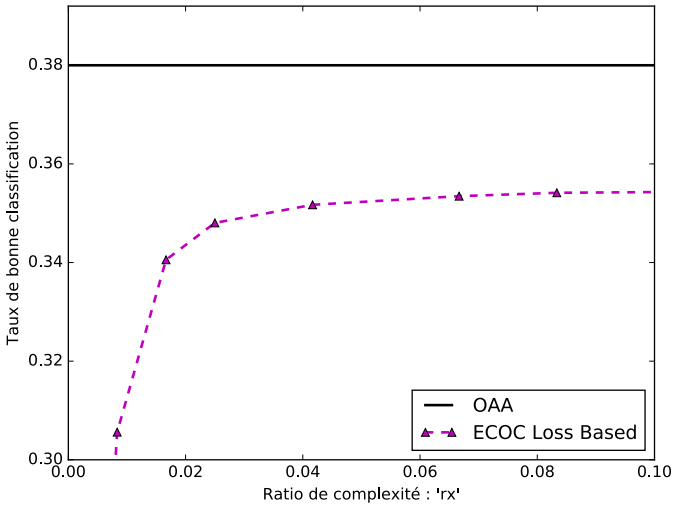


FIGURE 3.7: Taux de bonne classification pour un modèle ECOC en fonction du ratio de complexité r_{X} atteint (la ligne noire est le prolongement du point obtenu avec le modèle OAA d'abscisse $r_{\mathrm{X}} = 1$) que l'on ne peut pas voir sur la figure)

Une méthode ECOC à besoin d'une certaine longueur de code pour que le processus de correction d'erreur entre pleinement en jeu.

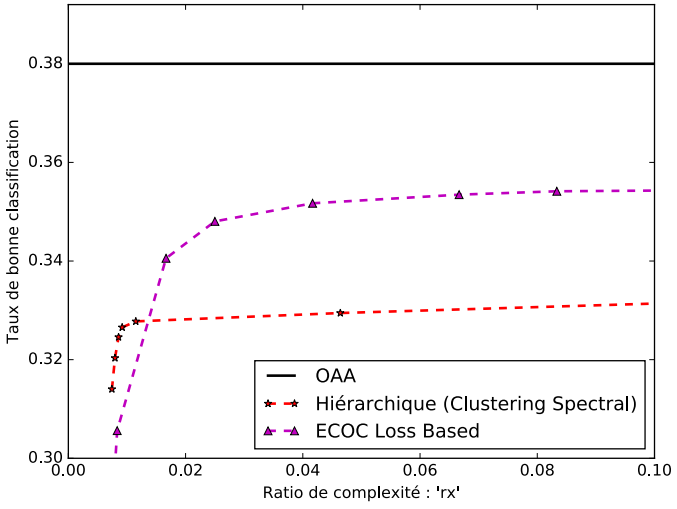
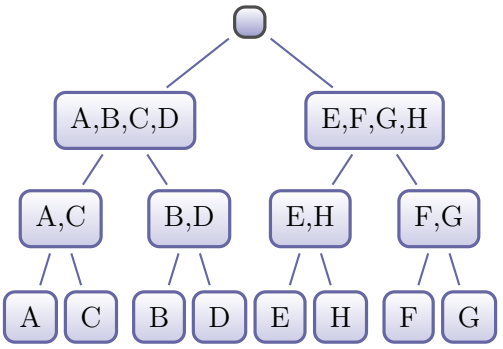


FIGURE 3.8

Pour illustrer plus clairement ces considérations, considérons un problème avec 8 classes $\{A, B, C, D, E, F, G, H\}$. On suppose une organisation hiérarchique de ces 8 classes comme ici :



Un modèle de classification hiérarchique reprenant cette organisation hiérarchique doit entraîner 7 classifieurs (un par dichotomie). Chaque classifieur va apprendre une certaine séparation des classes comme le montre le tableau suivant :

	A	B	C	D	E	F	G	H
$\mathbf{c}_1$	1	1	1	1	-1	-1	-1	-1
$\mathbf{c}_2$	1	-1	1	-1	0	0	0	0
$\mathbf{c}_3$	0	0	0	0	1	-1	-1	1
$\mathbf{c}_4$	1	0	-1	0	0	0	0	0
$\mathbf{c}_5$	0	1	0	-1	0	0	0	0
$\mathbf{c}_6$	0	0	0	0	1	0	0	-1
$\mathbf{c}_7$	0	0	0	0	0	1	-1	0

Par exemple, la dichotomie $\mathbf{c}_1$ correspond à la séparation au niveau du nœud racine. Pour obtenir une classification, un exemple devra *traverser* l'arbre depuis le nœud racine jusqu'à un nœud feuille. Cet exemple utilisera ainsi 3 classifieurs pour obtenir une prédiction.

Considérons cette fois un modèle de type ECOC. Il y a 8 classes à coder. Il suffit de 3 bits d'information pour séparer ces 8 classes. Ainsi, 3 classifieurs suffisent pour coder l'ensemble des classes.

L'encodage des dichotomies des classifieurs ECOC peut se faire facilement en suivant les dichotomies de chacun des étages de la hiérarchie (voir Figure 3.9).

Dans le cas du modèle hiérarchique tout comme dans le cas du modèle ECOC, une classification est obtenue après l'utilisation de trois classifieurs. On peut se rendre compte que dans notre exemple, le classifieur $\mathbf{c}'_3$ de la méthode ECOC doit résoudre les mêmes

	A	B	C	D	E	F	G	H
$\mathbf{c}'_1$	1	1	1	1	-1	-1	-1	-1
$\mathbf{c}'_2$	1	-1	1	-1	1	-1	-1	1
$\mathbf{c}'_3$	1	1	-1	-1	1	1	-1	-1

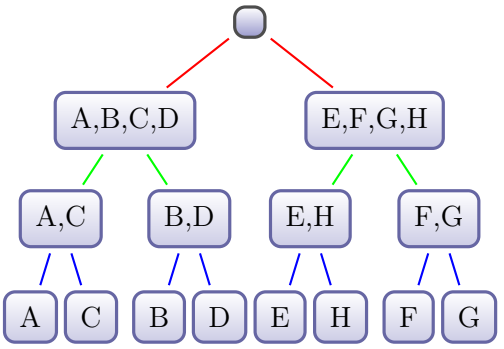


FIGURE 3.9: Tableau des dichotomies reprenant les dichotomies issues de chacun des étages de la hiérarchie donnée.

problèmes de séparation que tous les classifieurs $\mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_6$ et $\mathbf{c}_7$ (de la méthode hiérarchique). Dans le cas du modèle ECOC, le classifieur correspondant à la dichotomie $\mathbf{c}'_3$ a plus de données, mais a également une tâche plus complexe que les quatre classifieurs $\{\mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_6, \mathbf{c}_7\}$ du modèle hiérarchique.

Ceci illustre la difficulté pour une méthode ECOC à rester compétitive face à une méthode hiérarchique avec des contraintes de temps de classification très réduit. On voit bien la force de la méthode hiérarchique qui est capable de proposer des classifieurs de plus en plus spécifiques en fonction du chemin suivi par l'exemple. Contrairement au modèle hiérarchique, la méthode ECOC a un processus de classification similaire pour tous les exemples.

Il faut noter que le modèle ECOC ne bénéficie pas de correction d'erreurs dans le cas présent étant donné que les codes de chacune des classes sont très proches (distance de hamming de 1) les uns des autres. Cet exemple illustre un cas limite du fonctionnement ainsi que les points négatifs des méthodes présentées.

Les résultats Figure 3.8 montrent ensuite comment en relâchant les contraintes de complexité du processus de classification, les modèles ensemblistes de type ECOC sont capables de mieux corriger les erreurs et d'atteindre des niveaux de précision supérieurs aux modèles hiérarchiques. Lorsqu'il y a moins de contraintes de complexité, le modèle ECOC a le droit d'augmenter la longueur des codes de classes. Ce faisant, la distance séparant deux codes augmente ce qui a pour effet de *permettre* plus d'erreurs pour les classifieurs locaux sans pour autant amener à une mauvaise classification globale. Pour les modèles hiérarchiques, tant qu'il n'y a pas de redondance dans les classes, n'importe

quelle erreur de classification survenant pendant la descente de l'exemple dans l'arbre, aboutira à une mauvaise prédiction.

Cette simple comparaison montre aussi l'importance de la place en mémoire qu'occupe un modèle. Pour un même temps de classification logarithmique en fonction du nombre de classes, le modèle hiérarchique a besoin d'un nombre de classifieurs égal à $K-1$ alors que le modèle ECOC n'a besoin que de $\log(K)$ classifieurs en mémoire.

3.4 Conclusion

Nous avons décrit dans ce chapitre un formalisme regroupant les méthodes auxquelles nous nous sommes intéressés dans ce travail de thèse. Il est ainsi possible de voir que ces méthodes qui semblaient très différentes présentent des traits communs. L'étude des méthodes classiques rentrant dans notre formalisme nous a permis de dresser un tableau regroupant les spécificités de chacune d'elles. Nous avons ensuite étudié plus en détail ces méthodes dans le but de bien comprendre leur fonctionnement, leurs points forts ainsi que leurs limites. Ce travail a permis de mettre en évidence des pistes de recherche qui ont conduit aux contributions qui vont être présentées dans les chapitres suivants. En particulier, l'étude des méthodes classiques a permis de comprendre les qualités respectives des modèles hiérarchiques et des modèles ensemblistes de type ECOC. Nous verrons dans le chapitre 5 un modèle que nous avons développé combinant les forces des modèles hiérarchiques et des modèles ECOC. Avant cela, nous allons présenter un modèle purement hiérarchique dans le chapitre 4.

Chapitre 4

Création de hiérarchie et partitionnement de classes

4.1 Introduction

Le premier travail présenté ici repose sur une hiérarchisation des classifieurs dans une structure d'arbre. Étant donné les propriétés naturelles des arbres, une classification dans ce type de structure ne nécessite qu'un nombre très réduit de classifieurs par rapport au nombre de classes du problème. En effet, comme vu précédemment, le processus de classification dans un arbre de classifieurs équilibré nécessite un classifieur par noeud. Comme la hauteur de l'arbre est logarithmique par rapport au nombre K de classes présents au niveau des feuilles de la hiérarchie, le temps de classification peut être réduit jusqu'à $\log_2 K$ classifieurs pour un arbre binaire par exemple.

Ces hiérarchies de classes ou de concepts sont parfois pré-existantes et sont fournies avec les données du problème. On peut voir Figure 4.1 un exemple de hiérarchie de classes.

Bien entendu, il existe une infinité de façons de hiérarchiser les classes d'un problème. Si on considère un ensemble de mots qu'il faudrait organiser d'une façon hiérarchique, une façon de faire pourrait être de les hiérarchiser selon la nature des mots, alors qu'une autre approche pourrait privilégier une hiérarchisation selon la fonction des mots. Dans un cas comme dans l'autre, ces hiérarchies répondent à des impératifs différents selon la tâche de classification définie au préalable.

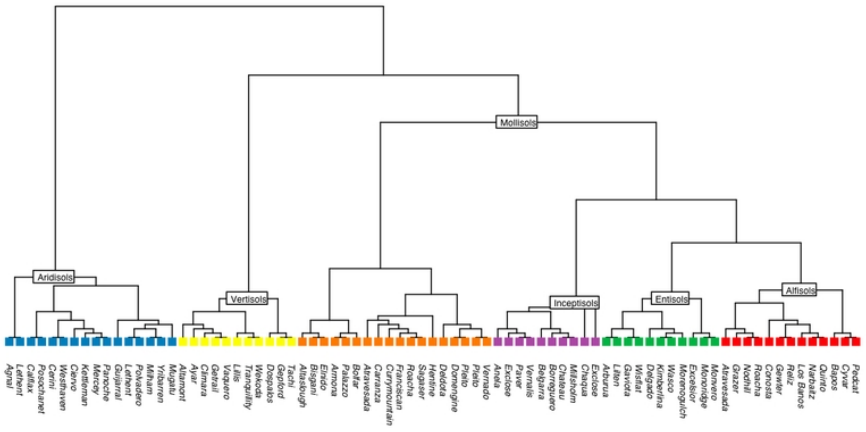


FIGURE 4.1: Exemple de hiérarchie de classes (*US Soil Taxonomy*).

L'idée à retenir est qu'une hiérarchie donnée peut ne pas être adaptée à une tâche de classification spécifique, d'où la volonté de proposer des méthodes permettant de créer de toutes pièces une hiérarchie de classes adaptée au problème à résoudre.

Dans la suite de ce chapitre nous allons tout d'abord présenter un travail préliminaire qui propose de créer une hiérarchie en utilisant un algorithme classique de construction de hiérarchies (bottom-up) mais doté d'une nouvelle métrique pour agréger les classes. Cette méthode montrera des faiblesses quant au passage à l'échelle et conduira à changer d'angle d'approche. Nous verrons alors un travail plus conséquent sur un nouvel algorithme dynamique de construction de hiérarchies capable de construire des hiérarchies de façon plus souple et doté d'une nouvelle méthode de partitionnement.

4.2 Construction de hiérarchie et mesure d'agrégation

Dans le cas où il n'y a pas d'ontologie de classes disponible ou adaptée à la tâche de classification, et que l'on souhaite néanmoins utiliser une méthode de classification hiérarchique, il est nécessaire d'en créer une. On rappelle que la tâche que l'on cherche à résoudre ici concerne une tâche de classification multi-classes mono-label, et que la hiérarchie que l'on cherche à construire ne sera pas redondante (une classe n'apparait que dans une seule feuille).

La création de cette hiérarchie va avoir un double impact par rapport au formalisme du schéma de classification, qui est décrit par 4 processus. Une hiérarchie de classes permet tout d'abord de définir précisément l'ensemble des classifieurs qui vont être entraînés

lors de l'apprentissage du modèle. En effet, le processus $f_{\mathcal{S}}$ en charge de la création de l'ensemble des classifieurs simples va utiliser les dichotomies de classes présentes à chacun des nœuds de l'arbre pour définir la tâche des classifieurs. Le second processus qui va bénéficier directement de cette hiérarchie de classe est le processus séquentiel $b_{\mathcal{S}}$. Ce processus qui a pour but de sélectionner le prochain classifieur à utiliser va reproduire la hiérarchie de classes par la hiérarchisation des classifieurs, ce qui permettra de faire descendre l'exemple à classifier dans l'arbre depuis le nœud racine jusqu'à un nœud feuille. Ce processus va ainsi regarder la réponse du dernier classifieur utilisé ce qui va guider naturellement vers l'utilisation d'un classifieur correspondant à un étage plus bas dans la hiérarchie. Tous les schémas de classification basés sur l'utilisation d'une hiérarchie de classes auront les mêmes spécificités dans le fonctionnement des processus $f_{\mathcal{S}}$ et $b_{\mathcal{S}}$. Nous allons ainsi nous concentrer sur la création d'une hiérarchie de classes qui permettra ensuite de générer un schéma de classification hiérarchique.

Nous avons vu comment la littérature a déjà exploré différentes voies pour la construction de hiérarchies de classes. Le problème de construction de hiérarchies peut être décomposé en deux tâches. La première tâche consiste à construire la structure de l'arbre. Deux grandes familles de méthodes de construction existent : 1/ les méthodes *top-down* et 2/ les méthodes *bottom-up*. La seconde tâche consiste à affecter les classes dans les nœuds : c'est le travail de partitionnement. Ces méthodes sont basées sur des mesures de similarités pour décider des classes à regrouper ensemble.

Pour bien résoudre le problème de partitionnement des classes, il faut trouver une mesure adaptée. Une méthode usuelle consiste à utiliser une matrice de confusion (obtenue via un schéma OAA) comme base pour créer une mesure de similarité [Bengio et al., 2010]. Cependant, cette méthode ne garantit pas la création d'une *bonne* hiérarchie dans notre contexte. En effet, dans ce contexte de classification dans un grand nombre de classes, la dimension des espaces initiaux peut être très grande. Avec de tels espaces, si les données sont creuses, même un perceptron simple est capable de séparer facilement une classe du reste des classes. Dans ce cas, la matrice de confusion ne fournit pas une information véritablement utile. Les hiérarchies qui en découlent sont ainsi construites sur une information de confusion trop peu *robuste*.

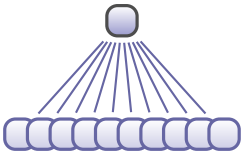
4.2.1 Algorithme d'agrégation

La mesure de similarité présentée a vocation à construire itérativement une hiérarchie en agglomérant des arbres (*bottom-up*). Au départ de l'algorithme, un ensemble de K nœuds est constitué : un par classe. Chacun de ces nœuds est considéré comme un arbre simple constitué d'un seul nœud racine. Cet ensemble de nœuds-arbres forment notre forêt initiale. Itérativement, les arbres les plus proches selon la mesure de similarité sont agrégés pour ne former qu'un seul arbre : les deux nœuds racines de deux arbres ainsi agrégés deviennent des nœuds frères, enfants d'un même nouveau nœud racine nouvellement créé. Ainsi, à chaque itération, la population d'arbres de notre forêt diminue de 1.

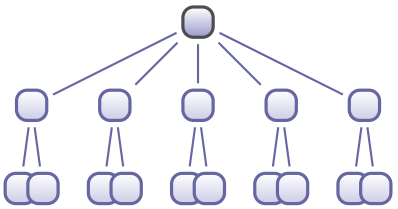
Il y a deux façons d'arrêter l'algorithme. Soit la population d'arbres de la forêt atteint 1, ce qui signifie qu'il n'y a plus d'arbres à agréger, et on a directement notre arbre final. Soit on stoppe l'algorithme selon un critère d'arrêt, et à ce moment là, il suffit d'agréger l'ensemble des arbres restant dans notre forêt d'arbres en un seul arbre en les mettant tous directement enfant d'un nouveau nœud racine. L'arbre final est ainsi constitué d'un nœud racine avec comme enfants l'ensemble des arbres qui constituent notre forêt.

Prenons l'exemple d'un problème avec 10 classes :

A l'itération 1, l'ensemble d'arbre initial est constitué de 10 arbres simples *racines*. Si l'algorithme était stoppé maintenant avant même d'avoir procédé à la moindre agrégation, on obtiendrait un arbre *à plat* :



On peut se retrouver après 5 itérations avec 5 arbres ayant chacun deux classes. Pour finaliser la hiérarchie après ces 5 itérations, l'ensemble de ces arbres est utilisé pour former un seul arbre contenant un nœud racine avec comme enfants les 5 arbres :



L'algorithme 3 montre le processus de création de la hiérarchie. A la fin du processus de création de la hiérarchie, on peut obtenir des hiérarchies qui ne sont pas forcément bien équilibrées comme sur la Figure 4.2.

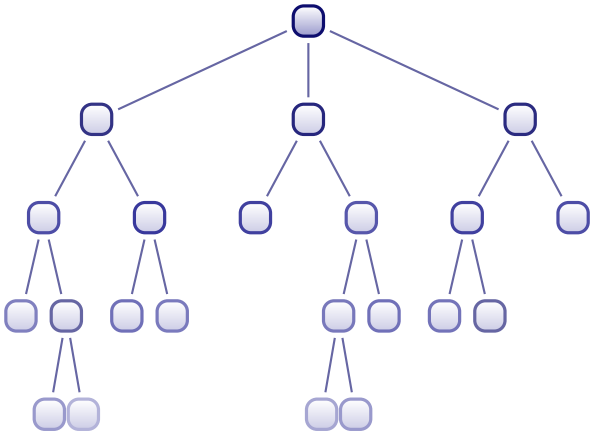


FIGURE 4.2: Arbre de classes

Le fait de découper la création de l'arbre en une phase itérative d'agrégation binaire, suivie d'une phase finalisant la hiérarchie par une agrégation n-aire a pour but de répartir la complexité dans l'arbre en fonction de la difficulté des problèmes de classification. Généralement, il est observé que les premiers niveaux d'une hiérarchie ont à résoudre des problèmes de classification plus compliqués. De ce fait, il est nécessaire d'avoir des modèles plus complexes en haut de la hiérarchie. Une solution est de définir un problème de classification n-aire et d'employer n classifieurs binaires. C'est ce que nous proposons ici.

