Table des matières

Table des figures

Introduction.

Au cours des années récentes, la prolifération de dispositifs computationnels numériques et de leur utilisation dans la communication, a produit une production croissante et une grande disponibilité de textes dans divers domaines de l'activité humaine en produisant une importante quantité de données textuelles. Ce phénomène a rendu nécessaire le développement de techniques permettant d'analyser ces données en recherchant des patrons utiles et non triviaux qui seraient impossibles de trouver par une recherche « à main » effectuée par des personnes.

On considère, par exemple, le contexte scientifique dans lequel il y a chaque fois une plus grande génération d'articles scientifiques et didactiques avec contenu de texte numérique, tandis que les bibliothèques numérisent de leur côté leurs patrimoines de livres et autres documents avec contenu de texte. L'analyse de ce type de sources peut représenter une grande occasion de recherche comme l'a démontré Don R. Swanson [79] en utilisant des techniques de fouille de texte pour trouver des relations entre des symptômes, des drogues et leurs effets à partir des titres et des résumés d'articles scientifiques médicaux de l'entrepôt Medline [14], qui, à la fin de l'année 2013, contenait plus de vingt-deux millions d'articles et dont la croissance se poursuit de manière exponentielle [34].

Toutefois, dans beaucoup d'autres domaines, l'utilisation de l'analyse automatique de texte n'a pas été encore grandement exploitée et elle constitue un secteur d'occasion pour les organisations dont le 80% de leur information correspond à des

documents de texte[77]. De même, comme le mentionne [3] « il n'est pas difficile de voir comment presque toute affaire pourrait obtenir des bénéfices éventuels de la capacité d'analyser les documents de millions de personnes pour identifier des désirs ou des nécessités de divertissement, de repas, de voyages, vente au détail et pratiquement toute chose ».

Par ailleurs, les chercheurs en sciences sociales peuvent utiliser les sources de données de texte pour découvrir des patrons intéressants comme l'ont fait des chercheurs du Vermont [20] qui ont construit un outil pour mesurer le niveau de bonheur d'une communauté par rapport aux mots contenus dans les messages twitter de ses membres.

L'analyse automatique de texte est, habituellement, à caractère multidisciplinaire, c'est-à-dire qu'elle inclut, en plus des disciplines reliées au type particulier d'application, des disciplines telles que la statistique, la récupération d'information (information retrieval), la linguistique computationnelle, le traitement du langage naturel et l'apprentissage automatique. Ce dernier, s'est démarqué dans les dernières années par l'apparition de nouvelles approches et stratégies qui lui ont permis de développer des applications capables d'effectuer des tâches pratiques telles que, par exemple, la classification automatique de spam, ce qui est pourvu ordinairement aujourd'hui par les fournisseurs de service de courrier électronique.

La catégorisation de texte.

D'après [46, 65], référés par [13] ,« De nos jours la catégorisation des textes est une discipline aux carrefours de l'apprentissage automatique (Machine learning) et de la recherche d'information (Information retrieval) et partage certaines des caractéristiques avec d'autres tâches comme l'obtention d'information / connaissance à partir de textes et la fouille de textes (Text Mining) ».

Le but principal de la catégorisation des textes est la classification des

> documents dans un nombre fixe de catégories prédéterminées. Chaque document sera dans multiples catégorie, ou dans une, ou dans aucune. Utilisant l'apprentissage automatique, le but principal est d'apprendre des classificateurs à partir des exemples qui effectuent l'assignation de catégories automatiquement [13].

La classification automatique de texte est actuellement utilisée dans différents types de tâches, telles que le filtrat de documents par importance, l'organisation de documents, la création automatique de métadonnées, l'analyse de l'ambiguïté des mots et l'indexation par ordre de documents selon le vocabulaire utilisé [73, 13, 25]. En plus de pouvoir constituer elle-même une technique de fouille de texte, la classification automatique de texte peut s'avérer être une étape des techniques de fouille de texte plus complexes.

L'objectif de la recherche.

L'objectif du présent travail est d'explorer la performance de deux techniques des plus récentes pour la classification automatique de texte : les machines à support de vecteurs (MSV ou SVM) et les forêts de décision (FD ou RF), ainsi que d'évaluer leur pertinence pour l'analyse automatique de documents de texte. On réalise une application capable de classer différents documents ou segments de texte en fonction de la catégorie à laquelle ils appartiennent, et ce, en utilisant les deux différents modèles dont la nature nous permettra d'utiliser le résultat pour explorer le contenu central ou caractéristique des documents ou segments de chaque catégorie. On développe aussi un logiciel ayant une interface graphique qui sera intégré au logiciel REGASS[17] [1]. Pour cela on ajoute les modèles d'apprentissage mentionnés comme nouveaux outils disponibles du programme. On réalise, finalement, l'expérimentation pour mesurer la pertinence des résultats des outils implémentés.

1. Développé à l'UQTR.

Le reste de ce mémoire se déroule comme suit . Le chapitre un décrit l'évolution de l'apprentissage automatique. Il présente aussi quelques méthodes de classifications de l'état de l'art pour comprendre les raisons de l'élection des modèles utilisés dans notre recherche. Le chapitre deux décrit la structure commune du processus d'optimisation de tous les modèles d'apprentissage automatique. Nous poursuivons avec le chapitre trois, dans lequel on explique la procédure de vectorisation de texte pour rendre possible son utilisation par les modèles de classification. Dans les chapitres quatre et cinq, on présente les modèles de machines à support de vecteurs et de forêt de décision, respectivement. Dans le chapitre six, on décrit le développement de l'outil et, dans le sept, ses fonctionnalités. Nous procédons, dans le chapitre huit, à l'expérimentation du logiciel développé avec l'analyse automatique d'un texte concret. Enfin, dans le dernier chapitre, nous concluons par une synthèse des résultats obtenus et des possibles démarches de recherche future.

Passons maintenant au premier chapitre dans lequel on présente une perspective historique de l'apprentissage automatique jusqu'à l'arrivée au paradigme actuel qui a permis l'apparition de nouveaux modèles dont certains seront présentés ici.

Chapitre 1

L'état de l'art.

1.1 Les antécédents.

L'apprentissage automatique a été considéré comme étant une branche de l'intelligence artificielle. En prenant ceci en considération, on pouvait faire remonter l'origine de l'apprentissage automatique, d'après [71], à 1943, quand W. McCulloch et W. Pitts ont effectué le premier travail reconnu en IA dans lequel ils ont proposé un modèle constitué par des neurones artificiels au sein duquel chaque neurone se caractérisait comme étant activé ou désactivé ; ces auteurs introduisent, avec cette publication, un des paradigmes les plus importants dans le domaine de l'intelligence artificielle : le paradigme de modélisation neuronale (Neural model paradigm [7]). À partir de ce moment, plusieurs approches ont été développées en essayant différentes manières d'aborder des problématiques diverses.