L'algorithme 3 présente ce processus d'agrégation. La mesure de similarité utilisée dans l'algorithme 3 est libre. Par exemple, on peut utiliser la mesure basée sur une matrice de confusion construite à partir de mauvaises classifications d'un classifieur multi-classes OAA classique vu précédemment. Cette mesure d'agrégation présentant des défauts, nous allons voir dans la sous-section suivante une mesure d'agrégation améliorée.

Algorithm 3 Algorithme d'agrégation *bottom-up*

1: **Init.** : $Foret = \{T_1, \ldots, T_{|Y|}\}$ l'ensemble d'arbres simples constitué d'un seul nœud, un par classe l_i.
2: **tant que** Complexité($Foret$) $<$ Complexité visée **faire**
3: *Trouver* les deux arbres T_i, T_j les plus proches selon la mesure de similarité.
4: *Créer* un nouvel arbre avec un seul nœud racine n_k. Placer les arbres T_i et T_j comme les deux nœuds fils du nœud racine n_k.
5: **fin tant que**
6: *Combiner* tous les arbres restants en les plaçant en tant que fils d'un unique nœud racine.
7: *Apprendre* les classifieurs dans chacun des nœuds.

4.2.2 Mesure d'agrégation

Un travail préliminaire a été effectué sur cette question de mesure d'agrégation. Nous avons développé une mesure basée sur des récentes études qui ont montré d'un point de vue théorique que la séparabilité de deux ensembles de points est fortement corrélée à la complexité d'un arbre de boosting entraîné à séparer ces deux ensembles de points [Shalev-Shwartz and Singer, 2010]. L'idée ici est d'utiliser la capacité à facilement sur-apprendre des arbres de boosting (par l'augmentation de la profondeur des arbres) ainsi que leur efficacité computationnelle dans le but de regrouper les problèmes de classification par rapport à leur difficulté. Simplement, plus un problème de séparation requiert d'époques de boosting avant de sur-apprendre, plus cette séparation concerne des concepts complexes.

Ainsi, les nœuds sont agrégés en regardant les nœuds dont les problèmes de séparation sont *les plus simples*.

La politique d'agrégation des nœuds prend en considération d'une part la proximité *sémantique* de deux classes (via l'utilisation d'un indicateur de confusion entre classes), ainsi que la proximité *de niveau d'abstraction* obtenue via l'utilisation d'une mesure basée sur les capacités de sur-apprentissage des arbres de boosting. Par exemple, une classe "Chats" est proche sémantiquement d'une classe "Félins", mais on préférera mettre ensemble "Chats" et "Tigres" avant de considérer l'agrégation de la classe "Félins" qui correspond à un niveau d'abstraction plus élevé que les deux classes *plus simples*.

Nous avons ainsi tenté de construire une mesure capable de rendre compte du niveau d'abstraction et de la simplicité des problèmes de classification binaire dans le but de commencer l'agrégation des classes de plus faible niveau d'abstraction. La mesure du

sur-apprentissage dans le boosting donne une idée de la complexité des problèmes de séparabilité, permettant de se concentrer sur l'agrégation des classes les plus simples à séparer.

L'algorithme 4 présente comment la mesure du *niveau d'abstraction* fonctionne. On commence par apprendre un arbre de boosting de hauteur 1. Si le taux de mauvaise classification est supérieur à un ϵ fixé, on incrémente le nombre d'époques pour l'arbre de boosting et on réitère jusqu'à ce que le taux de mauvaise classification soit inférieur à ϵ.

Algorithm 4 Algorithme de mesure du niveau d'*abstraction* entre deux ensembles de classes.

Entrée: $\mathcal{A}$ et $\mathcal{B}$ deux ensembles de classes.
Soit $b^d_{\{\mathcal{A},\mathcal{B}\}}$ un modèle d'arbre de boosting ayant appris à séparer $\mathcal{A}$ et $\mathcal{B}$, et utilisant un nombre d'*époques* d.
Soit $e(b^d_{\{\mathcal{A},\mathcal{B}\}})$ le taux de bonne classification du modèle $b^d_{\{\mathcal{A},\mathcal{B}\}}$.
Initialisation : $t = 1$
tant que $e(b^t_{\{\mathcal{A},\mathcal{B}\}}) < \epsilon$ **faire**
  $t = t + 1$
fin tant que
retourner t

Modèles	Type	Complexité r_x	DMOZ : Acc% (std) Set1	Set2	Set3
OAA	Plat	1 (×1)	60.5% (0.2)	75.05% (0.6)	59.4% (0.5)
Notre modèle	Arbre	**0.2** (×5)	**58.75**% (0.3)	**74.56**% (0.4)	**58.7**% (0.6)
Clustering Spectral	Arbre	0.2 (×5)	54.47% (1.3)	71.65% (0.8)	53.6% (1.1)
ECOC-Hamming	Plat	0.2 (×5)	47.07% (3.0)	61.38% (8.6)	44.8% (3.0)
	Plat	0.5 (×2)	57.27% (0.9)	72.93% (0.9)	56.1% (1.1)

FIGURE 4.3: Taux de bonne classification pour différents modèles sur des échantillons de 1000 classes de la base de données DMOZ.

Bien que cette méthode ait permis d'obtenir des bons scores de classification (voir tableau 4.3) sur des problèmes de classification de 1000 classes, elle n'a pas pu montrer qu'elle était capable de passer à l'échelle et d'offrir des performances similaires sur des bases de données plus larges.

Ce constat rejoint les commentaires de Langford [Choromanska and Langford, 2014] sur la question des problèmes liés à l'utilisation de méthodes bottom-up pour construire une hiérarchie. Il est très difficile d'avoir un horizon clair à long terme lors de l'agrégation de deux classes tout en bas de la hiérarchie en devenir. Cela explique pourquoi il semblerait que cette méthode ne puisse pas obtenir sur des problèmes de très grande taille des résultats à la hauteur des résultats obtenus sur des bases de données avec un nombre

de classes moins important. Le travail qui s'en est suivi s'est concentré sur l'élaboration d'une méthode top-down. Nous allons le présenter dans la section suivante.

4.3 Construction de hiérarchie par dichotomie (DSAA)

Le travail présenté ici concerne aussi la création d'une hiérarchie de classe. Cependant à la différence des méthodes vues précédemment, l'algorithme de construction de la hiérarchie va chercher à optimiser la structure en parallèle de la recherche de hiérarchie.

En effet, les différents travaux étudiés précédemment de génération de hiérarchie n'apprennent pas la structure. Ces approches génèrent généralement un arbre de structure fixe. Le plus souvent, cette structure dépend d'un paramètre spécifiant le nombre d'enfants par nœud. Ainsi, la structure de la partition est fixée à l'avance et le but de l'algorithme de construction de la hiérarchie va être d'allouer les étiquettes des labels aux nœuds de cette structure fixe. De plus, le mécanisme de construction de la hiérarchie commence par un bout de la hiérarchie (soit par le haut, soit par le bas). Cette rigidité limite en pratique la capacité de la hiérarchie produite à ressembler à l'ontologie latente organisant au mieux les classes du problème. Ces méthodes reprennent généralement l'un des deux schémas de construction habituels : *top-down* ou *bottom-up*. Ces approches sont dites gourmandes car elles optimisent au mieux la situation de l'instant présent sans chercher à savoir si la solution trouvée optimisera le critère final. Elles sont ainsi sujettes à des problèmes d'optimalité de la hiérarchie produite.

De façon plus visuelle, la Figure 4.4 montre ce qu'une méthode top-down cherche à faire lors des premières itérations de construction de la hiérarchie. L'algorithme va chercher en premier lieu à partitionner le nœud racine pour former le premier étage. Dans le cas où l'espace des données est très grand (données textuelles par exemple), le nombre de paramètres des classifieurs simples à apprendre est naturellement élevé. Il est facile dans ce cas là de faire du sur-apprentissage de modèles. Cela peut rendre le partitionnement en deux groupes assez facile sur l'ensemble d'apprentissage et conduire à des sous-ensembles de classes trop différentes les unes des autres. Dès lors, le reste du partitionnement peut souffrir de ces premières dichotomies.

La Figure 4.5 montre le principe d'une méthode bottom-up. Ce que l'on cherche à illustrer ici est le côté arbitraire et statique des méthodes de construction de hiérarchies classiques.

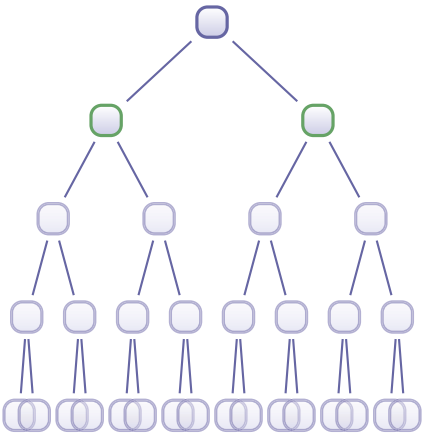


FIGURE 4.4: Les nœuds en vert représentent l'ensemble des nœuds que cherche à construire une méthode top-down de construction de hiérarchie lors des premières itérations de l'algorithme.

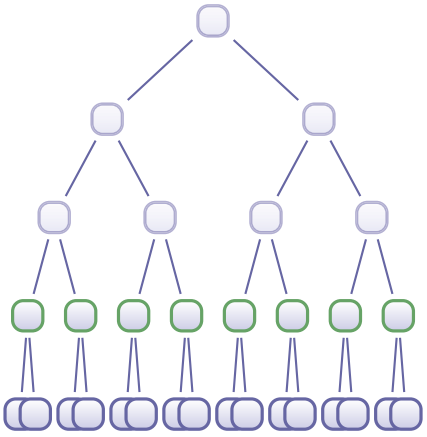


FIGURE 4.5: Les nœuds en vert représentent l'ensemble des nœuds que cherche à construire une méthode bottom-up de construction de hiérarchies lors des premières itérations de l'algorithme.

Par exemple, une mesure de similarité peut ne pas être fiable lorsqu'il s'agit de mesurer une similarité entre deux classes parmi un très grand ensemble de classes [Marszalek and Schmid, 2010]. D'où la difficulté d'utiliser une méthode bottom-up.

Ainsi, il n'est pas forcément optimal de commencer par le bas, ou par le haut, pour un problème donné. Une des idées mise en avant dans ce travail est d'autoriser plus de souplesse dans la création de la hiérarchie en autorisant des partitionnements moins contraints. Cela va engendrer des constructions de hiérarchies où l'algorithme va commencer à apprendre les nœuds centraux comme montré très schématiquement sur la Figure 4.6. L'intuition derrière cela est de considérer que la création de hiérarchies peut être plus facile en commençant à apprendre des regroupements de classes ni trop petits (cas du *bottom-up*) ni trop gros (cas du *top-down*).

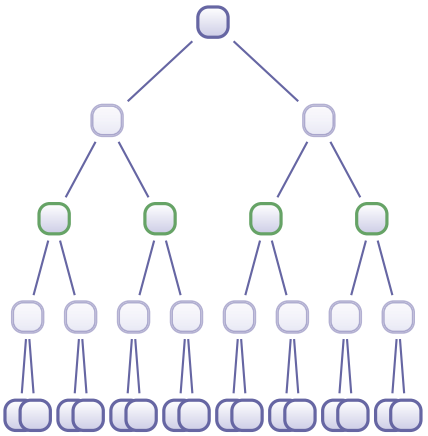


FIGURE 4.6: Les nœuds en vert représentent l'ensemble des nœuds que cherche à construire une méthode par dichotomie de construction de hiérarchie lors des premières itérations de l'algorithme.

Le travail présenté ici propose un nouvel algorithme gourmand visant à apprendre une structure adéquate (correspondant aux ensembles N et E du formalisme précédent), ainsi que l'assignation des labels (L) dans l'arbre, le tout de façon simultanée. L'algorithme de construction de l'arbre est novateur dans le sens où la construction de l'arbre vise à raffiner l'arbre courant par développement successif des nœuds. Notre algorithme est dynamique dans le sens où le nœud à développer ainsi que le nombre d'enfants vont être spécifiques à l'état de la hiérarchie courante.

4.3.1 Présentation de l'approche

La première étape de l'algorithme proposé ici est de sélectionner un nœud existant dans l'arbre dans le but de le *développer* en plusieurs clusters. Choisir le nœud à développer permet de contrôler l'évolution de la structure de l'arbre de classe ainsi que de contrôler le gain de vitesse de classification que permet l'utilisation de la hiérarchie. La Figure 4.7 illustre schématiquement une itération du processus de sélection et de développement de l'arbre.

L'optimisation de l'affectation des labels parmi les nouveaux nœuds créés est fortement corrélée à la performance finale du modèle. A l'inverse des méthodes habituelles commençant la création de la hiérarchie par le haut ou par le bas, l'algorithme proposé ici vise à créer les niveaux du milieu de la hiérarchie avant de la compléter itérativement. La Figure 4.8 montre le fonctionnement global de l'algorithme.

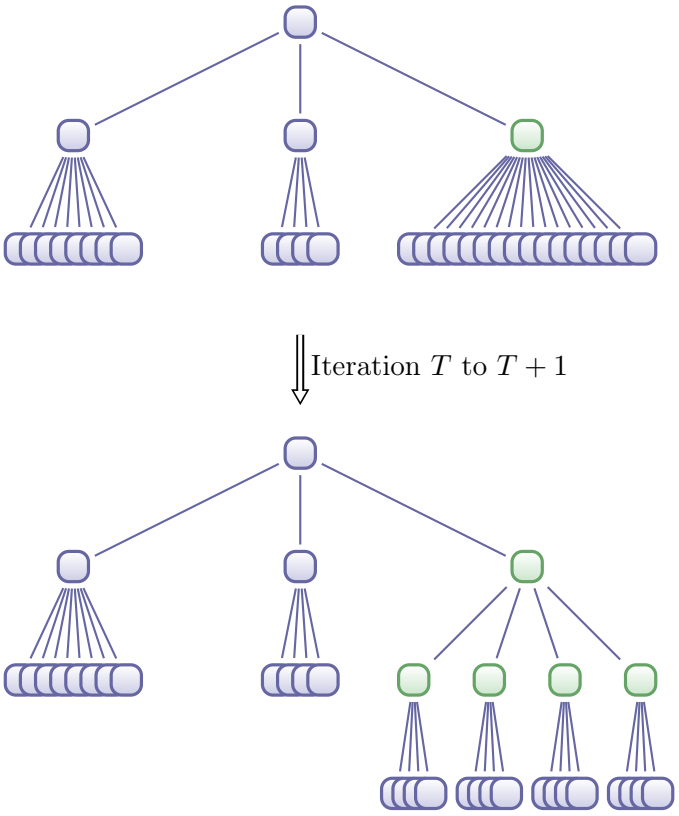


FIGURE 4.7: Illustration d'une itération de l'algorithme. Tout d'abord le nœud vert est sélectionné pour être développé car il représente le nœud qui ralentit le plus le processus de classification. Ensuite, un partitionnement de ses enfants est calculé de façon à favoriser les erreurs *au sein des* clusters plutôt que les erreurs *entre* les clusters.

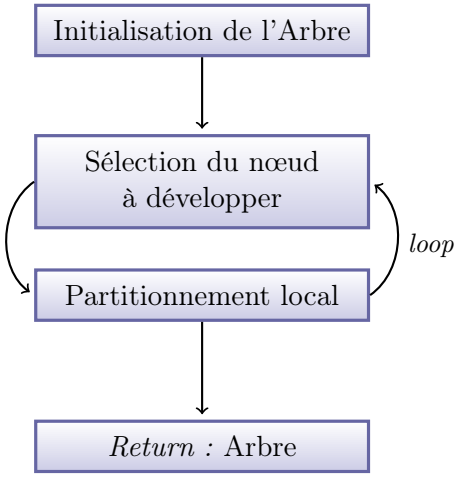


FIGURE 4.8: Diagramme de fonctionnement de l'algorithme global. A l'initialisation, l'arbre est *plat* : un nœud racine connecté à K feuilles avec K classes différentes. À chaque itération, un nœud est sélectionné et un partitionnement de ses enfants est calculé (à l'initialisation, il n'y a que le noeud racine à développer). Une fois que la hiérarchie est suffisamment *rapide* (i.e plus profonde que plate), le processus s'arrête. Les classifieurs dans chacun des nœuds sont appris ensuite indépendamment les uns des autres.

Une fois que le nœud à développer a été sélectionné, le processus de partitionnement de labels commence. L'idée de base du partitionnement est de créer des clusters de classes les plus cohérents possibles. C'est à dire qu'il faut chercher à mettre ensemble les concepts proches sémantiquement. La figure 4.9 montre de façon très schématique le but recherché lors de la tâche de partitionnement.

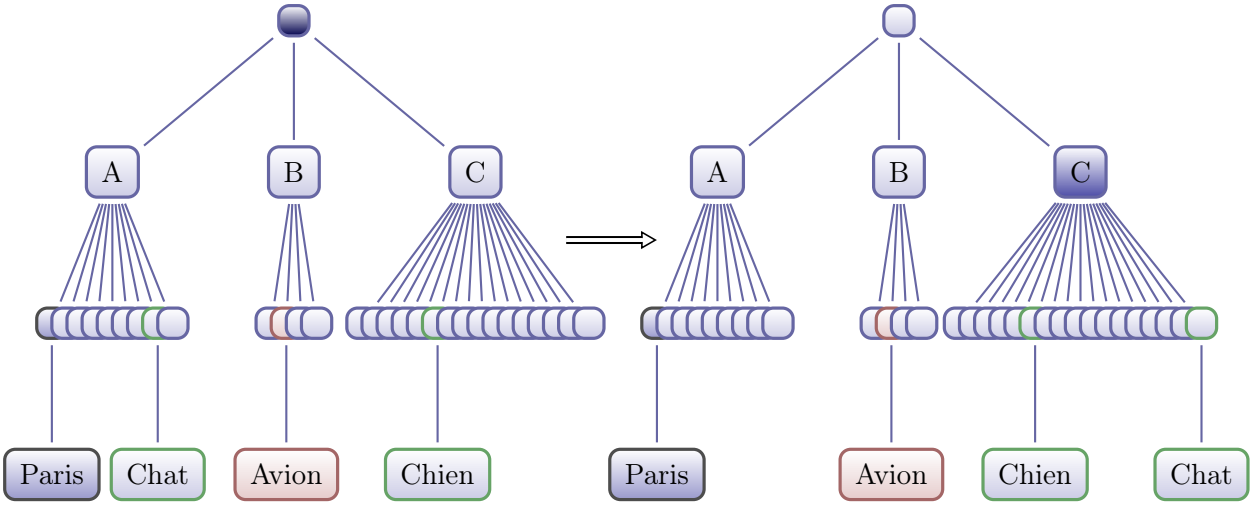


FIGURE 4.9: Exemple de partitionnement. Considérons deux classes : {Chat} et {Chien}. On calcule la séparabilité de chaque classe avec les clusters A, B et C. Il apparaît alors que les exemples appartenant à la classe {Chat} font plus de confusion avec les exemples du cluster C avec {Chien} dedans qu'avec les deux autres clusters A et B. Ainsi, le nœud {Chat} va être assigné au cluster C avec {Chien} dedans plutôt que de rester dans le cluster A.. Cela aura pour conséquence de faciliter la séparabilité des clusters en haut de la hiérarchie.