Un autre paradigme introduit au cours de ces premières années est le paradigme d'acquisition symbolique de concepts.

> Le paradigme utilisait des structures de représentation logiques ou graphiques au lieu des méthodes et des représentations mathématiques ou statistiques. Les systèmes apprenaient des descriptions symboliques

en représentant des connaissances de haut niveau et ils faisaient de fortes hypothèses structurelles sur les concepts à acquérir [7].

Les algorithmes génétiques, introduits par John Holland au début des années 70s [53], se basent sur l'idée de simuler des processus de sélection évolutifs, produits au moyen de mutations aléatoires, pour obtenir des systèmes qui améliorent leur performance ou qui obtiennent un certain apprentissage par cette évolution.

D'après [67], à partir du début des années 80s, les systèmes experts ont été capables de résoudre des cas de problèmes récurrents dans des domaines de connaissance restreinte en requérant d'avoir une connaissance préalable d'application, ainsi que de règles de but particulier qui permettaient des étapes de raisonnement long pour arriver à la connaissance cherchée. Ces systèmes ont commencé à être acceptés par les grandes industries, lesquelles ont trouvé en ces derniers, une manière d'améliorer leur performance et leur efficacité, leur permettant par le fait même d'économiser des frais. À partir de ce moment, l'apparition de nouvelles techniques et de nouvelles approches, capables d'obtenir des très bons résultats, a eu comme conséquence le surgissement de nouvelles disciplines, telle que la fouille de données qui est également devenue une importante industrie elle même.

1.2 L'état de l'art.

Depuis la fin des années 80s, une révolution s'est produite, tant dans le contenu que dans la méthodologie utilisée, dans le domaine de l'Intelligence artificielle et, conséquemment, dans celui de l'apprentissage automatique.

Cette réalisation a été rendue possible principalement grâce à l'incorporation de matières comme la théorie du contrôle ou de la statistique en faisant que l'IA fasse déjà partie des méthodes scientifiques. Par conséquent, actuellement, pour soutenir les hypothèses, celles-ci doivent être soumises à des expériences empiriques et les

résultats doivent statistiquement être analysés pour identifier leur importance [9] cité par [67]. À noter que l'utilisation de l'internet et la répartition de dépôts de code source de test permettent de répliquer les expériences[67].

David McAllester [51], cité par [67], indique clairement cet important changement :

> Durant les premières années de l'IA il paraissait parfaitement possible que les nouvelles formes du calcul symbolique, par exemple les cadres et les réseaux sémantiques, fassent que la plus grande partie de la théorie classique devienne désuète. Ceci a porté à lclassique'IA à une espèce d'isolement qui l'a séparée des sciences informatiques. Actuellement on abandonne cet isolement. Il existe la croyance que l'apprentissage automatique ne doit pas être séparé de la théorie de l'information, que le raisonnement incertain ne doit pas être séparé des modèles stochastiques, que la recherche ne doit pas être isolée de l'optimisation classique et le contrôle, et que le raisonnement automatique ne doit pas se séparer des méthodes formelles et de l'analyse statistique. Depuis la fin des années 80s, une révolution s'est produite, tant dans le contenu que dans la méthodologie utilisée, dans le domaine de l'Intelligence artificielle et, conséquemment, dans celui de l'apprentissage automatique.

Cette réalisation a été rendue possible principalement grâce à l'incorporation de matières comme la théorie du contrôle ou de la statistique en faisant que l'IA fasse déjà partie des méthodes scientifiques. Par conséquent, actuellement, pour soutenir les hypothèses, celles-ci doivent être soumises à des expériences empiriques et les résultats doivent statistiquement être analysés pour identifier leur importance [9] cité par [67]. À noter que l'utilisation de l'internet et la répartition de dépôts de code source de test permettent de répliquer les expériences[67].

David McAllester [51], cité par [67], indique clairement cet important change-

ment :

> Durant les premières années de l'IA il paraissait parfaitement possible que les nouvelles formes du calcul symbolique, par exemple les cadres et les réseaux sémantiques, fassent que la plus grande partie de la théorie classique devienne désuète. Ceci a porté à lclassique'IA à une espèce d'isolement qui l'a séparée des sciences informatiques. Actuellement on abandonne cet isolement. Il existe la croyance que l'apprentissage automatique ne doit pas être séparé de la théorie de l'information, que le raisonnement incertain ne doit pas être séparé des modèles stochastiques, que la recherche ne doit pas être isolée de l'optimisation classique et le contrôle, et que le raisonnement automatique ne doit pas se séparer des méthodes formelles et de l'analyse statistique.

Il est donc important que les nouvelles techniques d'intelligence artificielle et d'apprentissage automatique aient une base théorique mathématique ou statistique ainsi qu'un appui empirique pratique d'application dans de grandes bases de données qui leurs permettent une fonctionnalité robuste dans différents domaines [67].

Finalement, selon [48], un autre facteur qui a promu un grand changement dans la manière d'aborder les tâches d'apprentissage, à partir des années 80s, est l'utilisation de la connaissance préalable comme entrée pour diriger et restreindre le processus d'apprentissage. Cette connaissance préalable est attachée à la disposition de données d'entraînement qui servent d'entrée afin d'effectuer l'apprentissage.

Tout ceci a produit tant la résurgence des anciennes techniques, en utilisant le nouveau paradigme, que le surgissement de nouvelles techniques. Nous mentionnerons certaines d'entre elles.

Il est donc important que les nouvelles techniques d'intelligence artificielle et d'apprentissage automatique aient une base théorique mathématique ou statistique ainsi qu'un appui empirique pratique d'application dans de grandes bases de données

qui leurs permettent une fonctionnalité robuste dans différents domaines [67].

Finalement, selon [48], un autre facteur qui a promu un grand changement dans la manière d'aborder les tâches d'apprentissage, à partir des années 80s, est l'utilisation de la connaissance préalable comme entrée pour diriger et restreindre le processus d'apprentissage. Cette connaissance préalable est attachée à la disposition de données d'entraînement qui servent d'entrée afin d'effectuer l'apprentissage.

Tout ceci a produit tant la résurgence des anciennes techniques, en utilisant le nouveau paradigme, que le surgissement de nouvelles techniques. Nous mentionnerons certaines d'entre elles.

1.2.1 Les Modèles de Markov Cachés.

Les modèles de Markov cachés, HMM par leurs sigles en anglais, se basent sur la supposition que les données observables proviennent d'une chaîne de Markov dont les paramètres sont inconnus et non observables, c'est-à-dire cachés (de là son nom). Par opposition aux modèles de Markov, dans lesquels les états sont visibles en permettant d'utiliser cette information pour estimer la probabilité de transition entre des états, dans les HMM on observe seulement les données résultantes mais pas les états sous-jacents qui les produisent, en permettant seulement l'utilisation de cette information pour estimer les paramètres de la chaîne de Markov sous-jacent. Les algorithmes d'apprentissage automatique qui utilisent cette approche tentent alors d'estimer les paramètres de la chaîne de Markov sous-jacente génératrice des données, pour ainsi essayer de prévoir les données suivantes en connaissant une séquence de données observées.

Ces modèles ont été utilisés avec grand succès dans le cadre de la reconnaissance vocale. On peut citer la publication très connue de Lawrence R. Rabiner (1989) sur l'utilisation des Modèles de Markov Cachés pour la reconnaissance vocale [69]. Ils ont été aussi utilisés dans d'autres types de contextes comme la détection d'anomalies

[75], soit afin de détecter la fraude bancaire ou des intrusions, soit pour la simulation du comportement humain dans les jeux de vidéo [78] ou encore dans la traduction automatique [50] entre autres.

1.2.2 L'analyse discriminante.

Ce modèle est utilisé pour classer automatiquement en K catégories. On modélise les données de chaque classe comme étant générées par une loi de probabilité connue, par exemple la loi normale, dont les paramètres doivent être estimés avec l'ensemble de données d'entraînement appartenant à la classe, puis, pour la classification, c'-est-à-dire, l'assignation de l'étiquette de classe $\hat{y}$ aux nouveaux exemples $\mathbf{x}$ on utilise le théorème de Bayes comme suit :

$$\hat{y} = \underset{k\in\{1,\ldots,K\}}{argmax} P(Y = k \mid \mathbf{x}); \text{ où } P(Y = k \mid \mathbf{x}) = \frac{\pi_k f_k(\mathbf{x})}{\sum_{l=1}^{K} \pi_l f_l(\mathbf{x})} \tag{1.1}$$

où π_k est la probabilité à priori qu'une observation aléatoirement choisie vient de la k-ème classe et qui peut être estimée avec un échantillon aléatoire de valeurs Y de la population, et $f_k(\mathbf{x})$ dénote la fonction de densité de probabilité pour une observation $\mathbf{x}$ qui vient de la k-ème classe.

1.2.2.1 Le modèle bayésien naïf (naïve Bayes).

C'est un cas particulier de l'analyse discriminante, dans lequel on assume l'indépendance conditionnelle des caractéristiques. Cette supposition permet que l'expression de la distribution $f_k(\mathbf{x})$ soit plus simple en réalisant un calcul plus facile et rapide.

Intuitivement, si les caractéristiques sont représentées par des mots, il s'avère facile de penser que cette supposition est fausse, étant donné que certains mots peuvent être corrélés dans le discours de chaque classe ; c' est pour cette raison qu'on

emploie le terme naïf.

Malgré ses fortes suppositions, le modèle bayésien naïf est robuste et fonctionne généralement bien. Une justification théorétique de la robustesse des modèles bayésiens naïfs est donnée par [21] référé par [25].

1.2.3 La régression logistique.

C'est un vieux modèle statistique de classification qui a été redécouvert et qui a récemment gagné une grande popularité grâce à sa bonne performance en ce qui a trait à la classification automatique.

Ce modèle permet de calculer la probabilité d'appartenance à la catégorie k, $Pr(Y = k|X = x)$, comme suit :

$$P(Y = k \mid \mathbf{x}) = \frac{e^{\beta \mathbf{x}}}{1 + e^{\beta + x}}.$$

où β est le vecteur de coefficients de régression qui doivent être estimés avec des exemples d'entraînement en utilisant, par exemple, la méthode de moindres carrés. La catégorie assignée sera celle dont la probabilité est la plus grande, c'est-à-dire :

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{argmax} P(Y = k \mid \mathbf{x})$$

La régression logistique a bénéficié de beaucoup de travail de recherches et est devenue un modèle pratique dans de nombreux systèmes commerciaux à grande échelle, en particulier aux grandes sociétés d'Internet comme Google et Yahoo qui l'emploient pour apprendre de grands ensembles de données [36, 16].

Ce modèle, en plus de pouvoir être utilisé seul, constitue en outre le bloc fondamental des réseaux neuronaux.

1.2.4 Les classeurs paresseux (lazy learners).

Ils font le calcul direct de la similitude des exemples à classer et de l'ensemble de données d'exemple, appelé l'ensemble d'entraînement. Son entraînement consiste simplement à garder les représentations des données d'entraînement avec ses étiquettes de catégorie.

Le classeur des k voisins les plus proches est un exemple largement connu de ce type de classificateurs. Pour décider si un document $\mathbf{x}$ appartient à la catégorie c, cet algorithme regarde, entre les données d'entraînement, les k documents les plus semblables (les voisins plus proches) à $\mathbf{x}$ et la catégorie à laquelle ceux-ci appartiennent en assignant à $\mathbf{x}$ la catégorie ayant la plus grande fréquence entre les k voisins les plus proches, c'est-à-dire, à laquelle appartient la plus grande proportion d'entre eux.

Pour utiliser l'algorithme, on doit définir au début le nombre de k voisins à utiliser pour la classification. Le nombre optimal, peut être trouvé en utilisant l'une des méthodes de validation (Voir la section 2.4.7). D'autre part, [25] fait référence à des tests empiriques montrant que le choix de $30 \leq k \leq 45$ donne la meilleure efficacité. Le même auteur mentionne que diverses expériences ont montré qu'augmenter la valeur de k ne dégrade pas de manière significative la performance et que c'est l'un des classificateurs les plus performants des textes disponibles aujourd'hui, car il est très robuste parce qu'il n'exige pas que les catégories soient linéairement séparées.

1.2.5 Les arbres et forêts de décision.

À la fin des années 70s et au début des années 80s, J. Ross Quinlan, un chercheur dans l'apprentissage automatique, a développé un algorithme d'arbre de décision connu sous le nom d'ID3 (Iterative dichotomiser). En 1984, un groupe de statisticiens (L. Breiman, J. Friedman, R. Olshen, et C. Stone) a publié un livre sur les arbres de classification et de régression (CART), décrivant la génération d'arbres de décision binaires. Des améliorations postérieures ont été faites tels que l'algorithme C4.5,

successeur de l'ID3, CHi-squared Automatic Interaction Detector (CHAID) [29] .

Les arbres de décision sont facilement interprétables, toutefois, la capacité de prévision qu'ils ont est presque toujours dépassée par les autres modèles de classification. Cette caractéristique a limité son utilisation jusqu'au début des années 2000, puis ils ont été repris comme élément de base d'une nouvelle technique, appelée la forêt de décision. Cette nouvelle technique utilise de manière combinée les arbres de décision et la théorie statistique pour réduire la variance du classeur en calculant la moyenne d'un ensemble d'arbres de décision en générant des classeurs avec une très bonne capacité de prévision.