Pour bien saisir l'idée du fonctionnement de l'algorithme, il faut que l'on présente d'abord la notion de plus proche ancêtre commun pour deux classes. Le plus proche ancêtre commun de deux nœuds est le nœud le plus bas de l'arbre contenant les deux classes. L'ancêtre commun de deux classes qui sont difficiles à distinguer est mieux situé le plus *bas* possible dans la hiérarchie car la difficulté de la tâche de séparation de ces deux classes n'affectera pas négativement la précision des classifieurs des nœuds situés plus haut dans la hiérarchie. De plus, un classifieur entraîné à séparer uniquement deux classes sera plus précis que s'il avait été entraîné à séparer ces deux classes *noyées* dans des ensembles de classes. (la tâche de séparation de l'ensemble A avec l'ensemble B est plus difficile que la tâche de séparation du sous-ensemble $\hat{A} \subset A$ avec le sous-ensemble $\hat{B} \subset B$). Ainsi, le fait de chercher à placer le plus bas possible l'ancêtre commun de classe difficile à séparer est bénéfique pour la qualité des classifieurs de la hiérarchie. De plus, en faisant en sorte

que les tâches difficiles de séparation arrivent le plus tard possible dans le processus de classification, les problèmes de séparation en haut de la hiérarchie sont facilités.

L'algorithme proposé construit itérativement les partitions de classes en s'assurant que l'ancêtre commun des paires de classes difficiles à séparer soit le plus bas possible dans la hiérarchie. L'apprentissage de cet arbre de classification peut être divisé en deux parties : 1/ l'algorithme de modification itérative de l'arbre dont le but est de sélectionner le prochain nœud à développer (Section 4.3.2) et 2/ l'algorithme de partitionnement des classes utilisé lorsqu'un nœud doit être développé (Section 4.3.3).

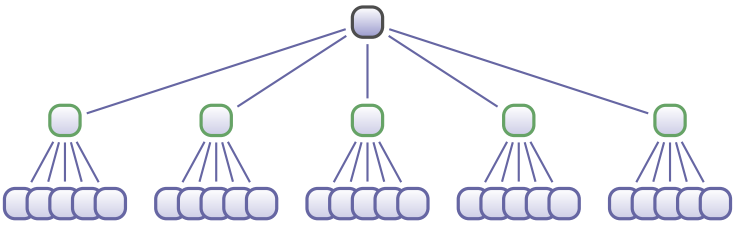
4.3.2 Algorithme de sélection du nœud à développer

Cet algorithme vise à la sélection du prochain nœud à développer pour permettre une classification plus rapide.

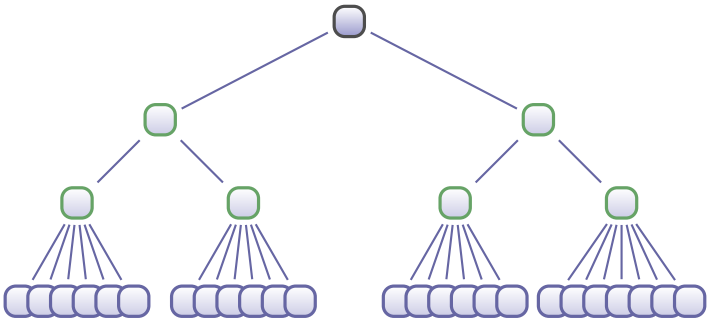
Les nœuds d'un arbre peu profond ont besoin de classifieurs d'une plus grande complexité pour distinguer parmi plusieurs enfants. A chaque itération, l'algorithme va introduire des nouveaux niveaux dans l'arbre, ce qui va avoir pour effet d'accélérer la classification de nouveaux exemples. Comme notre but ici est de réduire le temps de classification d'un nouvel exemple, nous allons utiliser la mesure *ratio de complexité* r_{x} qui a été définie précédemment (sous-section 3.2.2). Ce ratio calcule le rapport entre le nombre moyen de classifieurs utilisés pour classifier un nouvel exemple avec le modèle, et le nombre de classifieurs qu'utiliserait un modèle classique de classification OAA. Par exemple, le nombre de classifieurs utilisés dans un modèle hiérarchique avec un arbre binaire équilibré est égal à la profondeur de l'arbre.

Étant donné un ratio de complexité R_{min} à atteindre, l'algorithme va itérativement développer l'arbre jusqu'à ce que le modèle ait atteint le ratio de complexité voulu. Théoriquement, le plus petit nombre de classifieurs requis pour obtenir une classification avec un modèle hiérarchique est de $\log_2(|K|)$ avec K le nombre de classes (il s'agit alors de construire un arbre binaire parfaitement équilibré).

Cependant, notre but ici n'est pas de construire l'arbre le plus rapide, mais plutôt de trouver un compromis entre la complexité de l'arbre (le nombre de classifications nécessaires pour prédire une classe pour un nouvel exemple) et la qualité de la prédiction de l'arbre. Nous avons constaté empiriquement que le fait de rendre l'arbre plus profond,



Structure A



Structure B

Deux structures de hiérarchies différentes (A et B) mais ayant la même complexité : en supposant les feuilles équilibrées en nombre d'exemples, les deux structures ont besoin de 10 classifieurs pour obtenir un chemin complet allant du nœud racine à un nœud feuille.

	Complexité	Δ-Précision
OAA	25	0
Structure A	10	$-3,1$ (0.4)
Structure B	10	$-5,7$ (0.5)

Perte de précision liée à l'utilisation de hiérarchies générées aléatoirement selon deux structures différentes. Conduite sur 2 Datasets DMOZ et Wikipedia avec 1000 générations aléatoires (la variance est affiché entre parenthèses).

FIGURE 4.10: Étude de l'influence de la structure sur la performance des hiérarchies.

et donc de partitionner encore plus finement les classes dans une hiérarchie plus précise a tendance à dégrader les performances du modèle hiérarchique. Ainsi, plus il y a de partitionnement des classes, plus il y a des erreurs de partitionnement qui peuvent venir des erreurs d'approximation liées à l'impossibilité de bien partitionner les classes.

Considérons un problème de classification avec 25 classes. Pour simplifier le propos, on suppose une densité équilibrée des exemples parmi ces 25 classes. On suppose qu'on vise une prédiction utilisant seulement 10 classifieurs. Il y a différentes façons d'obtenir cette complexité de classification avec des structures de hiérarchies différentes comme les deux structures de la figure 4.10.

Ces deux hiérarchies ont une complexité de classification similaire en supposant les classes équilibrées entre elles.

Nous avons généré aléatoirement un grand nombre de hiérarchies suivant différentes structures. Les résultats expérimentaux tendent à accréditer l'idée qu'il vaut mieux une hiérarchie simplifiée pour minimiser les erreurs de partitionnement.

Notre approche vise ainsi à limiter le plus possible les étapes de partitionnement lors de la création de la hiérarchie. L'idée est de diminuer le plus rapidement possible la complexité de l'arbre en un minimum d'étapes, de façon à minimiser le rapport *profondeur* sur *complexité*. En pratique, nous avons travaillé avec une complexité à atteindre fixée. Pour minimiser le nombre d'étapes de partitionnement, il faut que chaque étape soit optimale en terme de gain de complexité. L'algorithme que nous présentons cherche ainsi à optimiser le gain en complexité à chaque itération.

L'algorithme sélectionne le nœud de l'arbre qui coûte le plus cher lors de l'inférence d'un exemple dans le but d'ajouter un niveau de hiérarchie sous ce nœud là, et ainsi réduire son coût de classification. On choisit le noeud $\hat{n}$ tel que $\hat{n} = \arg\max_n w_n|\mathcal{E}_n|$ (où w_n correspond à l'estimation de la probabilité d'utilisation du noeud et $\mathcal{E}_n$ correspond à l'ensemble de ses noeuds fils). L'Algorithme 5 décrit le processus itératif d'apprentissage de la hiérarchie de classe. Au début, l'arbre est plat. Il possède un nœud racine connecté directement à K feuilles. A chaque itération, le nœud avec le plus grand coût est sélectionné pour être développé en utilisant l'algorithme de partitionnement qui sera décrit plus tard. Quand l'arbre courant est capable d'effectuer une classification aussi rapidement que requis, l'algorithme met fin au processus itératif et apprend les classifieurs de tous les nœuds de la hiérarchie.

Algorithm 5 Apprentissage de l'arbre de classes

Entrée: R_{min} : La complexité minimale à atteindre. Initialiser T avec un arbre plat : $T = T^0$
tant que $r_{\mathrm{x}}(T) > R_{min}$ **faire**
  $\hat{n} = \arg\max_n w_n|\mathcal{C}_n|$
  Calcul du partitionnement du noeud $\hat{n}$
  Mise à jour de l'arbre T
fin tant que
retourner T

Une fois que le nœud à développer a été sélectionné, il faut déterminer le nombre de clusters qui vont être créés sous ce nœud (développer un nœud consiste à ajouter un

niveau de hiérarchie sous ce nœud). Le choix de ce nombre de clusters suit la même stratégie que précédemment : le but est de réduire la complexité du modèle en minimisant le nombre d'étapes de partitionnement. On note Z le nombre de fils du noeud n avant développement, x le nouveau degré du nœud n après développement et y le degré de ses fils (voir Figure 4.11).

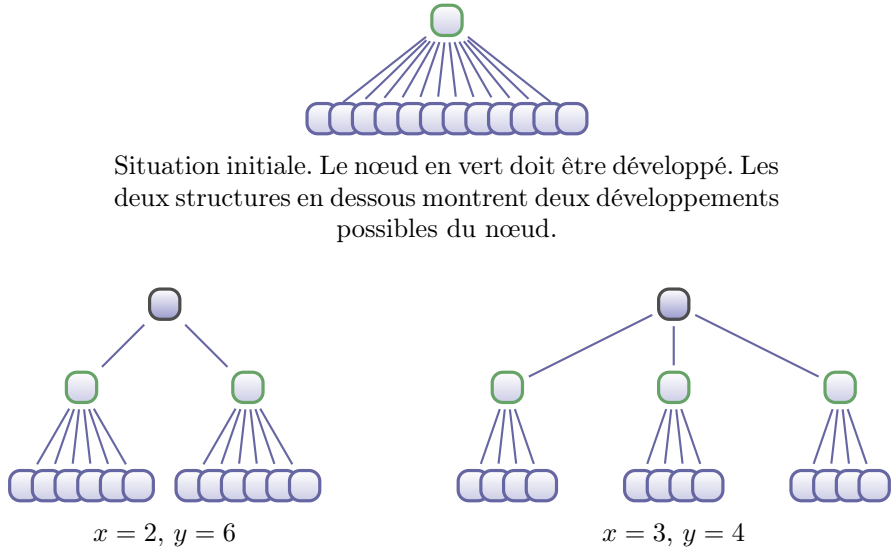


FIGURE 4.11: Illustration des notations x et y lors du développement d'un nœud.

La valeur $x + y$ correspond au coût de classification (le nombre de classifieurs) de ces deux étages de l'arbre. En effet, on considère qu'on utilisera un classifieur multi-classes classique OAA dans chaque nœud. Un classifieur qui doit sélectionner un nœud parmi k nœuds a besoin de k classifications [1]. Pour obtenir le coût de classification, il suffit alors de comptabiliser le nombre d'arêtes de chacun des nœuds traversés pendant la classification d'un exemple. Ainsi, minimiser $x + y$ revient à minimiser le coût de classification lors de l'ajout d'un niveau de hiérarchie (en supposant que les noeuds sont équilibrés en terme de nombre d'exemples). Le système à résoudre peut s'écrire ainsi :

$$\text{minimiser}\, x + y \quad \text{s.t} \quad x\ y = Z, x > 0,\ y > 0,\ Z > 0$$

La valeur $x \times y$ correspond au nombre de nœuds enfants qui est constant et égal à Z. La minimisation de cette équation est obtenue quand $x = y$.

1. En réalité, le cas limite de la classification binaire n'a besoin que d'un seul classifieur, et non de deux.

Par exemple, on considère un arbre avec un nœud racine connecté directement à 10000 feuilles. En résolvant le système, on obtient $x = 100$ et $y = 100$ ce qui signifie que la création d'un nouveau niveau de hiérarchie avec 100 noeuds est optimal vis à vis du gain de complexité. Ainsi, 100 clusters vont être créés à partir des 10000 classes. Le coût de classification d'un exemple sera alors de $200(= 100 + 100)$ au lieu de 10000.

4.3.3 Algorithme de partitionnement des classes

Nous avons vu comment l'algorithme de sélection du nœud opère ainsi que le choix du nombre de clusters à construire. Nous allons présenter maintenant ce qui se passe une fois que le nœud a été sélectionné et qu'il faut partitionner les nœuds en un certain nombre de clusters.

Le but de cet algorithme est d'apprendre un partitionnement adéquat des enfants d'un nœud donné de l'arbre. Ces clusters de nœuds enfants permettront ensuite la création d'un niveau intermédiaire dans la hiérarchie qui sera situé juste en-dessous du nœud à développer.

Étant donné un nœud n, son ensemble de classes associées s_n et le nombre de clusters à créer Q, le processus de partitionnement vise à apprendre un partitionnement optimal :

$$\mathcal{C}_n = \{c_1, \ldots, c_Q\} \tag{4.1}$$

$$\text{avec} \qquad c_i \subset s_n \tag{4.2}$$

$$\bigcup_i c_i = s_n \tag{4.3}$$

$$\forall (c_i, c_j), c_i \cap c_j = \emptyset \tag{4.4}$$

Pour affecter les classes aux noeuds fils, nous proposons une heuristique. Elle repose sur l'idée suivante : si deux classes sont difficiles à séparer alors il faudra les séparer à un noeud le plus bas possible dans l'arbre. Cela représente deux avantages : 1/ limiter les erreurs au noeud père et 2/ la séparation tardive de ces classes permettra d'affiner la frontière de décision et de minimiser les erreurs entre ces deux classes. Dans cette optique, on va définir une mesure *noeud-cluster* basée sur la précision de la reconnaissance des exemples des classes du *noeud* par rapport aux exemples des classes présentes dans le *cluster*. L'avantage de cette mesure est qu'elle est propre à la famille de classifieurs qui

sera utilisée dans la hiérarchie. Ainsi, la mesure sera plus adéquate car spécifique à cette famille de classifieurs.

Définissons $f_{y,c}$ le classifieur entraîné à séparer la classe y de l'ensemble de classes c, qui retourne $+1$ si x est classifié comme appartenant à la classe y, et -1 si l'exemple est classifié comme appartenant à l'ensemble de classes c. On définit par $e(y, c)$ la somme des erreurs de classification des exemples de la classe y lorsqu'ils sont classifiés par $f_{y,c}$:

$$e(y, c) = \sum_{i|y_i=y} \mathbb{1}(f_{y_i,c}(x_i) \neq 1)$$

Nous souhaitons maximiser cette erreur *intra* sur l'ensemble des clusters :

$$\underset{\mathcal{C}}{\text{maximiser}} \quad \sum_c \sum_{y \in c} e(y, c) \tag{4.5}$$

$$\text{tel que :} \quad \forall (c_i, c_j),\ c_i \cap c_j = \emptyset \tag{4.6}$$

$$\bigcup_i c_i = s_n \tag{4.7}$$

$$\forall c,\ \|c\| \leq c_{max} \tag{4.8}$$

où c_{max} est le nombre maximum de noeuds que peut contenir un cluster et $|c_i|$ le nombre de noeuds présents dans le cluster i.

Comme souvent avec les problèmes d'optimisation sur les entiers, il est très compliqué de trouver une solution optimale sans avoir à parcourir toutes les combinaisons possibles. Le problème est *NP-dur* et des solutions alternatives doivent être trouvées.

On propose l'algorithme suivant pour résoudre le problème de partitionnement. Pour un noeud n possédant Z noeuds enfants. On souhaite faire une partition des Z noeuds enfants en Q clusters homogènes. A l'initialisation, les Z enfants sont repartis aléatoirement entre les Q clusters de façon équilibrée. Ensuite, on itère le processus suivant : pour chaque noeud enfant z et chaque cluster q, on entraîne un classifieur $f_{\{s_z,q\}}$ à séparer les exemples du noeud enfant des exemples du cluster[2]. Soit $e(z, q)$ l'erreur obtenue. Une fois cette étape terminée, on reconstruit le partitionnement en affectant chaque noeud enfant au cluster avec lequel il a fait le plus d'erreurs.

Ce processus itératif est décrit dans Algorithme 6.

2. Lorsque l'on parle des exemples d'un noeud, on parle des exemples des classes présents dans ce noeud. De la même façon, les exemples d'un cluster correspondent aux exemples des classes de l'ensemble des noeuds présents dans ce cluster.

La procédure prend fin lorsque le partitionnement est stable, ou lorsque l'on atteint le nombre d'itérations maximum autorisé [3].

Algorithm 6 Partitionnement des Nœuds

Entrée: n (index du noeud où partitionner)
Entrée: $Z = |\mathcal{E}_n|$ (le nombre d'enfants du noeud n)
Entrée: Q (le nombre de clusters à construire)
  Initialisation aléatoire des Z nœuds en Q clusters : $\mathcal{C}$
  tant que Clusters non stable **faire**
    pour chaque nœud fils $z \in Fils(n)$ **faire**
      pour chaque cluster $q \in \mathcal{C}$ **faire**
        Entraîner les classifieurs : $f_{\{z,q\}}$
      fin pour
    fin pour
    Ré-initialiser le clustering de $\mathcal{C}$: $\forall q \in \mathcal{C}, q = \emptyset$
    pour chaque nœud fils $z \in Fils(n)$ **faire**
      Assigner le nœud z au cluster $\arg\max_q e(z, q)$
    fin pour
  fin tant que
  retourner $\mathcal{C}$

Dans le cas où le temps d'apprentissage est trop important, il est possible de travailler à partir de *classifieurs une-classe*, ce qui permet de diminuer d'un ordre de grandeur le nombre de classifieurs à apprendre. Expérimentalement, l'utilisation de ce type de classifieurs a montré des résultats proches de ceux utilisant des classifieurs binaires classiques bien que plus variables et un peu moins robustes. Le choix du type de classifieur (binaire ou *une-classe*) dépendra ainsi des contraintes du problème vis-à-vis du temps d'apprentissage.

4.4 Expérimentations

4.4.1 Protocole

Nous avons comparé notre méthode avec plusieurs approches permettant d'offrir une réduction du temps de classification.

Tout d'abord, nous avons comparé notre modèle avec deux méthodes de l'état de l'art dans la construction de hiérarchies de classification. La première utilise une approche

3. En pratique, le nombre d'itérations nécessaire est très faible pour atteindre une quasi-stabilité des clusters. On peut alors définir une limite à partir de laquelle on considère les clusters comme suffisamment stables.

de partitionnement utilisant du *clustering spectral* (présentée précédemment sous-section 2.4.2.1 [Bengio et al., 2010]). Pour rappel, une matrice de confusion est calculée grâce à un ensemble de classifieurs OAA sur un ensemble de validation. Ensuite, un partitionnement est effectué récursivement sur les classes des nœuds de l'arbre en commençant par le nœud racine. La seconde méthode comparée est une approche construisant une hiérarchie sur deux niveaux [Weston et al., 2013] via l'utilisation de l'algorithme des K-moyennes appliqué directement aux exemples.

Ensuite, nous avons introduit dans notre protocole deux méthodes classiques de codes correcteurs d'erreurs (*ECOC*) [Allwein et al., 2001; Dietterich and Bakiri, 1995] avec deux procédures de décodage différentes (distance de Hamming et *LossBased* (2.3.3) [Passerini et al., 2004]). Pour ces deux méthodes ECOC, cinq milles matrices de codes pour les classes ont été tirées aléatoirement puis celles maximisant les distances de Hamming entre les codes des classes ont été gardées.

Finalement, nous avons calculé les taux de bonne classification d'un classifieur multi-classes OAA . Nous avons aussi construit un modèle hiérarchique basé sur l'ontologie de classes lorsqu'elle était fournie avec les données.

Pour toutes ces méthodes, la famille de classifieurs simples choisie est la même pour toutes les méthodes. Nous avons choisi des machines à vecteurs de supports ou SVM (pour *Support Vector Machine*). Nous avons choisi des noyaux linaires car les tests ont été réalisés sur du texte et la séparation linéaire est bien adaptée à des vecteurs creux en grande dimension. Cette famille de modèles a montré qu'elle était une méthode de choix pour ce type de tâche [Cai and Hofmann, 2004; Liu et al., 2005a; Yang et al., 2003; Dumais and Chen, 2000; Zhou and Xiao, 2011; Weston et al., 2013; Hao et al., 2007]. L'implémentation des SVM linéaires utilisés vient de *scikit learn library* et les paramètres ont été régularisés par cross-validation.