Les arbres de décision ont été utilisés dans différentes tâches comme la classification d'images [4] et la détection humaine en temps réel [11].

1.2.6 Les machines à vecteurs de support (SVM).

Elles ont été inventées par Boser, Guyon et Vapnik [5, 10] et présentées pour la première fois dans la conférence Computational Learning Theory (COLT) de 1992.

Les SVM utilisent une approche géométrique pour classer les données en deux catégories. En modelant les données comme des points (vecteurs) dans l'espace, elles construisent un plan qui sépare les données dans chacune des catégories.

Une fois la frontière de décision construite, la SVM sera capable de classer de nouvelles données en observant de quel côté de la frontière elles tombent, et en leur assignant la catégorie correspondante. Finalement, l'utilisation des SVM peut facilement être étendue à la classification de plus de deux catégories ainsi qu'à la prévision de valeurs continues.

Les machines à support de vecteurs ont été appliquées avec succès dans divers domaines comme la vérification et la reconnaissance, telle que l'identification de visages, [28, 61, 70], la reconnaissance de caractères manuscrits et des chiffres [63], la vérification et reconnaissance du discours et du parlant [22, 82] et la prédiction et

le pronostic[37, 74, 30, 85].

Les SVM constituent une classe spécifique d'algorithmes qui est caractérisée par, d'après [12], l'utilisation de kernels (fonctions noyau), la capacité d'arriver à un résultat optimal global, la faible densité de la solution et sa capacité de contrôle en agissant sur la marge ou sur une autre quantité « indépendante de la dimension » comme le nombre de vecteurs de support. Ces caractéristiques font en sorte qu'elle soit une des techniques dont l'application peut être très adéquate pour la classification de texte, comme nous verrons dans la section 4 où on va expliquer les SVM en détails.

1.2.7 Les réseaux de neurones artificiels.

Comme il a été mentionné au début du chapitre, les premiers modèles de réseaux de neurones ont été introduits en 1943 par les neurologues McCulloch et Pitts. Toutefois, la technologie de l'époque ne leur a pas permis d'obtenir beaucoup des progrès. D'autres chercheurs comme Donald Hebb, qui a présenté en 1949 une série d'idées sur la structure et le fonctionnement des systèmes biologiques de neurones [45] et Frank Rosemblatt, qui a développé entre 1957 et 1959 [76, 80] le perceptron, un algorithme neuronal simple, ont contribué au développement de ce type d'algorithmes. Toutefois on a dû attendre le milieu des années 1980 pour que cette approche acquiert une nouvelle force, grâce à l'algorithme d'apprentissage de rétro-propagation (BackPropagation) introduit par Rumelhart et McClelland en 1986, à partir duquel ils ont montré que les réseaux de neurones de multiples couches ont une capacité exceptionnelle de discrimination en étant capables d'apprendre des patrons complexes [64].

Pour comprendre le fonctionnement des réseaux de neurones artificiels, il est utile de savoir comment fonctionnent les neurones naturels.

Les neurones naturels reçoivent des signaux par des synapses situées sur

> les dendrites ou membrane du neurone. Quand les signaux reçus sont assez forts (surpassant un certain seuil), le neurone est activé et émet un signal à travers l'axone. Ce signal pourrait être envoyé à une autre synapse, et pourrait activer d'autres neurones.[27].

Inspirés de ce mécanisme, les réseaux de neurones artificiels sont représentés par des nœuds qui constituent les « neurones artificiels ». Chaque neurone reçoit

> des signaux d'entrée (comme les synapses), pondérées par des poids (intensité des signaux respectifs) et puis calcule, par une fonction mathématique, l'activation (ou non) du neurone. Une autre fonction (qui peut être l'identité) calcule la valeur de sortie du neurone artificiel (parfois dépendant d'un certain seuil). Les réseaux de neurones artificiels combinent des neurones artificielles pour traiter l'information [27].

Le modèle réseau de neurones artificiels est entraîné avec des données d'exemple. Après l'entraînement, certains groupes de neurones seront activés en reconnaissant certains des patrons appris avec les données d'exemple, leur permettant ainsi de faire des prévisions de nouvelles données, encore non vues, en imitant ainsi le fonctionnement des neurones biologiques.

Une des caractéristiques principales des réseaux de neurones est leur capacité d'apprendre des relations complexes non linéaires d'entrée et de sortie, en utilisant des procédures séquentielles. Ils sont, en outre, capables de s'adapter aux données.

1.2.8 Les modèles d'ensemble.

Beaucoup parmi les méthodes d'apprentissage, comme les machines à vecteurs de support et les arbres de décision, se basent sur la recherche de la meilleure hypothèse h dans un espace de possibles hypothèses H, où chaque hypothèse correspond à un modèle candidat, et ce, jusqu'à que soit trouvé le modèle optimal du problème d'apprentissage.

D'après Dietterich[19] :

> Les algorithmes d'ensemble adoptent une approche différente. Plutôt que trouver la meilleure hypothèse pour expliquer les données, ils construisent un ensemble avec des hypothèses (parfois appelées un « comité » ou « ensemble ») et puis obtiennent ces hypothèses « en votant » d'une certaine façon pour prévoir la valeur des nouveaux points de données.
>
> Plus précisément, une méthode de comité construit un ensemble d'hypothèses $\{h_1, h_2, \ldots, h_k\}$, choisit un ensemble de poids $\{w_1, w_2, \ldots, w_k\}$ et construit le classeur « voté »

$$h_{ens} = w_1 h_1(\mathbf{x}) + w_2 h_2(\mathbf{x}) + \cdots w_k h_k(\mathbf{x}).$$

Le diagramme 1.1 illustre le fonctionnement des méthodes.

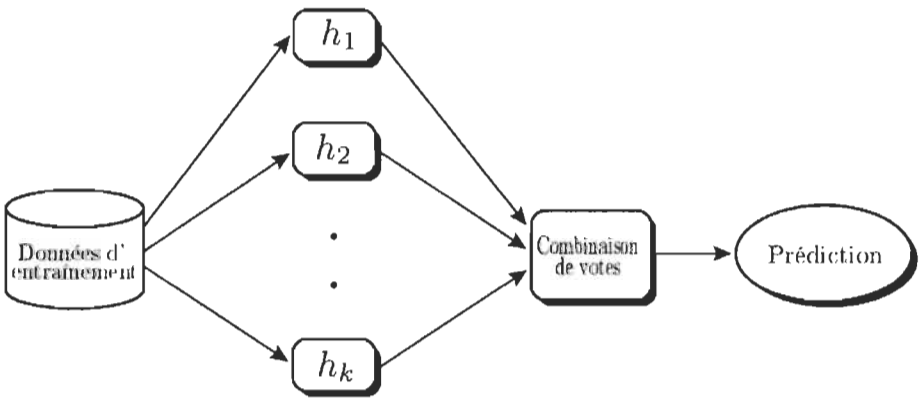


FIGURE 1.1 – Diagramme de fonctionnement des méthodes d'ensemble.[29]