Comme le principal but est d'observer la capacité de notre méthode à accélérer le processus de classification des exemples, nous allons comparer le taux de bonne classification de chaque méthode en fonction du ratio de complexité r_{x} atteint par cette méthode. On rappelle que cette mesure calcule la fraction du nombre moyen de classifieurs utilisés pour la classification par exemple, par rapport au nombre de classifieurs qu'aurait utilisé un schéma de classification OAA pour la même tâche. Ainsi, un ratio de complexité de

1 signifie que le modèle est aussi lent qu'un modèle OAA. Un ratio de complexité inférieur à 1 signifie que le modèle est plus rapide pour classifier un nouvel exemple qu'un classifieur OAA.

Étant donnée que notre méthode vise seulement un gain de vitesse de classification de type algorithmique, nous n'avons pas utilisé de technique visant à réduire l'espace dimensionnel des exemples par des méthodes de sélection de caractéristiques. Bien entendu, ces méthodes de réduction de dimension peuvent être utilisées conjointement à notre modèle.

Pour chaque modèle, nous avons calculé la précision pour différentes valeurs de ratio de complexité. Pour les modèles hiérarchiques *gourmands*, il suffit de changer le nombre d'enfants par nœud. Pour les modèles de codes correcteurs d'erreur, il suffit de changer la longueur des codes. Notre modèle est très souple de ce côté-là car il suffit de stopper l'algorithme à n'importe quel moment pour atteindre un certain ratio de complexité.

4.4.2 Résultats

Les tableaux 4.1 et 4.2 montrent les précisions obtenues pour différents ratios de complexité sur les données DMOZ et sur les données Sector. Notre modèle est capable d'obtenir des résultats légèrement supérieurs à la référence qu'est le modèle OAA tout en permettant une classification 55 fois plus rapide. Les autres méthodes produisant une classification aussi rapidement ne sont pas capable d'atteindre des précisions similaires. En réduisant le ratio de complexité, donc en autorisant moins de temps de calcul pour une prédiction, on voit que les performances de toutes les méthodes diminuent rapidement (les colonnes où $r_{\mathrm{x}} = 0.012$ et $r_{\mathrm{x}} = 0.0081$ du tableau 4.1 correspondent aux versions des modèles les plus rapides). Cependant, notre modèle continue de produire de meilleures classifications que les autres méthodes avec une confortable marge.

Les résultats venant de la méthode de Weston et al. [2013] obtenus sur nos bases de données sont bien en deçà des résultats des autres méthodes. Cela s'explique par la première étape qui consiste à faire un clustering des exemples. Si les clusters sont de taille trop différente, cela handicape fortement les performances étant donné que chaque cluster est ensuite associé à un même nombre de classes. Ainsi, les gros cluster sont susceptibles de ne pas être associés à suffisamment de classes, ce qui entraînera irrémédiablement

des erreurs de classification à la fin du processus de classification. Les résultats de cette méthode n'ont ainsi pas été reportés dans les tableaux.

Nous avons inclus dans le tableau 4.1 les performances d'un modèle hiérarchique utilisant une ontologie des classes fournie avec les données DMOZ.

Chaque colonne donne les performances d'un modèle pour un ratios de complexité donné. Si le modèle n'est pas capable de produire une classification pour un ratio de complexité donné, un tiret (−) est inséré à la place dans la cellule correspondante. Par exemple, un schéma de classification OAA n'est pas capable de produire une classification pour un ratio de complexité autre que 1. Le graphique Figure 4.12 montre l'évolution du taux de bonne classification pour différents modèles en fonction du ratio de complexité.

Modèles	Type	Acc r_x@1	r_x@0.018)	r_x@0.012	r_x@0.0081
OAA	Plat	38.28%	-	-	-
Ontologie Originale	Arbre	-	-	-	*38.05%*
Notre Modèle	Arbre	-	**38.58%**	**36.45%**	**33.71%**
Clustering Spectral	Arbre	-	32.83%	32.78%	31.37%
ECOC-Hamming	Plat	-	29.61%	25.06%	21.19%
ECOC-LossBased	Plat	-	34.17%	32.08%	30.55%

TABLE 4.1: Taux de bonne classification de différents modèles sur le dataset de 13000 classes de DMOZ. Nous avons reporté en police gras le meilleur résultat significatif pour chaque accélération. "r_x@" donne le ratio de complexité d'une colonne. Le caractère tiret (−) signifie que le modèle n'est pas capable de classifier avec ce ratio de complexité. Clustering Spectral décrit par Bengio et al. [2010], ECOC-LossBased décrit par Allwein et al. [2001].

Modèles	Type	Acc r_x@1	r_x@0.2	r_x@0.14
OAA	Plat	**94.1%**	-	-
Notre modèle	Arbre	-	**93.7%**	**90.4%**
Hiérarchique - Clustering Spectral	Arbre	-	88.5%	88.2%
ECOC - Distance de Hamming	Plat	-	27.7%	18.3%
ECOC - LossBased	Plat	-	89.6%	73.5%

TABLE 4.2: Taux de bonne classification sur le dataset Sector avec 105 classes. Les résultats en gras sont les meilleurs résultats significativement supérieurs aux autres pour un même ratio de complexité.

4.4.3 Discussion

Il est intéressant de noter que les modèles de type ECOC souffrent plus lorsqu'on les pousse à atteindre les ratio de complexité les plus extrêmes comparativement aux modèles

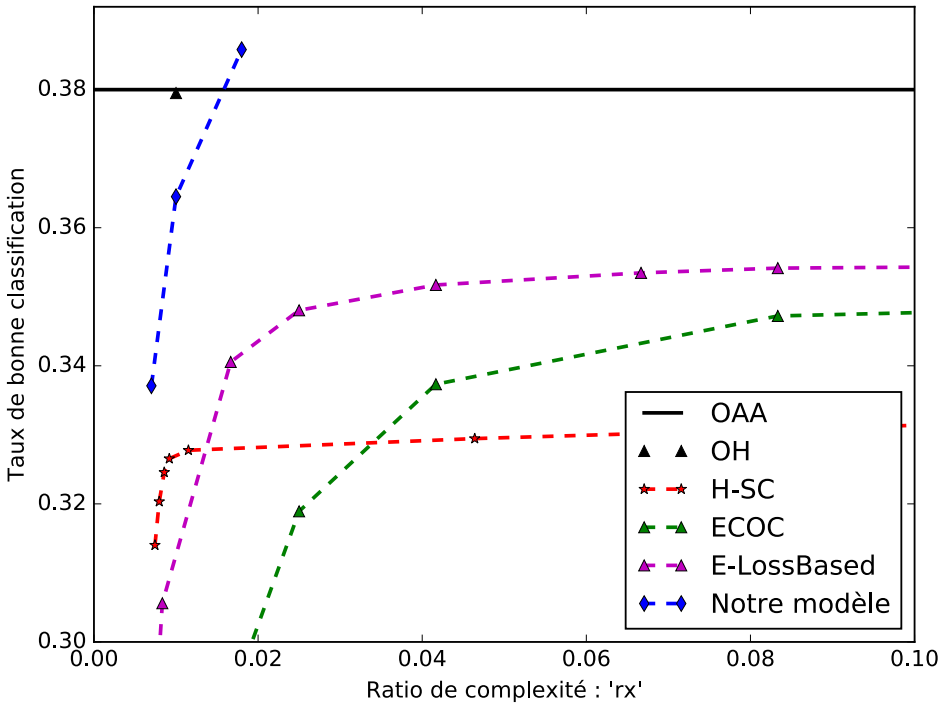


FIGURE 4.12: Taux de bonne classification sur le dataset DMOZ en fonction du ratio de complexité. La ligne noire est une extension du point du modèle OAA ($r_{\mathrm{x}} = 1$). Notre modèle permet de construire des hiérarchies *plus lentes* que $r_{\mathrm{x}} = 0.018$ car dès la première itération de l'algorithme, le gain en complexité est substantiel. Ainsi, nous ne pouvons pas nous comparer aux autres méthodes pour de trop grandes valeurs de r_{x}. OH correspond au modèle hiérarchique utilisant la hiérarchie de classes originale. H-SC correspond au modèle hiérarchique par clustering spectral. E-Dense correspond au modèle ECOC avec codage classique dense. E-LossBased correspond au modèle ECOC utilisant des fonctions de perte pour décoder le signal.

hiérarchiques. Les tendances s'inversent lorsque la contrainte sur le ratio de complexité est un peu relâchée.

Cela peut être expliqué par la propriété naturelle des modèles hiérarchiques à atteindre facilement une forte réduction du nombre de classifieurs nécessaire au processus de classification. Avec ces modèles, les classifieurs se spécialisent suivant leur emplacement dans la hiérarchie. Les modèles de type ECOC produisent des classifieurs qui ont tous à peu près la même importance pour le processus de classification. Ainsi, en réduisant fortement le nombre de classifieurs à leur disposition, il apparaît rapidement une diminution des performances.

Le modèle proposé ici est capable d'obtenir de meilleures performances que des modèles à *plat* plus lents. Notons que durant le challenge ([Partalas et al., 2015]) utilisant les

données DMOZ, l'utilisation d'informations sur l'organisation hiérarchique des classes était fournie.

La base de donnée Sector est un peu différente de DMOZ car elle ne possède pas une organisation naturelle des classes. Cela ne signifie pas forcément qu'il n'existe pas de hiérarchie latente. Cependant, il apparaît que trouver une organisation hiérarchique des classes de ce problème est moins évident. Notre modèle a néanmoins obtenu de meilleures performances que les méthodes de l'état de l'art.

Une étude plus en détail des erreurs par niveau pendant le déroulement de l'algorithme de partitionnement montre plus précisément le fonctionnement de notre modèle. La figure 4.13 montre l'évolution du taux de bonne classification à deux niveaux de la hiérarchie pendant le partitionnement de 12294 classes sur DMOZ. Il peut être noté que l'algorithme dégrade le pourcentage de bonne classification au second niveau en regroupant les classes qui sont difficilement séparables entre elles. Ce faisant, les partitions sont plus cohérentes et le pourcentage de bonne classification au niveau supérieur augmente. La précision globale sur l'ensemble de l'arbre est améliorée comme l'atteste les résultats des expérimentations. Cette observation peut s'expliquer par le principe du maillon faible : c'est le niveau qui fait le plus d'erreurs localement qui impacte le plus la précision finale du modèle. En améliorant la précision des *maillons faibles* aux dépend des maillons plus forts, le modèle final est globalement plus performant.

4.4.4 Étude de la hiérarchie obtenue

Nous avons comparé la hiérarchie obtenue avec la hiérarchie de classe originale pour la base de donnée DMOZ. Une façon de faire cela est de comparer la distance entre les classes dans chacune des hiérarchies [Dellschaft and Staab, 2006]. Pour avoir une référence, nous avons fait la même comparaison avec la hiérarchie obtenue via la méthode de clustering spectral.

Nous avons défini une fonction de voisinage $\mathcal{N}$ qui prend une classe ℓ et un arbre de classes T et qui retourne le voisinage de cette classe dans cette hiérarchie selon une mesure Δ et un rayon θ : $\mathcal{N}_\theta(\ell^*, T) = \bigcup_{\{\ell | \Delta_T(\ell^*, \ell) \leq \theta\}} \ell$. La mesure Δ utilisée ici calcule une distance d'arbre correspondant au nombre de noeuds du parcours allant de la feuille ℓ à une feuille ℓ'.

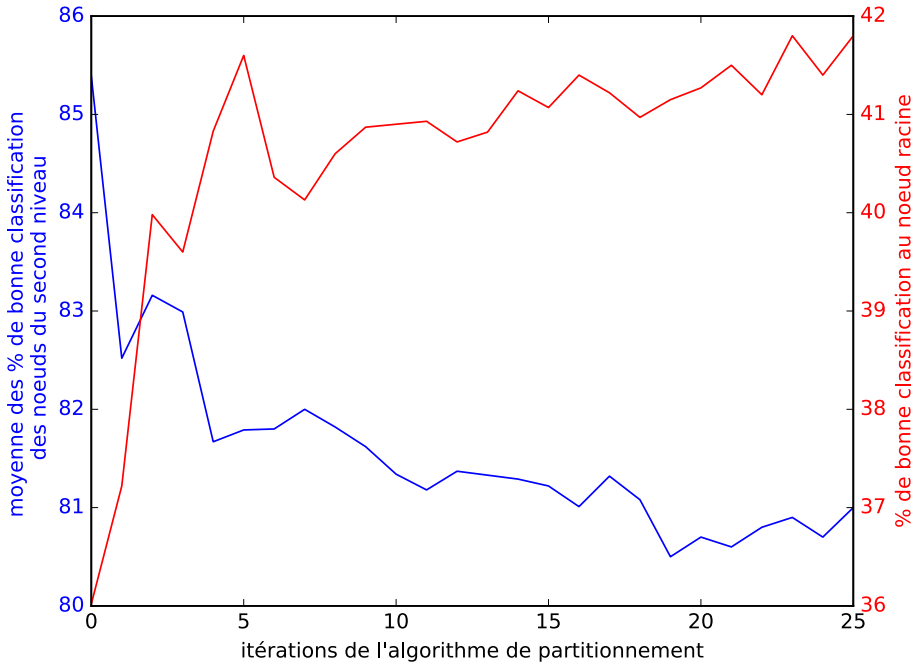


FIGURE 4.13: Exemple du comportement des pourcentages de bonne classification durant le processus de l'algorithme de partitionnement. Ces deux courbes correspondent aux pourcentages de bonne classification à deux différents niveaux de la hiérarchie lors du partitionnement de 12294 classes en 100 clusters : le niveau *haut* (en rouge) correspond au nœud racine et le second niveau correspond aux noeuds de l'ensemble des noeuds fils du noeud racine. Le taux de bonne classification affiché correspond à la moyenne des taux de bonne classification des classifieurs de ce niveau.

Nous avons ensuite comparé les voisinages obtenus des différentes hiérarchies pour chaque classe. Considérons une classe ℓ'. Nous avons calculé le voisinage θ de ℓ' dans la hiérarchie originale. Puis, nous définissons un autre rayon θ'. Nous calculons le voisinage de rayon θ' de ℓ' dans la hiérarchie à comparer avec la hiérarchie originale. Nous pouvons alors remplir le tableau suivant :

	$\mathcal{N}_{\theta'}$	$\overline{\mathcal{N}_{\theta'}}$
$\mathcal{N}_{\theta}$	$\lvert\mathcal{N}_{\theta'} \cap \mathcal{N}_{\theta}\rvert$	$\lvert\mathcal{N}_{\theta'}\rvert - \lvert\mathcal{N}_{\theta'} \cap \mathcal{N}_{\theta}\rvert$
$\overline{\mathcal{N}_{\theta}}$	$\lvert\mathcal{N}_{\theta}\rvert - \lvert\mathcal{N}_{\theta'} \cap \mathcal{N}_{\theta}\rvert$	$K - \lvert\mathcal{N}_{\theta'} \cap \mathcal{N}_{\theta}\rvert$

avec K le nombre de classes. Dans le cadre de la tâche visant à prédire l'ensemble $\mathcal{N}_{\theta'}$, on a :

$\lvert\mathcal{N}_{\theta'} \cap \mathcal{N}_{\theta}\rvert$: # vrais positifs (VP)

$\lvert\mathcal{N}_{\theta'}\rvert - \lvert\mathcal{N}_{\theta'} \cap \mathcal{N}_{\theta}\rvert$: # faux positifs (FP)

$\lvert\mathcal{N}_{\theta}\rvert - \lvert\mathcal{N}_{\theta'} \cap \mathcal{N}_{\theta}\rvert$: # faux négatifs (FN)

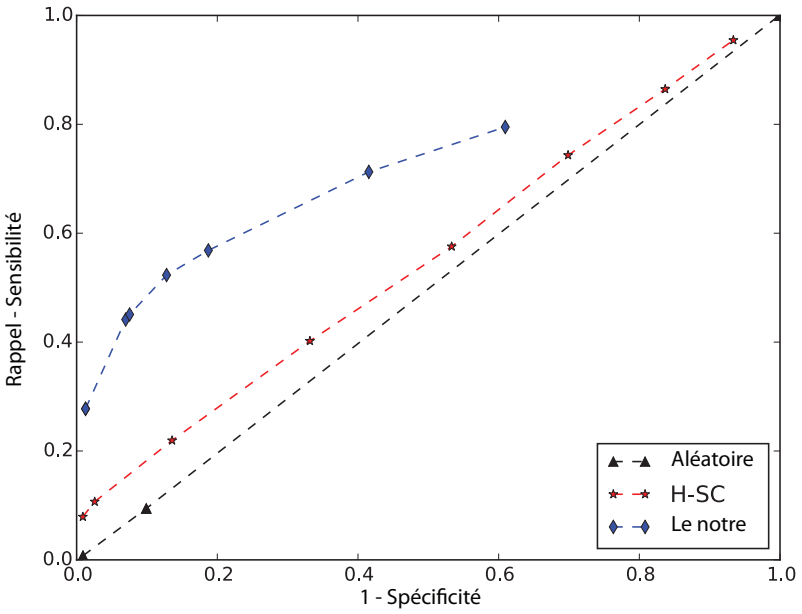


FIGURE 4.14: Comparaison des hiérarchies de classes produite par différents modèles comparées à la hiérarchie originale fournie avec les données DMOZ. Plus la précision est haute, plus la hiérarchie produite est proche de l'originale. H-SC : hiérarchie obtenue par clustering spectral.

$K - |\mathcal{N}_{\theta'} \cap \mathcal{N}_\theta|$: # vrais négatifs (VN)

Notons T_o l'arbre de classe basé sur l'ontologie de classes originale. On peut calculer une *sensibilité* ($\frac{VP}{VP+FN}$) :

$$\mathcal{S}_{\theta'}(T) = \frac{1}{K} \sum_{\ell} \frac{|\mathcal{N}_{\theta'}(\ell, T) \cap \mathcal{N}_\theta(\ell, T_o)|}{|\mathcal{N}_\theta(\ell, T_o)|}$$

On peut calculer de la même façon la *spécificité* ($\frac{VN}{VN+FP}$) et le *rappel* ($\frac{VP}{VP+FP}$).

En faisant varier le seuil θ', on change la taille du voisinage $\mathcal{N}_{\theta'}$. On obtient alors les courbes Figure 4.14 et Figure 4.15. Ces courbes ont été obtenues en choisissant un seuil pour θ de 3 et en tirant aléatoirement 1000 classes pour réduire le temps (qui est quadratique en fonction du nombre de classes) nécessaire au calcul de chaque distance classe à classe dans une hiérarchie.

Ces graphes montrent la capacité de notre modèle à retrouver partiellement une information d'organisation hiérarchique qui lui avait été cachée. Les méthodes standards de création de modèles hiérarchiques sont moins performants pour retrouver cette information, que la notre.

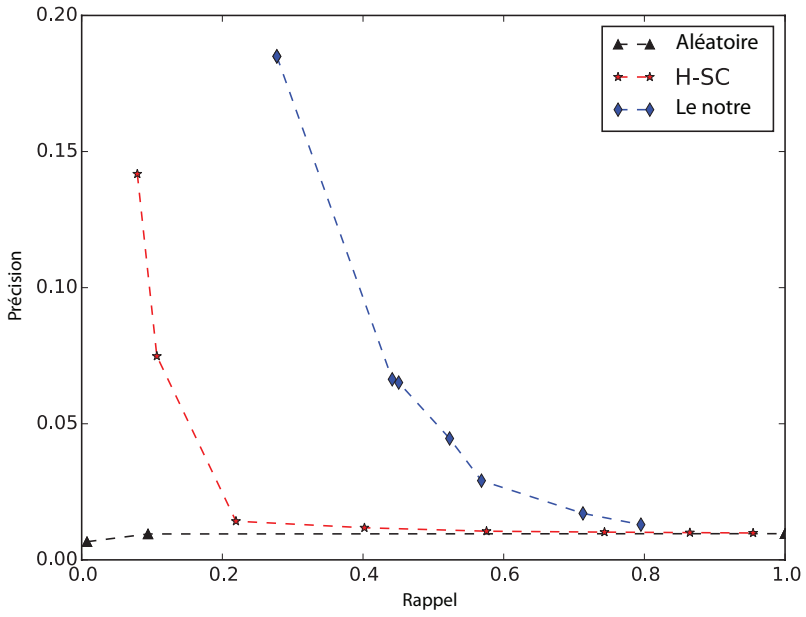


FIGURE 4.15: Comparaison des hiérarchies de classes produites par différents modèles avec la hiérarchie originale fournie avec le dataset DMOZ. Plus la *spécificité* est grande, plus la hiérarchie produite est proche de l'originale.

4.5 Conclusion

La méthode présentée décrit un nouvel algorithme d'apprentissage d'un modèle hiérarchique pour les problèmes de classification avec un grand nombre de classes. En commençant par un arbre *plat* avec un nombre important de feuilles, notre algorithme sélectionne itérativement les nœuds à développer dans le but d'optimiser le compromis complexité-performance de la classification d'un nouvel exemple par la création de niveaux intermédiaires dans la hiérarchie. Ainsi, l'espace des hiérarchies exploré est plus large et plus flexible qu'avec des méthodes gourmandes classiques.

Ce modèle est générique et permet d'utiliser n'importe quelle famille de classifieur binaire. C'est un facteur important car selon les problèmes et le type de données, il peut être important d'adapter la nature des classifieurs. De plus, comme le critère de similarité utilisé lors du partitionnement est basé sur l'erreur de ces mêmes classifieurs, la tâche de partitionnement est toujours en lien avec la tâche de classification et la nature des classifieurs.

La création de cette hiérarchie de classes s'inscrit simplement dans le formalisme des schémas de classification vu précédemment. La hiérarchie de classes produite fournit

immédiatement la liste des classifieurs simples à apprendre ($f_{\mathcal{S}}$). La séquence d'utilisation des classifieurs est donnée par la hiérarchie des classes qui peut alors être assimilée à une hiérarchie de classifieurs ($b_{\mathcal{S}}$).

Tout comme les autres modèles hiérarchiques, il n'y a pas de *mémorisation* des résultats obtenus par tous les classifieurs pendant la descente d'un exemple : il est seulement nécessaire de retenir le dernier classifieur utilisé ainsi que sa réponse ($u_{\mathcal{S}}$). La prédiction finale est obtenue dès lors qu'une feuille est atteinte.

Chapitre 5

Méthodes Ensemblistes : Modèle probabiliste séquentiel actif

5.1 Introduction

Comme vu précédemment, les méthodes basées sur l'organisation hiérarchique de classifieurs permettent d'accélérer considérablement les temps de classification d'un nouvel exemple. Une des principales forces des modèles hiérarchiques vient de la spécificité du chemin de classifieurs utilisé par l'exemple, qui se traduit par un processus séquentiel particulier qui va *suivre* l'ordre des classifieurs dans la hiérarchie.

Tous les exemples commencent le processus de classification au même nœud racine, mais ils vont emprunter des chemins différents suivant les réponses des classifieurs. De par ce mécanisme, ils vont bénéficier de classifieurs plus adaptés et plus spécifiques au fur et à mesure du processus de classification. Un point faible de ces modèles tient au fait que certains problèmes ne permettent pas une *bonne* hiérarchisation des classes. Cela arrive quand toutes les classes sont autant éloignées les unes des autres, et qu'il n'existe pas d'organisation hiérarchique des classes. Dans ce cas, l'organisation forcée des classes en un arbre de classes peut entraîner des erreurs de classification.

Les modèles ensemblistes de type Code Correcteur d'Erreurs (ECOC) sont plus flexibles que les méthodes hiérarchiques car ils ne cherchent pas à extraire une organisation hiérarchique pré-existante des classes du problème. Un ensemble de classifieurs simples est utilisé *à plat* et l'ensemble des réponses de ces classifieurs simples est agrégé pour former une

prédiction complexe. Ainsi, la classification est plus robuste car l'erreur d'un classifieur simple peut être rattrapée par les autres classifieurs simples. Cependant, contrairement aux modèles hiérarchiques, les modèles ensemblistes ne sont pas capables de proposer une classification spécifique à un exemple donné. Un ECOC utilise tous les classifieurs pour tous les exemples. Il n'y a pas de *spécialisation* du processus de classification.

Le travail qui va être présenté dans ce chapitre est basé sur la volonté de proposer un modèle robuste comme un modèle ECOC tout en proposant un processus de décision spécifique pour chaque exemple. L'idée est de combiner l'aspect ensembliste (agrégation des réponses d'un ensemble de classifieurs) avec l'aspect hiérarchique (sélection séquentielle des classifieurs). De façon plus formalisée, l'idée est de définir un processus de décision séquentiel qui va être capable de sélectionner judicieusement les classifieurs les plus utiles pour l'exemple en cours de classification en fonction des résultats des précédentes classifications.

5.2 Processus de décision dynamique pour la classification

Les algorithmes que nous avons étudiés jusqu'ici ont des schémas de classifications que l'on peut qualifier de *statiques* au sens où la séquence de classifieurs auquel un exemple va être confronté est fixée à l'avance : c'est le cas dans les méthodes hiérarchiques où l'enchaînement entre classifieurs est modélisé par un arbre ; c'est le cas également dans les méthodes ensemblistes telles que les ECOC ou le un contre tous, où tous les classifieurs sont utilisés à chaque fois comme le montre le diagramme 5.1. Ainsi, deux exemples différents vont agréger des réponses des mêmes classifieurs. Dans le cadre classique de classification multi-classe avec un nombre restreint de classes, ceci n'est pas un problème. Cependant, lorsque le modèle a pour but de passer à l'échelle avec des problèmes de plus de 10000 classes, il est important d'être parcimonieux avec l'utilisation des ressources lors du processus de classification. Par exemple, si un classifieur simple a clairement signalé que le document parle d'animaux, il est judicieux de se concentrer sur les types d'animaux plutôt que de perdre du temps avec des dichotomies ne permettant pas de mieux séparer les animaux entre eux [Koller and Sahami, 1997]. Cette remarque explique la difficulté d'appliquer des méthodes ensemblistes de type ECOC aux problèmes avec un grand nombre de classes, en particulier lorsque les contraintes sur le nombre de classifieurs à utiliser sont trop fortes.

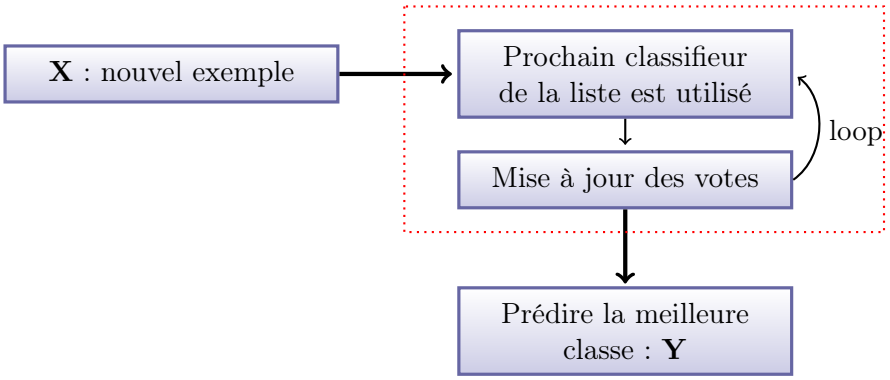


FIGURE 5.1: Processus global de classification d'une méthode ensembliste.

5.2.1 Principe du processus de décision

Si on change de point de vue et qu'au lieu de focaliser sur l'exemple, on regarde chacune des classes individuellement pendant la classification d'un exemple, l'action "*utiliser un classifieur binaire*" révèle une information vis-à-vis de l'appartenance de l'exemple aux différentes classes : en effet, utiliser un classifieur qui départage deux sous-ensembles de classes permet de donner un vote positif à chaque classe de l'ensemble des classes reconnues et un vote négatif à chaque classe du deuxième sous-ensemble. Cette information est cependant bruitée : d'une part du fait de l'imprécision du classifieur, d'autre part elle n'est valable que si l'exemple appartient bien à un des deux sous-ensembles de classes. Si l'exemple n'appartient à aucune classe utilisée pour apprendre le classifieur, la réponse de ce dernier sera aléatoire, en fonction de corrélations de l'exemple avec les autres classes.

Dans le cadre du formalisme explicité en section 3, notre objectif est de proposer un schéma de classification tel que la politique de sélection du prochain classifieur soit dynamique. Formalisé de la sorte, le problème consiste à explorer au mieux les classifieurs disponibles pour réussir à identifier la bonne classe en un nombre minimum d'itérations de l'algorithme d'exploration. Le travail qui est décrit par la suite se focalise sur la politique de sélection de classifieurs simples afin d'optimiser le nombre de classifieurs utilisés, dans le but d'obtenir le plus rapidement possible une bonne classification. Dans le présent travail, on ne s'interessera pas à la création de l'ensemble des dichotomies formant l'ensemble des classifieurs à notre disposition, seule l'utilisation appropriée de ces ressources nous interesse ici.

Nous considérerons comme mémoire du processus l'agrégation des votes obtenus pour

	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6	ℓ_7	**x**
h_1	1	1	-1	1	-1	1	-1	1
h_2	-1	-1	1	1	-1	1	1	-
h_3	1	-1	0	-1	1	-1	1	-1
h_4	-1	1	0	1	-1	0	1	-
h_5	1	0	-1	0	-1	0	-1	-
h_6	0	-1	0	-1	0	1	0	-1
h_7	1	1	0	0	1	0	-1	-
h_8	-1	1	0	0	0	0	1	-
h_9	0	-1	0	1	0	0	0	1
h_{10}	1	0	-1	0	0	0	0	-
Score	0.5	0.75	0	1	0	0.33	0	ℓ_4
Nombre	2	4	1	4	2	3	2	

FIGURE 5.2: Illustration de l'algorithme de décision séquentielle. **A gauche :** la matrice de codage des classifieurs binaires h_i. Chaque ligne code la dichotomie des classes (ℓ_k) du problème : les exemples des classes annotées 1 sont labellisés positivement et les exemples des classes annotées -1 sont labellisés négativement pour l'apprentissage du classifieur de la ligne. **A droite :** un exemple **x** à classifier et les réponses des 4 classifieurs h_1, h_3, h_6 et h_9) utilisés par l'algorithme séquentiel. La ligne **score** représente le score pour chaque classe vis-à-vis de la tâche de classification de l'exemple **x** et la ligne **nombre** le nombre de fois où chaque classe a été considérée. La dernière ligne de la colonne **x** représente la décision du classifieur global.

chaque classe au cours des itérations, normalisée entre 0 et 1 afin de dénoter une probabilité d'appartenance, ainsi que le nombre de fois où chaque classe a été considérée. À chaque itération de l'algorithme, la politique sélectionne le prochain classifieur à utiliser en fonction des scores et du nombre de votes pour chaque classe. Le processus de mise à jour de la mémoire modifie les votes selon la sortie du classifieur sélectionné sur l'exemple considéré comme dans le cas d'une méthode ensembliste classique.

Un exemple du processus de classification est décrit Figure 5.2. Cet exemple montre comment, à partir d'un ensemble de classifieurs $\{h_i\}_{i\in 1\ldots 10}$ pré-appris, l'algorithme considère itérativement certains de ces classifieurs pour classifier l'exemple **x**. Le processus de sélection des classifieurs sera explicité dans les prochaines sections, il faut retenir ici que ce processus va chercher à sélectionner les classifieurs qui vont apporter le plus d'informations pour la classification de l'exemple en cours.

Sur l'exemple de la figure 5.2, après avoir considéré trois classifieurs h_1, h_3 et h_6 pour l'exemple **x** donné, les classes de plus haut score sont les classe ℓ_2 et ℓ_4 (3 votes positifs sur 3 votes, soit un score normalisé de 1). L'utilisation du classifieur h_9 permet de désambiguïser entre ces deux classes et attribuer ainsi un score respectivement de 0.75 pour

la classe ℓ_2 (3 votes positifs sur 4) et de 1 pour la classe ℓ_4. À la fin de la procédure, la prédiction finale est donnée par la classe qui a le plus haut score : ℓ_4.

Un objectif supplémentaire de la politique consiste à sélectionner progressivement des classifieurs portant sur de moins en moins de classes. En effet, un classifieur dont le problème de classification est basé sur un code *parcimonieux*, qui ne fait intervenir que quelques classes parmi toutes celles disponibles, sera en moyenne plus performant pour résoudre son problème de classification qu'un classifieur ayant à départager un plus grand nombre de classes [Even-Zohar and Roth, 2001]. L'information obtenue sera donc de meilleure qualité si la dichotomie utilisée concerne un faible nombre de classes.

En résumé, une bonne politique doit choisir au fur et à mesure des itérations des classifieurs qui départagent les classes les plus probables (avec le meilleur score), tout en s'assurant avoir acquis une information fiable sur l'ensemble des classes (nombre de votes pour chaque classe) et que les classifieurs choisis soient de plus en plus spécifiques (élimination des classes les moins probables).

5.2.2 Formalisation

Comme précédemment nous désignerons par $\mathcal{L} = \{\ell_1, \ldots, \ell_K\}$ l'ensemble des K classes du problème de classification, $\mathcal{D} = \{(x_i, y_i) \in \mathbb{R}^D \times \mathcal{L}\}$ un ensemble d'apprentissage de ce problème, $\Phi \in \{-1, 0, 1\}^K$ un ensemble de dichotomies sur cet ensemble de classes, $\mathbf{c} = (c_1, \ldots, c_K) \in \Phi$ une dichotomie et $h_{\mathbf{c}} : \mathbb{R}^D \to \{-1, 1\}$ le classifieur correspondant entraîné à séparer les classes $\{\ell_i | c_i = 1\}$ des classes $\{\ell_i | c_i = -1\}$. Nous noterons également $|\mathbf{c}| = \sum_{i=1}^{K} |c_i|$ la norme 1 du vecteur $\mathbf{c}$. Cette quantité dénote le nombre de classes *encodées* par la dichotomie.

Afin de formaliser la mémoire du processus de classification, nous allons définir une notion de récompense associée à chaque dichotomie pour un exemple $\mathbf{x}$ donné. Rappelons ici qu'une dichotomie $\mathbf{c}$ procure une information pour une classe ℓ_i uniquement si la dichotomie encode la classe, c'est-à-dire si $c_i \neq 0$. Dans le cas inverse, l'information délivrée par le classifieur est complètement aléatoire et non informative pour cette classe. Si le résultat de la classification $h_{\mathbf{c}}(x)$ est 1, l'ensemble des classes$\{\ell_i | c_i = 1\}$ reçoit une récompense positive et l'ensemble $\{\ell_i | c_i = -1\}$ une récompense nulle ; si le résultat de la classification est -1, les récompenses sont inversées. Si $c_i = 0$, aucun vote ne sera

comptabilisé pour cette classe lors de l'utilisation de cette dichotomie. Formellement, la récompense pour une classe ℓ_i est définie de la manière suivante :

$$r(\mathbf{c}, \ell_i, x) = \begin{cases} 1 & \text{si } h_{\mathbf{c}}(x) = c_i \quad \text{(vote positif)} \\ 0 & \text{si } h_{\mathbf{c}}(x) = -c_i \quad \text{(vote négatif)} \\ 0 & \text{si } c_i = 0 \quad \text{(pas de vote)} \end{cases} \text{, soit } \; r(\mathbf{c}, \ell_i, x) = |c_i|(1+c_i.h_{\mathbf{c}}(x))/2$$

Étant donné un exemple $\mathbf{x}$ à classer, la mémoire du processus à un temps t est composée de deux vecteurs $\boldsymbol{\mu}^t = (\mu_{1,t}, \ldots, \mu_{K,t}) \in [0,1]^K$ et $T^t = (T_{1,t}, \ldots, T_{K,t}) \in \mathbb{N}^K$ qui dénotent respectivement la moyenne des récompenses obtenues pour chaque classe et le nombre de fois où chaque classe a été considérée par le processus jusqu'au temps t. Supposons que les dichotomies $\mathbf{c}^1, \ldots, \mathbf{c}^{t-1}$ ont été considérées préalablement, le processus de mise-à-jour de la mémoire pour la classe ℓ_i au temps t dépend de la récompense $r(\mathbf{c}^t, \ell_i, \mathbf{x})$ et se fait de la manière suivante :

- $T_{i,t} = \sum_{j=1}^{t} |c_i^j| = T_{i,t-1} + |c_i^t|$
- $\mu_{i,t} = \frac{1}{T_{i,t}} \sum_{j=1}^{t} r(\mathbf{c}^j, \ell_i, \mathbf{x}) = \mu_{i,t-1} + \frac{r(\mathbf{c}^t, \ell_i, \mathbf{x}) - \mu_{i,t-1}}{T_{i,t}}$

On remarque que l'état mémoire n'est changé que dans le cas où la dichotomie considérée encode la classe ($c_i \neq 0$).

5.2.3 Point de vue probabiliste

Le score μ_i de chaque classe peut être considéré comme une variable aléatoire dont la réalisation dépend de l'exemple $\mathbf{x}$ à classifier. En effet, une récompense $r(\mathbf{c}, \ell_i, \mathbf{x})$ correspond au résultat de la classification de l'exemple par le classifieur $h_{\mathbf{c}}$. Dans le cas où l'exemple $\mathbf{x}$ appartient à une classe ℓ_i telle que $c_i \neq 0$, la récompense suit une loi de Bernoulli de paramètre $q_{\mathbf{c}} = \mathbb{P}_{\mathbf{x}}(c_i h_{\mathbf{c}}(\mathbf{x}) = 1 | x \in \ell_i, c_i \neq 0)$, le taux de bonne classification du classifieur $h_{\mathbf{c}}$. Dans le cas contraire où l'exemple appartient à une classe non encodée par la dichotomie ($c_i = 0$), la réponse du classifieur suit également une loi de Bernoulli mais de paramètre non explicite : en supposant les classes indépendantes les unes des autres, l'espérance de la récompense pour un exemple d'une classe non encodée par un classifieur devrait tendre vers $\frac{1}{2}$; en pratique, la récompense dépend fortement de la corrélation de la classe ℓ_i avec les classes encodées par le classifieur. Remarquons également que les réalisations des récompenses peuvent être considérées

comme indépendantes à partir du moment où les classifieurs sont entraînés de manière indépendante (par exemple sur un échantillonnage de la base d'apprentissage).

A l'itération t, le processus d'accumulation des scores pour la classe ℓ_i est ainsi une variable aléatoire correspondant à la somme pondérée de $T_{i,t}$ variables indépendantes suivant des lois de Bernoulli de paramètres différents. Nous allons nous intéresser par la suite aux quantités $\mu^*_{i,t} = \mathbb{E}_{\mathbf{x}}(\mu_{i,t}|\mathbf{x} \in \ell_i)$ qui dénotent l'espérance du score pour la classe i à l'itération t dans le cas où l'exemple est de la classe ℓ_i. Il en découle que

$$\mu^*_{i,t} = \frac{1}{T_{i,t}} \sum_{j=1}^{t} \mathbb{E}(r(\mathbf{c}^t, \ell_j, \mathbf{x})) = \frac{1}{T_{i,t}} \sum_{j=1}^{t} |c^j_i| q_{\mathbf{c}^t}$$

Cette espérance nous permet d'utiliser la borne de Hoeffding pour la variable aléatoire $\mu_{i,t}$, somme de variables aléatoires indépendantes suivant des lois de Bernoulli de paramètre $q_{\mathbf{c}^t}$:

$$\mathbb{P}[\mathbb{E}[\mu_{i,t}] - \mu_{i,t} \geq \delta | \mathbf{x} \in \ell_i] \leq e^{-T_{i,t}\delta^2}$$

soit

$$\mathbb{P}[\mu^*_{i,t} - \mu_{i,t} \geq \delta | \mathbf{x} \in \ell_i] \leq e^{-T_{i,t}\delta^2}$$

Ainsi, on est capable de borner la probabilité d'observer un score déviant de plus d'un δ du score médiant attendu, toujours sous l'hypothèse que cette classe est la bonne classe.