D'après [19], les modèles d'ensemble, ou comité, ont montré, de manière expérimentale, être capables d'obtenir une meilleure performance que l'élection d'une hypothèse unique. Le même auteur indique que ces types de méthodes peuvent résoudre les problèmes des méthodes qui cherchent une seule hypothèse :

Le problème statistique apparaît quand l'espace d'hypothèse est trop grand pour l'ensemble de données d'entraînement. Comme résultat, il pourrait y avoir beaucoup d'hypothèses différentes ayant la même précision avec les données d'entraînement, et, de fait, l'algorithme d'apprentissage doit choisir une parmi elles comme résultat. Comme conséquence, il existe le risque que l'hypothèse choisie ne puisse pas avoir, de manière très précise, le résultat de nouvelles données. Une hypothèse construite par une pondération de votes de diverses hypothèses peut réduire ce risque-là [19].

Le problème computationnel apparaît quand l'algorithme d'apprentissage ne peut pas trouver la meilleure hypothèse possible dans l'espace d'hypothèse ; ceci arrive, par exemple, avec les réseaux de neurones et les arbres de classification, dans lesquels, pour choisir une hypothèse, étant donné la difficulté computationnelle de tester toutes les hypothèses possibles, on utilise des méthodes heuristiques qui peuvent résulter à des minimums locaux. Encore une fois, l'hypothèse de la nature construite par une méthode d'ensemble peut résoudre le problème [19].

Le problème de représentation apparaît quand l'espace d'hypothèse ne contient aucune hypothèse qui soit un bon rapprochement de la fonction objectif (fonction vraie) f. Une hypothèse obtenue par des méthodes d'ensemble peut étendre l'espace d'hypothèse qui est représenté en permettant une plus grande possibilité de trouver une hypothèse résultante qui soit une meilleure approximation à la fonction objectif f.

1.2.9 Les autres modèles.

Il existe d'autres méthodes utilisées pour la classification, comme par exemple les méthodes de régression et les méthodes Roccio. Le lecteur intéressé peut consulter

[25] qui offre une description de l'application de ces méthodes pour la classification de texte.

Les méthodes qu'on a mentionnées sont des méthodes supervisées dans lesquelles le classeur a besoin de connaître la classe d'appartenance des données d'entraînement. Il existe des méthodes non supervisées, comme les cartes auto adaptatives (Self organized maps SOM) ou cartes de Kohonen [47], qui sont une version non supervisée des réseaux de neurones artificiels, et les méthodes de conglomération (clustering methods), lesquelles permettent la classification sans connaître l'appartenance des données d'entraînement et en générant elles-mêmes les étiquettes de classe. Étant donné que dans ce travail les algorithmes de classification qui seront mis en œuvre sont des méthodes supervisées, on se concentrera sur ce type de méthodes, en laissant de côté les techniques non supervisées. Le lecteur intéressé peut aussi consulter [25] qui consacre un chapitre aux techniques de conglomération appliquées à la classification du texte.

1.3 Pourquoi la SVM et la forêt de décision.

Après avoir vu certains des modèles les plus pertinent de l'état de l'art pour la classification, il faut préciser qu'il n'existe pas une règle générale qui permet de savoir quel modèle utiliser. Ceci dépend du type de données disponibles ainsi que du domaine et des objectifs particuliers de l'application. On peut se demander alors pourquoi, dans le cadre de notre recherche, on a choisi les machines à support de vecteurs (SVM) et les forêts de décision.

Tout d'abord, le choix est dû à leur récente apparition par rapport aux autres modèles mentionnés. Par ailleurs, en raison de leurs caractéristiques, comme on verra dans les chapitres correspondant à chacun des deux modèles, ceux-ci sont particulièrement bons pour la classification de documents de texte en permettant, en outre,

l'identification des mots plus importants pour la classification dans chacune des catégories. Cette dernière caractéristique est très précieuse puisqu'elle nous permettra d'explorer le contenu des documents de manière automatique, ce qui est le principal objectif de la recherche. De plus, à la différence des autres modèles présentés, ils n'ont pas besoin d'assumer aucune fonction de densité de probabilité comme étant la fonction génératrice des données, ce qui peut amener à des mauvaises résultats si cette assomption n'est pas proche à la réalité. Finalement, notre intérêt est de les implémenter dans un logiciel de façon qu'ils soient disponibles dans le logiciel REGASS.

1.4 Résumé.

On a présenté, de manière chronologique, l'évolution des modèles d'apprentissage automatique de 1943 jusqu'à l'arrivée de l'approche actuelle fondée sur la modélisation mathématique statistique et le test empirique. Ce nouveau paradigme a produit l'apparition de nouveaux modèles d'apprentissage automatique dont certains ont été présentés ici pour donner une vision générale des modèles de classification les plus significatifs de l'état de l'art actuel. Finalement on justifie l'élection des deux modèles qui seront utilisés dans notre recherche. Il est nécessaire, cependant, de comprendre la structure générale des modèles d'apprentissage automatique, laquelle constitue une espèce de cœur commun utilisé par tous les modèles. C'est donc l'objectif du prochain chapitre.

Chapitre 2

L'apprentissage statistique automatique.

Dans ce chapitre on présente la structure générale de l'apprentissage statistique automatique.

2.1 L'approche statistique.

D'après [18, 23], référés par [40] :

> Dans l'approche statistique, chaque exemple est représenté par d caractéristiques et est observé comme un point dans un espace d-dimensionnel. Le but est de choisir les caractéristiques permettant aux vecteurs, appartenant à différentes catégories, d'occuper des régions disjointes dans l'espace de caractéristiques d-dimensionnel. L'efficacité de l'espace de représentation (ensemble de caractéristiques) est déterminée par la façon dans laquelle les exemples de différentes classes peuvent être séparés. Étant donné un ensemble d'exemples de chaque classe, l'objectif est d'établir les frontières de décision dans l'espace

de caractéristiques qui peuvent séparer les exemples appartenant à différentes classes. Dans l'approche théorétique de décision statistique, les frontières de décision sont déterminées par les distributions de probabilité du modèle d'appartenance à chaque classe, qui doit être spécifié ou appris.