5.2.4 Similarité avec le contexte Bandits

Le cadre des Bandits Manchots [Chen et al., 2014a] est privilégié dans l'étude des processus où il s'agit d'identifier une action optimale en fonction des retours des actions précédemment effectuées. Le modèle classique le plus étudié consiste à choisir itérativement parmi K actions disponibles (aussi appelées bras, en référence aux machines à sous) et observer une récompense stochastique associée à l'action choisie. L'objectif est d'accumuler le plus de récompenses au fur et à mesure des itérations en identifiant le plus rapidement possible la meilleure action : il s'agit de résoudre à chaque itération le dilemme bien connu *exploration vs exploitation* [Auer, 2003], à savoir : vaut-il mieux se contenter de l'information acquise jusqu'ici et jouer le meilleur bras estimé (exploitation), ou mieux vaut-il explorer encore les récompenses attendues des bras et agréger de

l'information pour affiner l'estimation des récompenses quitte à perdre une petite part de récompense à cette itération.

Cette problématique a été très largement explorée ces dernières années dans de nombreux contextes et sous diverses formes avec des résultats théoriques de convergence très importants (voir par exemple [Kuleshov and Precup, 2014] pour une revue détaillée des dernières avancées algorithmiques). Parmi les travaux se rapprochant de notre contexte, on peut citer le problème d'*identification du meilleur bras* [Audibert and Bubeck, 2010; Kaufmann et al., 2014], qui, étant donné un seuil de confiance ou un nombre d'itérations maximal, cherche à identifier sous ces contraintes le bras associé à la meilleure récompense. L'idée de considérer une relation linéaire sur les récompenses associées aux bras ([Soare et al., 2014]) ou plus généralement le plongement des récompenses dans un espace topologique ([Bubeck et al., 2008]) sont également proches de notre contexte. Enfin, d'autres travaux considèrent le cas où plusieurs bras peuvent être actionnés en même temps [Chen et al., 2014b] afin d'identifier le meilleur bras ou un ensemble de bras optimaux ont également un lien avec nos travaux.

Tout comme dans le problème des Bandits Manchots, notre politique de sélection de classifieurs a besoin d'optimiser la prochaine action à entreprendre - choix d'une dichotomie - dans le but d'identifier au plus vite la classe la plus probable. Le choix d'une dichotomie peut être vu comme la sélection de deux sous-ensembles de classes et la récompense associée le retour du classifieur pour ces classes. Cependant, contrairement au cadre des bandits-manchots, à chaque itération nous observons toujours une récompense positive pour un certain nombre de classes et une récompense nulle pour d'autres classes. De plus, la récompense observée est très dépendante du nombre de classes encodées par la dichotomie sélectionnée. Ainsi, la récompense moyenne associée à chaque classe est non stationnaire au fur et à mesure des itérations, puisque les dichotomies considérées doivent porter sur un nombre de plus en plus faible de classes en fonction des itérations. Ces particularités nous ont guidé à proposer une politique hors du contexte des bandits manchots, inspirée toutefois de ce contexte, en privilégiant un des outils principaux utilisés que sont les inégalités de concentrations, en particulier la borne d'Hoeffding.

5.3 Algorithme MUSCA : Multi-class Sequential Classification

L'algorithme proposé (nommé *MUSCA* pour Multi-class Sequential Classification) suit le schéma de classification précédemment défini. Nous allons préciser dans cette section la politique considérée, une heuristique de sélection de classifieurs gloutonne qui optimise le rejet de classes à chaque itération. Au terme de T_{max} itérations, l'algorithme retourne la classe possédant le plus haut score, les scores étant simplement la moyenne des votes qu'a reçu chacune des classes.

L'idée de la méthode est de chercher à rejeter à chaque itération le plus de classes possibles. Cette élimination s'opère via un test d'hypothèse basé sur l'information agrégée par les classifieurs utilisés au préalable. Pour effectuer ces tâches, l'idée est d'estimer la probabilité d'appartenance de l'exemple $\mathbf{x}$ à chacune des classes du problème de classification. En étudiant la différence entre le score attendu d'une classe ℓ_i (sous l'hypothèse que ℓ_i soit la bonne classe) et son score observé, il est possible alors de réfuter l'hypothèse selon laquelle l'exemple $\mathbf{x}$ appartient à la classe ℓ_i.

Dans ce contexte de classification avec un grand nombre de classes, notre but est de réduire le plus vite l'ensemble des classes possibles pour pouvoir se concentrer sur un plus petit nombre de classes. Notre heuristique aura ainsi pour but de privilégier les dichotomies qui permettent de minimiser le vecteur de scores des classes, ce qui favorisera le processus d'élimination des classes (en effet, plus un score de classe est faible, plus il y a de chance que cette classe soit éliminée par la suite).

5.3.1 Politique de sélection

Notre algorithme doit gérer différentes problématiques : 1/ définir la politique d'élimination des classes de l'ensemble des solutions possibles et 2/ maximiser la confiance de la prédiction en accumulant le plus de votes pour la bonne classe. Notre approche est basée sur le calcul de la déviation du score observé μ_i avec l'espérance de cette quantité sous l'hypothèse que la classe ℓ_i est la bonne classe de l'exemple $\mathbf{x}$ en cours de classification.

Comme spécifié en section 5.2.3, l'utilisation de la borne de Hoeffding [1] pour $\mu_{i,t}$ nous donne :

$$\mathbb{P}[\mu^*_{i,t} - \mu_{i,t} \geq \delta | \mathbf{x} \in \ell_i] \leq e^{-T_{i,t}\delta^2}$$

Un seuil α_{min} est défini au départ de l'algorithme. Ce seuil définit le seuil de confiance à partir duquel l'algorithme élimine une classe de l'ensemble des classes possibles. Nous allons noter par la suite $\alpha(\mu^*_{i,t}, \mu_{i,t}, T_{i,t}) = e^{-max(0,\mu^*_{i,t}-\mu_{i,t})^2 T_{i,t}}$ qui dénote la différence entre le score obtenu et le score espéré au temps t sous l'hypothèse que l'exemple à classifier est de la classe ℓ_i en fonction de $T_{i,t}$ le nombre de fois où la classe a été considérée. Lorsque la borne se vérifie pour α_{min}, la borne s'écrit :

$$\mathbb{P}\left[\mu^*_{i,t} - \mu_{i,t} \geq \sqrt{\frac{-\ln \alpha_{min}}{T_{i,t}}} | \mathbf{x} \in \ell_i\right] \leq \alpha_{min}$$

$$\mathbb{P}\left[\alpha(\mu^*_{i,t}, \mu_{i,t}, T_{i,t}) \leq \alpha_{min} | \mathbf{x} \in \ell_i\right] \leq \alpha_{min}$$

Ainsi, afin d'éliminer itérativement des classes il est nécessaire de minimiser au mieux à chaque itération la norme du vecteur $\boldsymbol{\alpha}^t = (\alpha(\mu^*_{1,t}, \mu_{1,t}, T_{1,t}), \ldots, \alpha(\mu^*_{K,t}, \mu_{K,t}, T_{K,t}))$. Nous proposons dans la suite une politique heuristique gloutonne dans cet objectif. Pour cela, à chaque itération t de l'algorithme, notre politique consiste à parcourir l'ensemble des dichotomies disponibles, à estimer pour chacune l'espérance de gain pour $\|\boldsymbol{\alpha}^t\|$ si le classifieur correspondant est sélectionné, puis à sélectionner celui qui permet la plus grande décroissance de $\|\boldsymbol{\alpha}^t\|$.

L'algorithme 7 résume les principes de notre approche : à chaque itération t, une dichotomie $\mathbf{c}$ est choisie par la politique, le classifieur associé $h_{\mathbf{c}^t}$ est considéré et les vecteurs T^t, μ^t sont mises à jour en fonction du retour de $h_{\mathbf{c}^t}(\mathbf{x})$. On rappelle que les valeurs de $q_{\mathbf{c}^t}$ correspondent aux paramètres des lois de Bernoulli et peuvent être estimées par l'utilisation d'un ensemble de validation.

A chaque étape, la sélection du classifieur se fait par une politique heuristique décrite par l'algorithme 8 qui est en charge de trouver la meilleure dichotomie dans le but de minimiser $\|\boldsymbol{\alpha}^t\|$. L'algorithme va estimer $\|\boldsymbol{\alpha}^{t+1}\|$ à l'étape $t+1$ en fonction de la dichotomie $\mathbf{c}$ utilisée. Ainsi, pour chaque dichotomie $\mathbf{c}$ à notre disposition (et non utilisée), on estime la variation de $\|\boldsymbol{\alpha}^t\|$ suivant que $r(\mathbf{c}, \ell_i, \mathbf{x})$ retournerait $+1$ ou -1. Ainsi, les fonctions

1. Nous prenons en compte uniquement un seul côté de la borne de Hoeffding, puisque nous nous intéressons uniquement au cas où la moyenne des votes reçues est trop basse par rapport à l'espérance.

Algorithm 7 Processus de Classification

$\mathbf{x} \leftarrow$ exemple à classifier
Choisir $\mathbf{c}^0$ aléatoirement parmi $\{\mathbf{c} \in \Phi || \mathbf{c}| = K\}$
pour $i = 1 \dots K$ **faire** ▷ Initialisation
    $T_{i,0} = |c_{i,0}|$
    $\mu_{i,0} = r(\mathbf{c}^0, \ell_i, \mathbf{x})$
    $\mu^*_{i,0} = |\mathbf{c}_{i,0}| q_{\mathbf{c}^0}$
fin pour
pour $t = 1 \dots T_{max}$ **faire**
    $\mathbf{c}^t = \pi_{MUSCA}(\boldsymbol{\mu}^t, \boldsymbol{T}^t)$ ▷ sélection de la dichotomie
    pour $i = 1 \dots K$ **faire**
        $T_{i,t} = T_{i,t-1} + |c_{i,t}|$ ▷ incrémentation du nombre de votes
        $\mu_{i,t} = \mu_{i,t-1} + (r(\mathbf{c}^t, \ell_i, \mathbf{x}) - \mu_{i,t-1})/T_{i,t}$ ▷ màj. des scores observés
        $\mu^*_{i,t} = \mu^*_{i,t-1} + |c_{i,t}|(q_{\mathbf{c}^t} - \mu^*_{i,t-1})/T_{i,t}$ ▷ màj. des scores attendus
    fin pour
fin pour
retourner $\arg\max_i \mu_{i,T_{max}}$

$\alpha_i^+(\mathbf{c})$ et $\alpha_i^-(\mathbf{c})$ estiment les nouvelles valeurs de $\boldsymbol{\alpha}^t$ suivant la valeur de $r(\mathbf{c}, \ell_i, \mathbf{x})$. Il est important de noter que les réponses des classifieurs $h_{\mathbf{c}}$ ne sont pas calculé réellement. On ne fait que simuler les réponses et estimer l'impact qu'*aurait* l'utilisation d'un tel code sur $\|\boldsymbol{\alpha}^t\|$.

Cette politique considère uniquement les dichotomies codant les classes qui vérifient le test d'hypothèse avec une confiance α_{min}, $\{\ell_i | \alpha(\mu^*_i, \mu_i, T_i) > \alpha_{min}\}$. Cette élimination itérative des classes permet d'une part de concentrer l'information demandée sur les classes les plus probables et d'éviter ainsi de considérer des classifieurs qui apportent des informations aléatoires ; d'autre part, elle permet d'obtenir au fur et à mesure une information plus fiable : comme les dichotomies restantes encodent de moins en moins de classes, les classifieurs correspondants seront généralement beaucoup plus précis que ceux considérés initialement.

Algorithm 8 Politique de sélection π_{MUSCA}

Soit $\boldsymbol{\mu}$, $\boldsymbol{T}$, α_{min}
$\mathcal{C} = \{i \,|\, \alpha(\mu^*_i, \mu_i, T_i) > \alpha_{min}\}$ ▷ Classes candidates
define $\alpha_i^+(\mathbf{c}) = \alpha(\mu^*_i + |c_i|(q_{\mathbf{c}} - \mu_i)/(T_i + |c_i|), \mu_i + |c_i|(c_i - \mu_i)/(T_i + |c_i|), T_i + |c_i|)$
▷ Simulation du gain pour $r(\mathbf{c}, \ell_i, \mathbf{x}) = 1$
define $\alpha_i^-(\mathbf{c}) = \alpha(\mu^*_i + |c_i|(q_{\mathbf{c}} - \mu_i)/(T_i + |c_i|), \mu_i + |c_i|(-c_i - \mu_i)/(T_i + |c_i|), T_i + |c_i|)$
▷ Simulation du gain pour $r(\mathbf{c}, \ell_i, \mathbf{x}) = -1$
retourner $\arg\min_{\mathbf{c} \in \Phi | c_i = 0 \forall i \notin \mathcal{C}} \sum_{i \in \mathcal{C}} \alpha_i^+(\mathbf{c}) + \alpha_i^-(\mathbf{c})$

5.3.2 Implémentation Pratique et Complexité

L'ensemble des dichotomies Φ (ensemble des dichotomies de l'ensemble $\mathcal{L}$) est bien trop large pour que tous les classifieurs correspondant aux dichotomies soient appris étant donné que la cardinalité de cet ensemble croît exponentiellement avec le nombre de classes. Pour l'implémentation de notre modèle, un sous-ensemble Φ_s est tiré aléatoirement et constitue l'ensemble des dichotomies à notre disposition. Afin de respecter un équilibre dans cet ensemble et s'assurer d'avoir des classifieurs génériques et des classifieurs plus spécifiques, le tirage aléatoire pour un code $\mathbf{c}$ de Φ_s s'effectue en deux étapes, un pour déterminer la parcimonie du code et un autre pour déterminer les classes concernées : un premier nombre k est tiré aléatoirement pour déterminer le nombre de classes encodées par la dichotomie (entre 2 et K) ; puis k classes sont tirées uniformément et les composantes correspondantes de $\mathbf{c}$ sont tirés aléatoirement parmi $\{-1, 1\}$.

Les classifieurs correspondants aux dichotomies de Φ_s sont entraînés ensuite avec un ensemble d'entraînement et l'évaluation de leur précision $q_{\mathbf{c}}$ est calculée grâce à un ensemble de validation.

La complexité du processus de classification est directement corrélée avec le nombre de classifieurs utilisés. Ce nombre de classifieurs est donné directement par le nombre d'itérations de l'algorithme, soit T_{max}. La complexité de classification τ de notre méthode est bornée par $O((|\Phi_s| + \tau_f)T_{max})$ ou τ_f correspond à la complexité de classification d'un seul classifieur. En pratique, la complexité de notre modèle est très inférieure : de nombreuses classes sont éliminées au fur et à mesure et le nombre de dichotomies à considérer se réduit très rapidement.

De plus, dans le contexte de classification avec un grand nombre de classes, τ_f est souvent très grand devant $|\Phi_s|$ étant donné que τ_f opère sur la dimension de l'espace des exemples qui peut atteindre des très grandes valeurs en particulier pour des documents textuels. En comparaison, pour K classes, le schéma de classification OAA a une complexité de classification de $O(\tau_f K)$, les approches ECOC sont en $O(\tau_f T_k)$ avec T_k la longueur des codes correcteurs d'erreurs, les méthodes hiérarchiques en $O(Qlog_Q(K)\tau_f)$ avec Q le nombre d'enfant par nœud.

En pratique, il a été observé sur différents problèmes d'apprentissage réels que le temps de calcul pour la politique de sélection d'une dichotomie pour un exemple était en moyenne

10 fois plus rapide que le temps de calcul d'un classifieur simple. Cela reste vrai tant que le nombre de dimensions de l'espace des exemples est plus grand que le nombre de classes. Dans le cas où l'on traiterait de problèmes avec moins de dimensions (en image par exemple), notre méthode se révélerait moins rapide sauf dans le cas où les classifieurs simples utilisés sont suffisamment complexe pour rendre négligeable le temps de calcul de la politique. Il faut ainsi garder en mémoire cette limitation qui va guider le choix de la famille de classifieurs qui va spécifier le type des classifieurs simples de façon à garder le gain de temps de calcul suffisamment intéressant.

5.4 Expérimentations

5.4.1 Protocole

Nous avons comparé notre modèle séquentiel (MUSCA) avec des méthodes hiérarchiques et ensemblistes de l'état de l'art permettant d'accélérer le processus de classification qui ont été présentées précédemment. Nous avons évalué les performances de notre méthode en la comparant avec un arbre de classes construit avec une méthode de clustering spectral comme dans Bengio et al. [2010] ainsi qu'avec des méthodes ECOC [Allwein et al., 2001; Dietterich and Bakiri, 1995]. Nous avons aussi comparé les résultats obtenus avec une méthode OAA ainsi qu'avec un arbre reprenant exactement la hiérarchie donnée avec les données lorsque cette information était disponible.

Pour chacune de ces méthodes, nous avons calculé les taux de bonne classification pour différents ratios de complexité en faisant varier des paramètres comme le nombre de fils par nœud ou bien la longueur des codes pour les méthodes de type ECOC. Pour notre modèle, il suffit d'arrêter l'algorithme au moment voulu pour obtenir un ratio de complexité spécifique. On rappelle que ce ratio de complexité représente la fraction de temps nécessaire au modèle en question pour produire une classification par rapport au modèle de référence OAA. Plus ce ratio est petit, plus la méthode permet une accélération du processus de classification.

Pour l'implémentation des ECOC, nous avons tiré aléatoirement cinq mille codes ternaires et nous avons conservé ceux qui maximisent la distance de Hamming minimale entre les paires de code de classe [Allwein et al., 2001]. Ensuite, nous avons implémenté

deux méthodes de décodage pour le processus de classification : la première façon consiste à utiliser simplement la distance de Hamming entre le code de l'exemple à classifier et ceux des classes [Dietterich and Bakiri, 1995], la seconde façon est plus élaborée et consiste à utiliser les distances aux hyperplans séparateurs pour avoir une mesure plus fine pour chaque classifieur simple (voir sous-section 2.3.3).

Pour notre modèle, nous avons utilisé un ensemble d'approximativement dix mille classifieurs générés aléatoirement pour la base de données DMOZ et 300 pour la base de données Sector. Le ratio de complexité est contrôlé en faisant varier T_{max} (nombre maximal d'itérations de l'algorithme). Nous avons fixé le seuil pour éliminer les classes les moins probables à $\alpha_{min} = 0.01$. Dans toutes les expériences, les classifieurs utilisés (dans les nœuds des arbres, ou pour les codes des ECOC) étaient des SVM linéaires [Cortes and Vapnik, 1995] régularisés sur un ensemble de validation [2].

5.4.2 Résultats

Les résultats reportés sont ceux de notre méthode (MUSCA) ainsi que des trois méthodes de l'état de l'art présentées précédemment (méthode Hiérarchique de clustering spectral : H-SC ; méthode ECOC avec distance de Hamming : E-Hamming ; méthode ECOC avec utilisation de la distance à l'hyperplan : E-LossBased ; méthode à plat un-contre-tous : OAA ; et hiérarchie originale : H-Ontology).

On rappelle que les méthodes de sélection automatique de classifieurs ne bénéficient pas de la connaissance d'une ontologie pré-existante. La méthode H-Ontology est à priori avantagé si l'ontologie est pertinente pour la classification. Les résultats sont moyennés sur plusieurs expériences à chaque fois (5) [3].