2.2 Le processus de décision.

On résume comme suit le processus de décision décrit par [40]) :

On veut faire une classification dans l'une de c catégories $\omega_1, \omega_2, \ldots, \omega_c$ en se basant sur le vecteur de caractéristiques des exemples $\mathbf{x} = (x_1, x_2, \ldots, x_d)$. On assume que les caractéristiques possèdent une fonction de densité de probabilité ou de masse de probabilité conditionnée à la classification des exemples. Ainsi, un exemple $\mathbf{x}$, de la classe ω_i, est considéré comme une observation aléatoire de la fonction de probabilité, conditionnée par classe $p(\mathbf{x} \mid \omega_i)$. Il est alors possible d'utiliser l'une des règles de décision comme celle de Bayes, celle de la vraisemblance maximale ou celle de Neyman-Pearson pour définir une frontière de décision dans le but de diminuer le risque (valeur attendue de la fonction de perte). Par exemple, en utilisant la règle de Bayes, ceci peut être déclaré comme : Assigner l'exemple $\mathbf{x}$, à la classe ω_i dont le risque conditionnel

$$J(\omega_i \mid \mathbf{x}) = \sum_{j=1}^{c} L(\omega_i, \omega_j) \cdot P(\omega_j \mid \mathbf{x}) \tag{2.1}$$

est minimal ; $L(\omega_i, \omega_j)$ est la perte commise en décidant ω_i quand la véritable classe est ω_j et $P(\omega_j, \mathbf{x})$ est la probabilité postérieure [23], référée par [40].

Les composants x_i des exemples représentent des valeurs scalaires des caractéristiques, ou des attributs dans le cas des variables discrètes.

La fonction J à optimiser est aussi appelée fonction objectif, de coût, de perte ou fonction d'énergie [15] ; une fois optimisée, on dit qu'on a un modèle entraîné capable

de prévoir des nouveaux exemples.

2.3 Le choix du modèle.

La fonction objectif à optimiser est attachée au modèle choisi. Le choix du modèle dépend, essentiellement, du but de la recherche ainsi que du type et de la quantité de données disponibles.

2.3.1 L'apprentissage paramétrique ou non paramétrique.

Si on suppose, ou on connaît, la densité de probabilité conditionnelle de classe de la fonction objectif, il est possible d'utiliser des techniques d'apprentissage paramétriques, lesquelles se basent sur l'estimation des paramètres de la fonction de densité pour optimiser la fonction objectif J. Autrement dit, si cette loi de probabilité n'est pas connue, ou bien qu'il n'est pas possible de supposer sa distribution, on doit alors recourir à des méthodes d'optimisation non paramétriques, lesquelles ne se basent pas sur des densités de probabilité.

2.3.2 L'apprentissage supervisé ou non supervisé.

Si on connaît la classe à laquelle appartiennent les données d'entraînement, on peut alors utiliser des techniques d'apprentissage supervisé. Dans le cas contraire, on peut utiliser des techniques d'apprentissage non supervisé, lesquelles n'ont pas besoin de cette information.

Les deux méthodes que l'on va explorer dans ce travail-ci, les SVM et les arbres de décision, sont des techniques d'apprentissage supervisé non paramétrique, c'est-à-dire, qu'elles ont besoin de recevoir des données d'entraînement dont la classe d'appartenance est connue mais elles ne considèrent pas les observations comme étant générées par une loi de probabilité pour estimer la densité conditionnelle de classe

de la fonction de perte. Elles s'appuient plutôt sur une approche géométrique pour construire des frontières de décision dans l'espace d'origine permettant de classer les données.

On utilise ce schéma de la figure 2.1, publié par [40], pour illustrer les différents choix selon les hypothèses et l'information disponible.

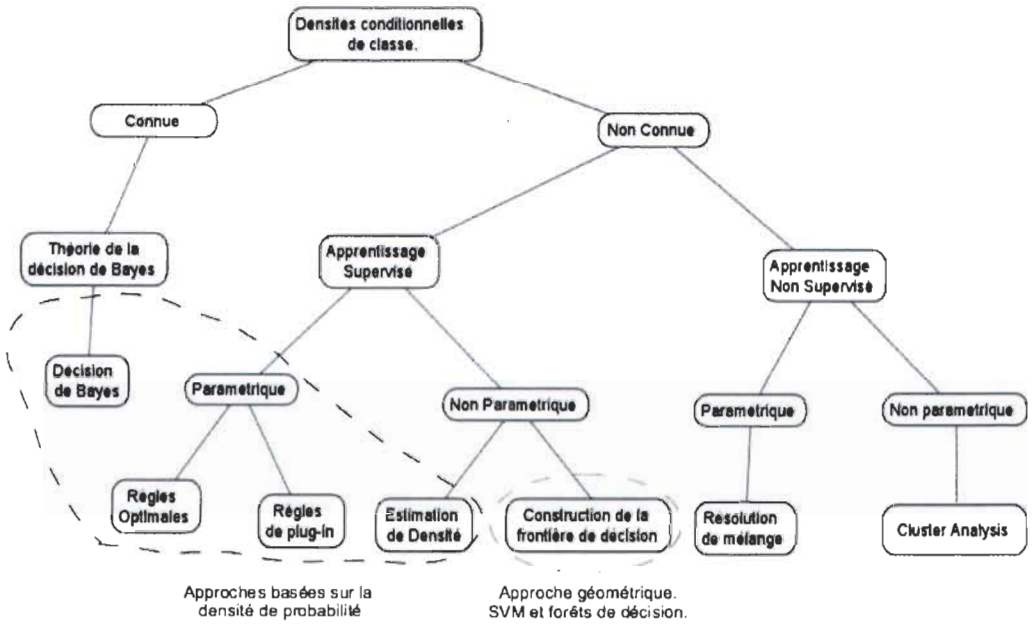


FIGURE 2.1 – Types de modèle.

2.4 L'optimisation du modèle.

Peu importe quelle règle de classification utilisée, l'apprentissage est fait au moyen de l'expérience acquise avec les données d'entraînement. Par conséquent, l'entraînement doit être fait avec les données d'entraînement disponibles.

2.4.1 La capacité de généralisation.

La capacité de généralisation d'un modèle de classification ou de décision, concerne sa performance en classant les données encore non vues par le modèle entraîné.

2.4.2 La complexité du modèle.

Il existe des développements théoriques dans la théorie de l'apprentissage de Vapnik–Chervonenkis, selon lesquels la performance d'un classeur dépend essentiellement du nombre d'exemples d'entraînement disponibles, de sa propre complexité ainsi que de la complexité du classeur.

Selon cette théorie, étant donné l'ensemble de données d'entraînement, la complexité du classeur a une grande influence sur la performance finale de celui-ci. D'une part, un classeur trop complexe, c'est-à-dire, dont la variance et la sinuosité sont trop grandes, s'adaptera excessivement aux données d'entraînement, et il aura par conséquence une pauvre performance en classant des nouvelles données encore non vues. Cela s'appelle le sur-ajustement du classeur aux données. D'autre part, un classeur excessivement lisse est un classeur dont la variance ou la sinuosité sont excessivement basses, et il aura une mauvaise performance, en raison de sa faible de représenter les données d'entraînement ayant, par conséquence, une mauvaise capacité de généralisation. Ce phénomène s'appelle le sous-ajustement du classeur aux données.

Pour illustrer ces deux idées, on analyse rapidement la figure 2.2 de points en $\mathbb{R}^2$: le classeur de complexité (sinuosité) minimale correspondrait à une ligne droite (figure degré 1), tandis qu'un classeur plus complexe, par exemple un classeur polynomial d'un plus grand degré, aurait une capacité majeure de s'adapter aux points de l'ensemble d'entraînement, mais, s'il est excessivement complexe, il sur-ajustera les données d'entraînement (figure degré 15).

On comprendra, par ailleurs, que si les données sont facilement séparables dans l'espace, elles peuvent être classées avec des classeurs plus simples, ayant peu de variance ou de sinuosité, tandis que si les données sont plus complexes, c'-est-à-dire, plus mélangées, il sera nécessaire d'utiliser des classeurs d'une plus grande complexité. Toutefois, pour éviter le sur-ajustement en augmentant la complexité du classeur, il est nécessaire que la quantité de données d'entraînement soit suffi-

samment grande. Sinon, le classeur le plus complexe se sur-adaptera aux données d'entraînement en provoquant une mauvaise capacité de généralisation. Finalement, si le nombre de caractéristiques est trop grand par rapport au nombre de données d'entraînement, généralement, un classeur de haute complexité sera nécessaire pour bien classer les données d'entraînement, mais il pourrait mener à une mauvaise capacité de généralisation. Ce phénomène est appelé la malédiction de la dimensionnalité (curse of dimensionality [39], référé par [40].

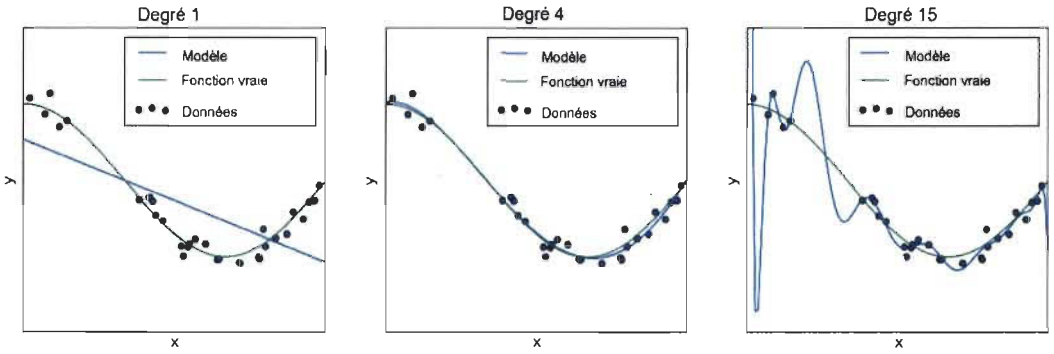


FIGURE 2.2 – Exemple sous-ajustement et sur-ajustement. [44]

2.4.3 Éviter la mauvaise généralisation.

Pour que le modèle de classification atteigne une bonne capacité de généralisation il est nécessaire d'arriver à un bon compromis entre la complexité du classeur d'un côté, et la quantité et la complexité des données de l'autre côté. Pour ce faire, on doit observer les stratégies suivantes.

2.4.4 Soulager le sur-ajustement.

Le sur-ajustement est dû à l'excessive complexité du classeur par rapport aux données d'entraînement. Il y a trois façons de le résoudre :

1. Transformer les données, en augmentant la dimension originale d, en une dimension plus grande, ce qui permettra d'utiliser des classeurs plus complexes.

On peut ajouter des nouvelles dimensions ou caractéristiques, si elles sont disponibles, ou bien carrément les créer à partir des caractéristiques existantes. Par exemple, si on a des données en deux dimensions, chaque donnée d'entraînement aurait la forme (x_1, x_2). Pour augmenter la dimension des données, on peut ajouter trois caractéristiques nouvelles de type polynomiale $x_1^2, x_2^2, x_1 \cdot x_2$. Maintenant l'ensemble de données d'entraînement sera dans un espace de plus haute dimension (de taille 5 au lieu de 2), où chaque point a la forme $(x_1, x_2, x_1^2, x_2^2, x_1 \cdot x_2)$, ce qui permettra d'utiliser un classeur plus complexe.

2. Augmenter le nombre de données d'entraînement. Plus de données sont disponibles, plus grande est la complexité possible du classeur.

3. Diminuer la complexité du classeur en augmentant le valeur du paramètre de régularisation λ (voir sec 2.4.6).

2.4.5 Soulager le sous-ajustement.

Le sous-ajustement est dû à la faible complexité du classeur par rapport aux données d'entraînement. Il y a deux façons de le résoudre :

1. Diminuer la dimension des données en enlevant des caractéristiques insignifiantes ou en utilisant une version comprimée des données d'entraînement, par exemple, par une décomposition en valeurs singulières SVD (Singular Value Decomposition) en projetant les données vers un espace à dimension réduite $k < d$. Dans le nouvel espace, les données pourront être classées avec un classeur moins complexe, améliorant ainsi sa capacité de généralisation.

2. Augmenter la complexité du classeur en réduisant la valeur du paramètre de régularisation λ (sec 2.4.6).

2.4.6 La Régularisation.

La régularisation est utilisée dans les modèles d'apprentissage automatique pour contrôler la complexité des classeurs construits. Elle consiste à ajouter un terme $R(J)$ de pénalisation sur la complexité de la fonction objectif J

$$J(\omega \mid \mathbf{X}) + \lambda R(J)$$

.

Le terme λ, aussi appelé paramètre de régularisation, permet d'augmenter ou diminuer l'importance du terme de pénalisation en variant la complexité du modèle.

L'élection adéquate du paramètre λ est faite en essayant la performance de différentes valeurs de λ par la technique de validation croisée (qui sera expliquée plus loin), en choisissant la valeur dont la performance soit la meilleure.

2.4.7 La validation.

Une autre tâche importante permettant d'éviter le sur-ajustement aux données d'entraînement est d'utiliser un autre ensemble de données appelé ensemble de test. Pour faire un usage optimal des données disponibles, et ainsi éviter d'avoir plusieurs ensembles de données indépendantes, les classeurs sont généralement entraînés avec des sous-ensembles des données en utilisant la validation croisée de k itérations (k fold cross validaton).

La validation croisée de k itérations (k fold validation). Soit n la taille de l'ensemble de données disponible. La validation croisée consiste à diviser aléatoirement les données disponibles en k sous-ensembles de données, de taille n/k, mutuellement exclusives. Ensuite, on entraîne le modèle en utilisant l'un des sous-ensembles produits comme l'ensemble de données de test et les restants $k-1$ comme

l'ensemble de données d'entraînement. On répète cette procédure k fois, en utilisant, à chaque fois un des sous-ensembles comme ensemble de test. Finalement, on obtient la moyenne des k résultats pour produire une évaluation unique de la performance du modèle. Différents modèles peuvent être entraînés de cette manière en variant leur complexité, choisissant finalement celui possédant la meilleure performance. Une autre façon d'utiliser des sous-ensembles des données est la construction de classeurs par la technique de bootsrap agregation ou bagging, qu'on explique dans la section 5.4.1.