Le tableau 5.1 montre les résultats sur le jeu de données Sector des différentes méthodes pour un ratio de complexité de 20% (plus le jeu de données est petit, plus il est difficile de gagner du temps). Les tableaux 5.2 et 5.3 montrent les pourcentages de bonne classification des méthodes à comparer pour un ratio de complexité de 4% sur le jeu de données DMOZ et sur des sous-échantillons de 1000 classes. Le tableau 5.4 montre

2. Nous avons utilisé l'implémentation des SVM venant de la bibliothèque de *Scikit Learn*. Les mêmes méta paramètres ont été utilisé pour toutes les méthodes : (*l2* loss, pondération automatique des classes suivant leur densité, pénalité *l2*, constante C obtenue par cross validation).

3. Pour notre méthode, la réserve de classifieurs est tirée aléatoirement à chaque expérience ; pour les méthodes ECOC, les codes sont tirés aléatoirement aussi. L'écart-type étant insignifiant par rapport aux écarts entre les méthodes, ils n'ont pas été reportés dans le tableau pour plus de clarté.

les résultats sur le jeu de données Wikipedia. La figure 5.3 présente les pourcentages de bonne classification obtenues sous la forme d'un graphique permettant de mieux se rendre compte du dilemme vitesse de classification / pourcentage de bonne classification sur le jeu de données DMOZ. Les figures **??** (en annexe) montrent ce pourcentage de bonne classification en fonction du ratio de complexité pour les sous-samples de DMOZ.

Modèles	Type	Ratio de Complexité	Sector Acc%
OAA	Plat	1 (×1)	**94.12**%
MUSCA	Plat	0.2 (×5)	**92.67**%
H-SC	Arbre	0.2 (×5)	88.52%
E-Hamming	Plat	0.2 (×5)	27.73%
E-LossBased	Plat	0.2 (×5)	89.55%

TABLE 5.1: Taux de bonne classification obtenus par les différents modèles sur la base de données Sector. En gras, les résultats étant significativement supérieurs à tous les autres.

Modèles	Type	Ratio de Complexité	DMOZ Acc%
OAA	Plat	1 (×1)	**38.28**%
H-Ontology	Arbre	0.008 (×125)	38.05%
MUSCA	Plat	0.04 (×25)	**36.49**%
H-SC	Arbre	0.04 (×25)	32.09%
E-Hamming	Plat	0.04 (×25)	33.68%
E-LossBased	Plat	0.04 (×25)	35.08%

TABLE 5.2: Taux de bonnes classification obtenus par les différents modèles sur la base de données DMOZ. En gras, les résultats étant significativement supérieurs à tous les autres.

Modèles	Type	Ratio de Complexité	DMOZ (sous-samples) : Acc% Set1	Set2	Set3	Set4	Set5
OAA	Plat	1 (×1)	45.50%	55.36%	60.83%	54.86%	52.16%
H-Ontology	Arbre	0.06 (×16)	46.73%	57.15%	**63.16**%	**57.08**%	54.84%
MUSCA	Plat	**0.2** (×5)	**49.78**%	**57.43**%	**62.36**%	**57.05**%	**55.29**%
H-SC	Arbre	0.2 (×5)	44.77%	53.72%	58.29%	52.63%	51.97%
E-Hamming	Plat	0.2 (×5)	45.03%	51.99%	56.98%	52.38%	50.89%
E-LossBased	Plat	0.2 (×5)	46.72%	54.42%	59.15%	54.20%	52.19%

TABLE 5.3: Taux de bonne classification obtenues par les différents modèles sur les échantillons issues du jeu de données DMOZ. En gras, les résultats étant significativement supérieurs à tous les autres.

Models	Type	Ratio de Complexité	Wikipedia (sous-samples) : Acc% Set1	Set2	Set3	Set4	Set5
OAA	Plat	1 (×1)	2.55%	7.75%	5.97%	5.98%	27.1%
MUSCA	Plat	**0.2** (×5)	**3.93**%	**8.01**%	**6.61**%	**6.23**%	**27.01**%
H-SC	Arbre	0.2 (×5)	0.26%	1.04%	4.20%	2.50%	25.9%
E-Hamming	Plat	0.2 (×5)	1.59%	7.22%	4.03%	5.24%	26.9%
E-LossBased	Plat	0.2 (×5)	2.40%	7.65%	5.64%	4.74%	26.9%

TABLE 5.4: Pourcentages de bonne classification obtenues par les différents modèles sur le jeu de données Wikipédia. En gras, les résultats étant significativement supérieurs à tous les autres.

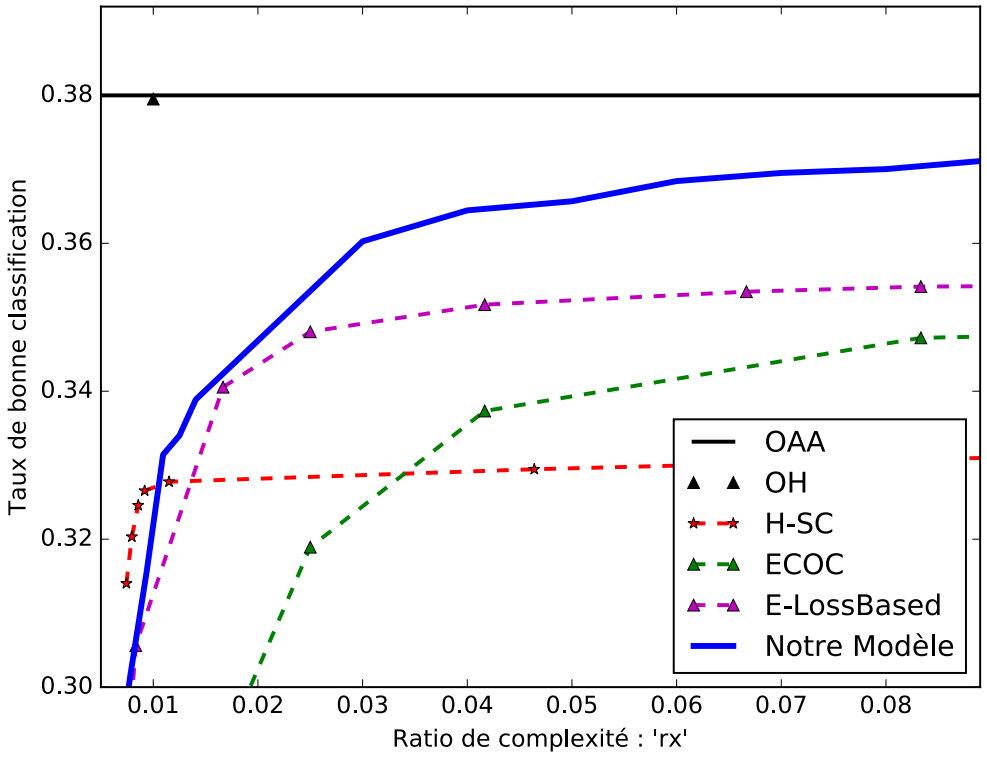


FIGURE 5.3: Pourcentages de bonne classification des différents modèles en fonction du ratio de complexité sur le jeu de données DMOZ.

5.4.3 Discussion

Notre méthode a obtenu des performances globalement meilleurs que ceux des méthodes de l'état de l'art. La méthode OAA a confirmé sa capacité à résoudre les problèmes multi-classes avec de bonnes performances en taux de bonne classification. Cependant, OAA reste lent comparé aux autres méthodes présentées ici. Sur les plus grosses base de données utilisées, nous avons obtenu un ratio de complexité de $\frac{1}{25}$ avec des performances sensiblement équivalentes. Par exemple, une classification d'un exemple de la

base de données DMOZ demande 12294 classifieurs pour OAA là où notre méthode n'en demande que 490 pour un pourcentage de bonne classification légèrement inférieur. Sur certaines expériences comme celles avec la base de données Wikipédia ou les échantillons de DMOZ, notre méthode à réussi à battre la méthode OAA tout en proposant un temps de classification significativement réduit. Ces différences de comportements suivant les tailles des bases de données pourrait s'expliquer par un caractère non linéaire du besoin en nombre de classifieurs pour cette tâche. Dans ce cas, cela pourrait être un frein quant à la capacité de passage à l'échelle de notre méthode dans des conditions défavorables.

En effet, s'il n'y a pas assez de variabilité parmi l'ensemble de classifieurs à disposition, alors notre approche va moins bien performer et ne pourra pas faire jeu égal avec une méthode de type OAA. Le travail consistant à optimiser cette réserve de classifieurs pour coller au mieux à la méthode de classification est une direction que nous n'avons pas explorée et qui pourrait se montrer prometteuse.

Notons que lors du challenge sur les données DMOZ, le meilleur score obtenu était significativement supérieur aux meilleurs résultats reportés ici (46.7%). Comme le challenge permettait l'utilisation de l'information hiérarchique originale des classes, ce score n'est pas comparable directement avec les résultats des méthodes présentées ici.

Comparativement à notre approche, la figure 5.3 montre que nous avons obtenu des résultats supérieurs aux deux méthodes ECOC pour tout ratio de complexité. Pour les plus grosses valeurs de ratio de complexité, notre approche réussit à obtenir de meilleures performances que les autres méthodes ensemblistes ce qui est expliqué par la qualité des classifieurs qui ont été sélectionnés là où les autres méthodes ensemblistes ne proposent pas une construction spécifique de l'ensemble de classifieurs à utiliser pour classifier un exemple particulier.

Une chose intéressante à noter est la capacité de l'approche hiérarchique par clustering spectral à être meilleure que notre approche pour les ratios de complexité les plus petits. Ceci est une conséquence de la capacité naturelle de la structure de l'arbre à éliminer rapidement les classes de l'ensemble des classes possible avec un facteur logarithmique. Dès lors que la contrainte sur le ratio de complexité est un peu relâchée, notre modèle réussit mieux à classifier les exemples et montre une plus grande robustesse. La aussi, les résultats présentés ici concernent uniquement une réduction du nombre de classifieurs.

Nous n'avons pas cherché à intégrer d'autres méthodes d'accélération de la classification comme la sélection de caractéristiques ou des projections.

La figure 5.4 montre la décroissance du nombre de classes encodées par les dichotomies utilisées - normalisée par le nombre de classes - au fur et à mesure des itérations de l'algorithme de classification pour un exemple. Une norme de 1 correspond à un classifieur portant sur l'ensemble des classes et une norme proche de 0 correspond à une dichotomie codant très peu de classes. La figure montre que *MUSCA* commence à agréger de l'information venant de classifieurs denses lors des premières itérations de l'algorithme. Une fois que suffisamment d'information a été agrégée, il est possible d'éliminer un nombre conséquent de classes qui présentent trop peu d'intérêt pour la classification de l'exemple en cours : le changement de stratégie s'opère une fois que l'algorithme doit départager un plus petit nombre de classes ; les classifieurs sont moins denses et donc plus précis pour réussir au mieux à départager les classes restantes.

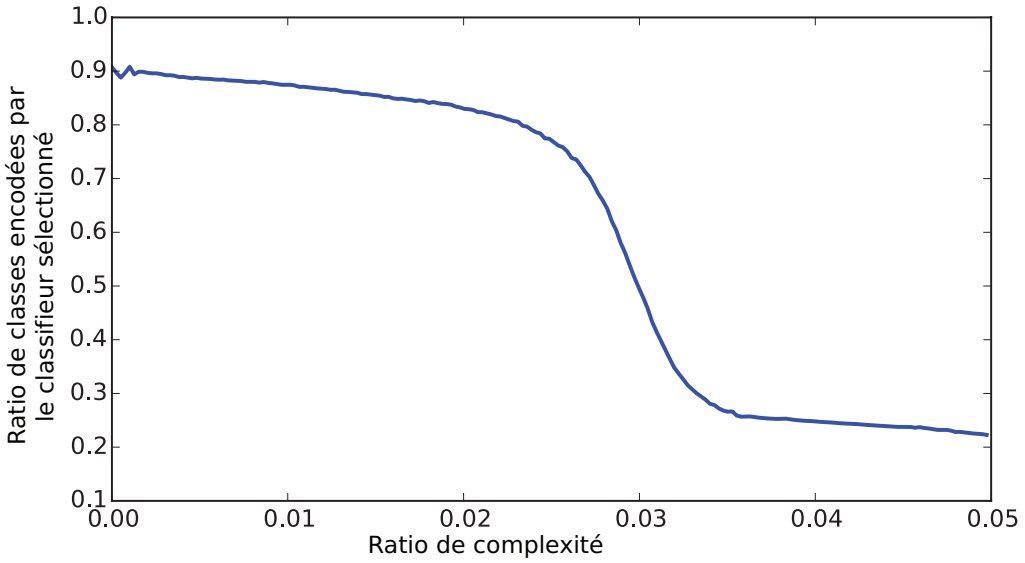


FIGURE 5.4: Moyenne du nombre de classes encodées par les dichotomies sélectionnées pendant le processus d'inférence à chaque itération sur la base de données DMOZ.

En optimisant la séquence de classifieur dynamiquement parmi la réserve de classifieurs à sa disposition, la méthode proposée se place à mi-chemin entre une méthode hiérarchique et une méthode ensembliste.

Nous avons affiché Figure 5.5 les scores de taux de bonne classification de groupes d'exemples appartenant a des classes possédant un nombre similaire d'exemples d'apprentissages. Cela permet de rendre compte du gain de notre méthode séquentielle par

rapport aux méthodes classiques ensemblistes. En particulier, on peut noter que notre approche permet de mieux distinguer les classes avec peu d'exemples d'apprentissage. En choisissant le classifieur le plus utile, notre méthode permet de désambiguïser une situation où il resterait plusieurs classes probables pour l'exemple à classifier.

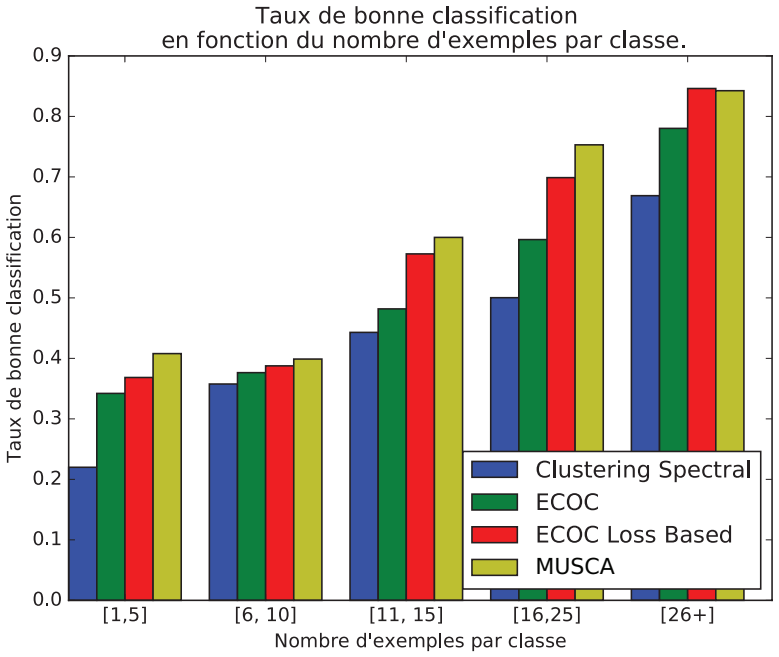


FIGURE 5.5: Comparaison des taux de bonne classification entre un modèle hiérarchique, des modèles ensemblistes de type ECOC ainsi que notre modèle séquentiel (MUSCA). L'ensemble de test à été séparé en trois groupes distincts, par regroupement des exemples dont la classe contient un certain nombre d'exemples.

Notre approche permet ainsi de sélectionner le classifieur qui va tenter de séparer les classes les plus corrélées, car ayant des scores de confiance semblable.

5.5 Conclusion

Nous avons présenté dans ce chapitre un modèle combinant les principes les plus intéressants des approches hiérarchiques et des approches ensemblistes. En adaptant la capacité de correction des erreurs des ECOC ainsi que l'idée d'utiliser un sous-ensemble spécifique de classifieurs dynamiquement choisis, le modèle que l'on a décrit permet d'aller au-delà des limites de ces deux ensembles de méthodes. Le tableau 5.5 montre en quoi la méthode présentée est allée au-delà de ce qui était proposé dans la littérature en termes de méthode de réduction : notre méthode combine un processus de prédiction séquentiel complé à un processus de redondance de type ECOC.

	$f_{\mathcal{S}}$	$\pi_{\mathcal{S}}$	$\mathcal{M}_{\mathcal{S}}$	$u_{\mathcal{S}}$	$p_{\mathcal{S}}$
Ensembliste	O	X	O	O	O
Hiérarchique	O	O	X	X	X
MUSCA	O	O	O	O	O

TABLE 5.5: Caractéristiques des méthodes étudié dans ce travail (formalisme décrit Section 3).

Le modèle proposé utilise la borne d'Hoeffding à la fois pour éliminer rapidement des classes non pertinentes mais également comme objectif pour la politique de sélection des dichotomies. En adaptant les bornes nécessaires à nos objectifs de classification, nous avons pu tirer parti d'un résultat théorique pour l'appliquer à une problématique réelle de classification multi-classes. Nous avons montré expérimentalement les avantages de notre modèle vis-à-vis de travaux récents traitant de la même problématique.

Des travaux futurs peuvent aisément découler de ce type de raisonnement. La première chose qui mériterait d'être exploré consiste à régler et à optimiser la réserve de classifieurs pour que cet ensemble soit adapté au processus de classification de notre méthode. En optimisant ces classifieurs, l'ensemble des dichotomies serait mieux couvert et moins d'erreurs d'approximations serait ainsi faites. Comme vu dans le travail de Crammer and Singer [2002a], la qualité et la précision des classifieurs simples bornent la précision finale du classifieur complexe agrégeant les réponses de ces classifieurs simples. Ainsi, en optimisant la qualité des classifieurs simples, on diminue directement l'erreur d'approximation. C'est pourquoi l'optimisation de la réserve de classifieurs a le potentiel d'augmenter considérablement le taux de bonne classification de ce modèle.

Un autre point d'intérêt dans le modèle proposé ici est sa capacité à s'adapter à la volée pour proposer une classification dans un temps imparti. Ainsi, dans le scénario où la tâche est de classifier un flux variable d'information en direct, il peut être intéressant d'être capable d'utiliser au mieux le temps imparti pour faire une classification, et d'adapter le temps de classification pour chaque exemple. Un exemple facile à classifier peut nécessiter beaucoup moins de classifieurs ce qui autoriserait un autre exemple à utiliser plus de temps de classification, augmentant ainsi la qualité moyenne sur un ensemble de classifications.

Dans un autre ordre d'idée, il serait important d'adapter le modèle pour le traitement de classification multi-label étant donné que les tâches de classification dans un grand nombre de catégorie concernent souvent des problèmes multi-étiquettes.

Chapitre 6

Conclusion

6.1 Synthèse des méthodes d'accélération du processus d'inférence

Nous avons étudié dans ce travail de thèse la classification dans un grand nombre de catégories. Ce problème de classification extrême est un problème crucial qui a retenu récemment l'attention avec l'apparition de problèmes de catégorisation pouvant atteindre plusieurs centaines de millier de classes.

Le dilemme classique du temps de classification par rapport à la précision de la classification prend alors toute son importance dans ce contexte. En effet, l'étude de ce dilemme permet de répondre à une question lorsqu'un tel problème d'extrême classification doit être résolu : "Quelle sera la perte en terme de taux de bonne classification du modèle si moins de puissance est allouée au processus de classification d'un exemple ?".

Dans le chapitre 2 de cette thèse, nous avons présenté les modèles usuels utilisés dans le cadre de la classification multi-classses ainsi que les modèles récents de la littérature scientifique spécifiques aux problèmes à très grand nombre de classes. Ces modèles utilisent quasiment tous un ensemble de classsifieurs spécialisés dans la séparation de deux sous-ensembles de classes du problème initial. Deux grandes familles sont généralement distinguées en fonction de l'utilisation qui est faîte de ces classifieurs : les approches hiérarchiques qui utilisent de manière séquentielle certains classifieurs afin de rejeter au

fur et à mesure des itérations des classes jugées non compatibles, et les approches ensemblistes qui agrègent les réponses de tous les classifieurs afin de reconnaître la classe la plus probable. Cet état de l'art montre par l'abondance des travaux cités l'actualité du problème d'extrême classification et son intérêt théorique et applicatif. Par ailleurs, il met en évidence le peu de travaux consacrés à l'étude du dilemme temps d'inférence/perfomance de classification.

Dans le chapitre 3, nous avons proposé un formalisme qui d'une part regroupe l'ensemble des modèles présentés dans notre état de l'art, et d'autre part a l'avantage d'unifier dans un même schéma ces modèles afin d'appréhender leurs différences et de comparer leurs performances en terme de temps d'inférence et d'erreurs de classification. Ce travail préalable permet de donner une vision synthétique du domaine de la classification extrême. Il montre également que les approches passées en revue dans l'état de l'art sont toutes statiques quant à leur temps d'inférence : la séquence de classifieurs utilisés est déterminée à l'avance. Il n'existe pas de méchanisme d'adaptation en fonction de la connaissance acquise au fur et à mesure.

Cette étude a posé les bases pour proposer une méthode capable de mieux gérer le compromis Précision/Vitesse. Les modèles présentés aux chapitres 4 et 5 présentent les modèles résultant du travail effectué pendant ce doctorat sur le problème de l'inférence en classification extrême.

Le premier modèle (chapitre 4) se place dans le cadre des méthodes hiérarchiques. Son objectif est de créer une hiérarchie de classifieurs qui permette d'atteindre un temps d'inférence moyen fixé à l'avance avec la meilleure performance en prédiction possible. L'approche proposée part de l'hypothèse que les classifieurs multi-classes les plus performants sont ceux du type un-contre-tous et que plus une hiérarchie est développée (donc plus elle permet une inférence rapide), plus la prédiction est mauvaise. Ainsi, notre algorithme considère initialement une hiérarchie plate - autant de feuilles que de classes - puis itérativement des nœuds sont développés afin de diminuer au mieux le temps d'inférence tout en dégradant le moins possible les performances en classification. Ce modèle montre de très bonnes performances, dépassant même celles du un contre tous. Cependant, tout comme les modèles hiérarchiques, il ne permet pas une flexibilité dans son utilisation, la séquence des classifieurs étant toujours apprise lors de l'apprentissage du modèle sans possibilité d'adaptation lors de l'utilisation du modèle.

Le second modèle proposé (chapitre 5) tire parti de l'analyse des forces des deux familles d'approches : d'une part notre modèle utilise la rapidité d'inférence des méthodes hiérarchiques par l'utilisation d'une séquence de classifieurs calculée en fonction de l'exemple considéré, et d'autre part il utilise la performance de classification des méthodes ensemblistes grâce à la grande redondance des informations considérées (essentiellement pour les approches ECOC). Nous avons formalisé le problème comme un processus de décision séquentielle qui vise à éliminer le plus grand nombre de classes à chaque itération en fonction de l'information acquise. La modélisation du score de chaque classe comme le résultat d'une série d'expériences de Bernouilli nous permet d'établir un critère d'élimination des classes. Pour cela, une borne de confiance est utilisée entre le score attendu pour une classe et celui obtenu en fonction des classifieurs utilisés. A chaque étape, le processus de décision choisit le classifieur qui a la plus grande espérance de mener aux plus grands nombres de classes éliminées. Au bout d'un nombre d'itérations fixé à l'avance, l'algorithme renvoie la classe la plus probable en fonction du score calculé, comme il est d'usage dans les méthodes ensemblistes. Notre approche présente ainsi une grande flexibilité pour le choix du nombre d'itérations et donc pour le temps d'inférence d'un exemple. Expérimentalement, de très bonnes performances ont été atteintes que ce soit lorsque peu d'itérations sont autorisées ou lorsque beaucoup de ressources sont disponibles.

6.2 Futures approches

De futures extensions de ce travail sont possibles pour permettre le développement et l'utilisation des modèles proposés ici. Nous avons parlé de classification multi-classes *mono*-label. Il est courant aujourd'hui que les problèmes avec beaucoup de catégories soient associés à une tâche de classification multi-labels (multi-étiquettes) où un exemple peut être associé à plus d'une catégorie. Un développement des modèles présentés pour prendre en charge les tâches de classification multi-labels permettrait d'augmenter les cas d'applications des modèles.

Une autre problématique qui a été mentionnée dans ce travail et qui pourrait faire l'objet de recherches futures concerne l'optimisation du temps de classification comptabilisé sur un ensemble d'exemples à classifier, et non pas comptabilisé sur un seul exemple. Les calculs de complexité des algorithmes de classification qui ont été explicités ici concernent

les temps de classification d'un seul exemple. Il serait intéressant de poser le problème d'optimisation du temps de classification non pas pour un seul exemple, mais pour un ensemble d'exemples. L'algorithme de classification aurait alors à choisir entre utiliser plus de temps pour la classification d'un exemple plus difficile nécessitant plus de calculs, et inversement pour des exemples faciles dont la catégorisation nécessiterait moins de puissance. Un travail futur consisterait à développer une extension du modèle de décision séquentielle (chapitre 5) qui serait outillé pour permettre une telle optimisation du temps de classification global.

Bibliographie

Samy Bengio, Jason Weston, and David Grangier. Label embedding trees for large multi-class tasks. *Advances in Neural Information Processing Systems*, 23(1) :163–171, 2010. URL http://0-research.google.com.topcat.switchinc.org/pubs/archive/36578.pdf.

Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary : A unifying approach for margin classifiers. *The Journal of Machine Learning*, 1 : 113–141, 2001. URL http://dl.acm.org/citation.cfm?id=944737.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20 : 273–297, 1995. ISSN 08856125. doi : 10.1007/BF00994018.

Leon Bottou and Olivier Bousquet. The Tradeoffs of Large Scale Learning. *Artificial Intelligence*, 20 :161–168, 2008. ISSN <null>. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.168.5389&rep=rep1&type=pdf.

Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5 :101–141, 2004. URL http://dl.acm.org/citation.cfm?id=1005336.

Andres McCallum and Kamal Nigam. A Comparison of Event Models for Naive Bayes Text Classification. *AAAI/ICML-98 Workshop on Learning for Text Categorization*, pages 41–48, 1998. ISSN 0343-6993. doi : 10.1.1.46.1529. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.9324&rep=rep1&type=pdf.

Anna Choromanska and John Langford. Logarithmic Time Online Multiclass prediction. pages 1–13, 2014.

Bernhard Scholkopf, Chris Burges, and Vladimir Vapnik. Extracting Support Data for a Given Task. In *first international conference on knowledge discovery and data mining*, number x, pages 252–257, 1995. ISBN 0-929280-82-2.

Vladimir Vapnik. *Statistical Learning Theory*. 1998.

Florent Perronnin, Zeynep Akata, Zaid Harchaoui, and Cordelia Schmid. Towards good practice in large-scale learning for image classification. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3482–3489, 2012. ISSN 978-1-4673-1228-8. doi : 10.1109/CVPR.2012.6248090. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6248090.

S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited : a stepwise procedure for building and training a neural network. 1990.

Jerome H. Friedman. Another approach to polychotomous classification. 1996.

John C. Platt, Nello Cristianini, and John Shawe-taylor. Large margin DAGs for multiclass classification. *Neural Information Processsing Systems*, pages 547–553, 2000. URL http://www.weizmann.ac.il/mathusers/bagon/CVspring07/files/DAGSVM.pdf.

TG Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 1995. URL http://arxiv.org/abs/cs/9501101.

John Langford and Alina Beygelzimer. Sensitive Error Correcting Output Codes. *Conference on Learning Theory*, pages 158–172, 2005.

Tianshi Gao and Daphne Koller. Multiclass Boosting with Hinge Loss based on Output Coding. *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 569–576, 2011a.

Robert E. Schapire. Using output codes to boost multiclass learning problems. *International Conference on Machine Learning*, (1) :1–9, 1997. URL http://www.cs.iastate.edu/~jtian/cs573/Papers/Schapire-ICML-97.pdf.

Moustapha Cissé, Thierry Artières, and Patrick Gallinari. Learning compact class codes for fast inference in large multi class classification. *European Conference on Machine Learninga*, 2012. URL http://link.springer.com/chapter/10.1007/978-3-642-33460-3_38.

Bin Zhao and Eric P. Xing. Sparse output coding for large-scale visual recognition. *Computer Vision and Pattern Recognition*, pages 3350–3357, 2013. ISSN 10636919. doi : 10.1109/CVPR.2013.430.

Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, pages 201–233, 2002a. URL http://link.springer.com/article/10.1023/A:1013637720281.

Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. *Conference on Information and knowledge management*, page 78, 2004. doi : 10.1145/1031171.1031186. URL http://portal.acm.org/citation.cfm?doid=1031171.1031186.

Tie-Yan Liu, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. Support vector machines classification with a very large-scale taxonomy. *ACM SIGKDD Explorations Newsletter*, 7(1) :36–43, 2005a. ISSN 19310145. doi : 10.1145/1089815.1089821.

Volkan Vural and Jennifer G. Dy. A hierarchical method for multi-class support vector machines. *International Conference on Machine Learning*, 2004. URL http://dl.acm.org/citation.cfm?id=1015427.

Shantanu Godbole, Sunita Sarawagi, and Soumen Chakrabarti. Scaling multi-class support vector machines using inter-class confusion. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02*, page 513, 2002. doi : 10.1145/775107.775122. URL http://portal.acm.org/citation.cfm?doid=775047.775122.

Yangchi Chen Yangchi Chen, M.M. Crawford, and J. Ghosh. Integrating support vector machines in a hierarchical output space decomposition framework. *IGARSS 2004. 2004 IEEE International Geoscience and Remote Sensing Symposium*, 2 :0–3, 2004. doi : 10.1109/IGARSS.2004.1368565.

Rohit Babbar and Ioannis Partalas. On Flat versus Hierarchical Classification in Large-Scale Taxonomies. *Neural Information Processsing Systems*, pages 1–9, 2013. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2013_5082.pdf.

Lei Liu, Prakash Mandayam Comar, and Sabyasachi Saha. Recursive NMF : Efficient label tree learning for large multi-class problems. *ICPR*, (Icpr) :2148–2151, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6460587.

Daphne Koller and Mehran Sahami. Hierarchically Classifying Documents Using Very Few Words. *International Conference on Machine Learning*, pages 170–178, 1997.

Soumen Chakrabarti, Byron E. Dom, Rakesh Agrawal, and Prabhakar Raghavan. Using taxonomy, discriminants, and signatures for navigating in text databases. In *Proceedings of VLDB-97, 23rd International Conference on Very Large Data Bases*, pages 446–455, 1997. ISBN 1-55860-470-7. URL http://www.vldb.org/conf/1997/P446.PDF.

Ke Wang. Building Hierarchical Classifiers Using Class Proximity. *Very Large Data Bases*, pages 363–374, 1999.

M Marszalek and Cordelia Schmid. Constructing category hierarchies for visual recognition. *European Conference for Computer Vision*, 2008. URL http://link.springer.com/chapter/10.1007/978-3-540-88693-8_35.

Gregory Griffin and Pietro Perona. Learning and using taxonomies for fast visual categorization. *Computer Vision and Pattern Recognition*, pages 1–8, June 2008. doi : 10.1109/CVPR.2008.4587410. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4587410.

Jia Deng, Sanjeev Satheesh, Alexander C. Berg, and Li. Fei-Fei. Fast and Balanced : Efficient Label Tree Learning for Large Scale Object Recognition. In *Neural Information Processsing Systems*, number 1, pages 1–9, 2011. URL http://books.nips.cc/papers/files/nips24/NIPS2011_0391.pdf.

Tianshi Gao and Daphne Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. *International Conference on Computer Vision*, 2011b. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6126481.

Edward Forgy. Cluster Analysis of Multivariate Data. 1965.

Song Liu, Haoran Yi, Liang-tien Chia, and Deepu Rajan. Adaptive Hierarchical Multi-class SVM Classifier for Texture-based Image Classification. *International Conference on Multimedia and Expo*, pages 3–6, 2005b.

Marcin Marszalek and Cordelia Schmid. Constructing category hierarchies for visual recognition. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5305 LNCS, pages 479–491, 2008. ISBN 3540886923. doi : 10.1007/978-3-540-88693-8-35.

Greg Hamerly and Charles Elkan. Learning the k in K means. *Advances in Neural Information Processing*, 2004. ISSN 1049-5258. doi : 10.1.1.9.3574âĂŐ.

Andrew Y Ng, Michael I Jordan, and Yair Weiss. On Spectral Clustering : Analysis and an algorithm. In T G Dietterich, S Becker, and Z Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, volume 14 of *Advances in Neural Information Processing Systems*, pages 849–856. MIT Press, 2001. doi : 10.1.1.19.8100. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.8100.

Marcin Marszalek and Cordelia Schmid. Semantic hierarchies for visual object Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007. ISSN 10636919. doi : 10.1109/CVPR.2007.383272.

Christiane Fellbaum. WordNet : An Lexical Electronic Database. 1998.

Alina Beygelzimer, John Langford, Yuri Lifshits, Gregory Sorkin, and Alex Strehl. Conditional Probability Tree Estimation Analysis and Algorithms. *Uncertainty in Artificial Intelligence*, 2009.

Max Planck and Ulrike Von Luxburg. A Tutorial on Spectral Clustering. *Statistics and Computing*, 17 :395–416, 2006. ISSN 09603174. doi : 10.1007/s11222-007-9033-z. URL http://www.springerlink.com/index/10.1007/s11222-007-9033-z.

Lihi Zelnik-manor, Lihi Zelnik-manor, Pietro Perona, and Pietro Perona. Self-tuning spectral clustering. *Advances in Neural Information Processing Systems 17*, 2 :1601–1608, 2004. doi : 10.1.1.84.7940. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.7940.

Liu Zhigang, Shi Wenzhong, Qin Qianqing, Li Xiaowen, and Xie Donghui. Hierarchical support vector machines.

Siyu Xia, Jiuxian Li, Liangzheng Xia, and Chunhua Ju. Tree-structured support vector machines for multi-class classification. *ISNN*, 2007.

Yiming Yang, J Zhang, and Bryan Kisiel. A scalability analysis of classifiers in text categorization. *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval SIGIR 03 (2003)*, pages 96–103, 2003. ISSN 01635840. doi : 10.1145/860435.860455. URL http://dl.acm.org/citation.cfm?id=860455.

Susan Dumais and Hao Chen. Hierarchical classification of Web content. *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '00*, pages 256–263, 2000. doi : 10.1145/345508.345593. URL http://portal.acm.org/citation.cfm?doid=345508.345593.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2 :265–292, 2002b. URL http://dl.acm.org/citation.cfm?id=944790.944813.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex Multi-task Feature Learning. *Machine Learning*, 73(3) :243–272, 2008.

Rich Caruana. Multitask Learning. *Machine Learning, 28*, 75 :41–75, 1997. ISSN 1573-0565. doi : 10.1023/A:1007379606734.

Theodoros Evgeniou and Massimiliano Pontil. Regularized Multi-task Learning. (August 2015), 2004. doi : 10.1145/1014052.1014067. URL http://eprints.pascal-network.org/archive/00000825/.

Ofer Dekel, Joseph Keshet, and Yoram Singer. Large Margin Hierarchical Classification. 2004. doi : 10.1145/1015330.1015374. URL http://eprints.pascal-network.org/archive/00000055/.

Siddharth Gopal and Yimming Yang. Bayesian models for large-scale hierarchical classification. *Advances in Neural Information Processing Systems*, pages 1–9, 2012. ISSN 10495258. URL http://papers.nips.cc/paper/4609-bayesian-models-for-large-scale-hierarchical-classification.

Siddharth Gopal and Yiming Yang. Recursive regularization for large-scale classification with hierarchical and graphical dependencies. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*, page 257, 2013. ISSN 1556472X. doi : 10.1145/2487575.2487644. URL http://dl.acm.org/citation.cfm?doid=2487575.2487644.

Dengyong Zhou and Lin Xiao. Hierarchical classification via orthogonal transfer. *ICML (2011)*, pages 1–6, 2011. URL http://research.microsoft.com/pubs/148332/ohsvm_icml2011.pdf.

Paul N. Bennett and Nam Nguyen. Refined experts. *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '09*, page 11, 2009. doi : 10.1145/1571941.1571946. URL http://portal.acm.org/citation.cfm?doid=1571941.1571946.

Jason Weston, A Makadia, and Hector Yee. Label partitioning for sublinear ranking. *International Conference on Machine Learning*, 28, 2013. URL http://machinelearning.wustl.edu/mlpapers/papers/weston13.

Kilian Weinberger and Olivier Chapelle. Large margin taxonomy embedding with an application to document categorization. *Neural Information Processsing Systems*, pages 1–8, 2008. URL http://www.cse.wustl.edu/~kilian/papers/taxo_nips.pdf.

Carlos N. Jr Silla and Alex A. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 44 (0), 2011. URL http://www.jstor.org/stable/2981629http://link.springer.com/article/10.1007/s10618-010-0175-9.

Shai Shalev-Shwartz and Yoram Singer. On the equivalence of weak learnability and linear separability : New relaxations and efficient boosting algorithms. *Machine learning*, 2010. URL http://link.springer.com/article/10.1007/s10994-010-5173-z.

Marcin Marszalek and Cordelia Schmid. Constructing Category Hierarchies for Visual Recognition. *ECCV*, 2010.

Andrea Passerini, Massimiliano Pontil, and Paolo Frasconi. New results on error correcting output codes of kernel machines. *IEEE transactions on neural networks*, 15 (1) :45–54, 2004. ISSN 1045-9227. doi : 10.1109/TNN.2003.820841. URL http://www.ncbi.nlm.nih.gov/pubmed/15387246.

Pei-yi Hao, Jung-Hsien Chiang, and Yi-Kun Tu. Hierarchically SVM classification based on support vector clustering method and its application to document categorization. *Expert Systems with Applications*, 33 :627–635, 2007. ISSN 09574174. doi : 10.1016/j.eswa.2006.06.009.

Ioannis Partalas, Aris Kosmopoulos, Nicolas Baskiotis, Thierry Artieres, George Paliouras, Eric Gaussier, Ion Androutsopoulos, Massih-reza Amini, and Patrick Galinari. LSHTC : A Benchmark for Large-Scale Text Classification. pages 1–9, 2015.

Klaas Dellschaft and Steffen Staab. On How to Perform a Gold Standard Based Evaluation of Ontology Learning. *International Semantic Web Conference*, 2006.

Yair. Even-Zohar and Dan Roth. A sequential model for multi-class classification. *EMNLP*, 2001. URL http://arxiv.org/abs/cs/0106044.

Shouyuan Chen, Tian Lin, Irwin King, Michael R. Lyu, and Wei Chen. Combinatorial Pure Exploration of Multi-Armed Bandits. *Neural Information Processsing Systems*, pages 1–9, 2014a. URL http://papers.nips.cc/paper/5433-combinatorial-pure-exploration-of-multi-armed-bandits.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3 :397–422, 2003. URL http://dl.acm.org/citation.cfm?id=944941.

Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *CoRR*, abs/1402.6028, 2014. URL http://arxiv.org/abs/1402.6028.

Jean-Yves Audibert and Sébastien Bubeck. Best arm identification in multi-armed bandits. *Conference of Learning Theory*, 2010. URL http://hal.archives-ouvertes.fr/hal-00654404/.

Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On the complexity of best arm identification in multi-armed bandit models. *CoRR*, abs/1407.4443, 2014. URL http://arxiv.org/abs/1407.4443.

Marta Soare, Alessandro Lazaric, and Rémi Munos. Best-arm identification in linear bandits. In *Advances in Neural Information Processing Systems 27 : Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 828–836, 2014.

Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. Online optimization in x-armed bandits. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing*

Systems, Vancouver, British Columbia, Canada, December 8-11, 2008, pages 201–208, 2008.

Shouyuan Chen, Tian Lin, Irwin King, Michael R. Lyu, and Wei Chen. Combinatorial pure exploration of multi-armed bandits. In *Advances in Neural Information Processing Systems 27 : Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 379–387, 2014b.