On peut alors varier la complexité du classeur par la régularisation et évaluer le résultat en utilisant la validation avec les données d'entraînement. Si le résultat n'est pas optimal, pour améliorer la performance, on peut répéter le processus en augmentant ou en diminuant la complexité du classeur ou des données au besoin.

2.4.8 Les mesures de performance.

Il est nécessaire, finalement, de mesurer la performance du classeur afin de prévoir ou de classer des données, tant avec les données d'entraînement que les données de test. Pour ce faire on présente par la suite certaines des mesures communément utilisées, d'après [42], pour mesurer cette performance.

Pour mesurer la performance d'un classeur, il est nécessaire de définir les quantités suivantes :

- *Positifs vrais (True Positives) (TP)* : Ce sont les exemples positifs qui ont été correctement classés par le classificateur.
- *Négatifs vrais (True Negatives) (TN)* : Ce sont les exemples négatifs qui ont été correctement classés par le classificateur.
- *Faux positifs (FP)* : Ce sont les exemples négatifs qui ont été inexactement classés comme positifs.
- *Faux négatifs (FN)* : Ce sont les exemples positifs qui ont été incorrectement

classés comme négatifs.

Ces mesures peuvent être résumées dans une matrice appelée *matrice de confusion* MC qui est un outil permettant de se rendre compte à quel point le classificateur peut identifier des exemples de différentes classes.

Par exemple, avec un classeur de texte dont la tâche est de classer des courriels, en assignant la valeur 1 numérique dans le cas de spam et de 0 autrement, en supposant qu'on utilise 1000 courriels pour l'entraînement du classeur, on pouvait avoir une matrice de confusion MC comme la suivante :

		Prediction	
		1	0
Actual	1	TP	FP
	0	FN	TN

		Prediction	
		1	0
Actual	1	130	20
	0	30	820

		Prediction	
		1	0
Actual	1	0.13	0.02
	0	0.03	0.82

La matrice de droite exprime les proportions correspondant aux fréquences montrées par la matrice du centre. On observe que pour qu'un classificateur ait une bonne performance, idéalement la plupart des exemples doivent être représentés le long de la diagonale de la matrice de confusion, correspondant à l'entré MC_{ii}.

Une fois la matrice obtenue, il est également possible de calculer certaines mesures communément utilisées pour mesurer et juger de la performance d'un classeur :

Mesure	Formula	
Exactitud (accuracy, recognition rate)	$\frac{TP+TN}{P+N}$	
Précision (error rate, misclassification rate)	$\frac{FP+FN}{P+N}$	
Sensitivité (sensitivity, true positive rate, recall)	$\frac{TP}{P}$	[42]
Specificité (specificity, true negative rate)	$\frac{TN}{N}$	
Exactitude (Accuracy)	$\frac{TP}{TP+FP}$	
$F, F_1 F - score$	$\frac{2 \times precision \times recall}{precision+recall}$	

2.5 Résumé.

Dans ce chapitre on a expliqué la procédure générale de choix et d'optimisation des modèles dans l'apprentissage automatique.

Partant de la représentation vectorielle des exemples, le processus de décision est effectué en utilisant une fonction objectif dont l'optimisation permettra de diminuer le risque empirique, c'est-à-dire, la proportion d'exemples dont la prévision a été erronée.

Le choix du type du modèle (paramétrique, ou non, supervisé ou non) dépendra du type de données ainsi que des objectifs de la recherche.

On a présenté les différentes stratégies pour optimiser le modèle choisi en utilisant la validation pour éviter le sur-ajustement aux données d'entraînement.

Toutefois, dans notre recherche on utilise des documents de texte. Alors, on doit être en mesure de les représenter sous forme de matrice numérique afin de pouvoir utiliser le modèle d'apprentissage choisi. Dans le chapitre suivant, nous abordons ce processus en plusieurs étapes.

Chapitre 3

Le prétraitement et la transformation du texte.

Le processus de classification du texte par des modèles d'apprentissage automatique est essentiellement le même que celui utilisé pour la classification d'un autre type de données. La principale différence, est constituée par le processus de transformation de données pour que celles-ci puissent être passées à l'algorithme de classification comme une représentation vectorielle numérique. Dans cette transformation, il est nécessaire de passer les données du texte pur à une représentation dans laquelle les documents de texte sont numériquement représentés dans une matrice que le classeur peut interpréter. En se basant sur la description de tâches du processus de classification de [13], Illustré par la figure 3.1, on explique ci-dessous les différentes étapes de pré-traitement du texte.

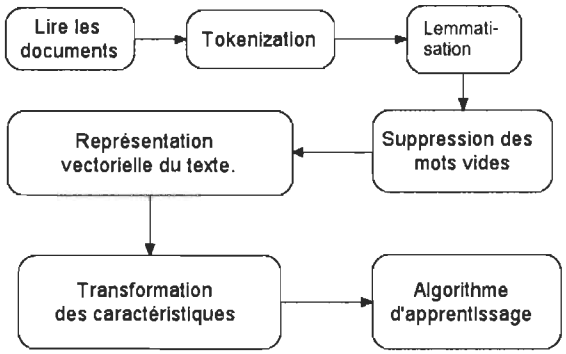


FIGURE 3.1 – Le processus de prétraitement et transformation du texte [13].

3.1 L'extraction de termes (Tokenization).

Cette tâche consiste essentiellement à diviser le texte qui a été lu dans les structures de base pour l'analyse future. Ces structures peuvent être des mots (monogrammes), des ensembles de deux ou plusieurs mots adjacents (bigrammes ou m-grammes), des phrases ou des déclarations, des symboles ou une autre structure de base offrant une information utile pour la classification. Le résultat est une liste de « tokens », correspondant aux mots, bigrammes, etc., séparés par des caractères d'espace simple. Les espaces et les signes de ponctuation du texte original pourraient, ou non, être inclus dans la liste résultante de tokens.

3.2 La lemmatisation ou réduction à la tige (Stemming).

D'après [13] :

> en morphologie linguistique, et dans la recherche d'information (information retrieval), la réduction à la tige est le processus de diminution (ou parfois augmentation) de mots déviés à leur tige forme d'origine. La tige n'a pas besoin d'être identique à la racine morphologique du mot.

Il est habituellement suffisant qu'elle permet de regrouper des mots avec une tige et sens semblable, même si cette tige n'est pas une racine valide.

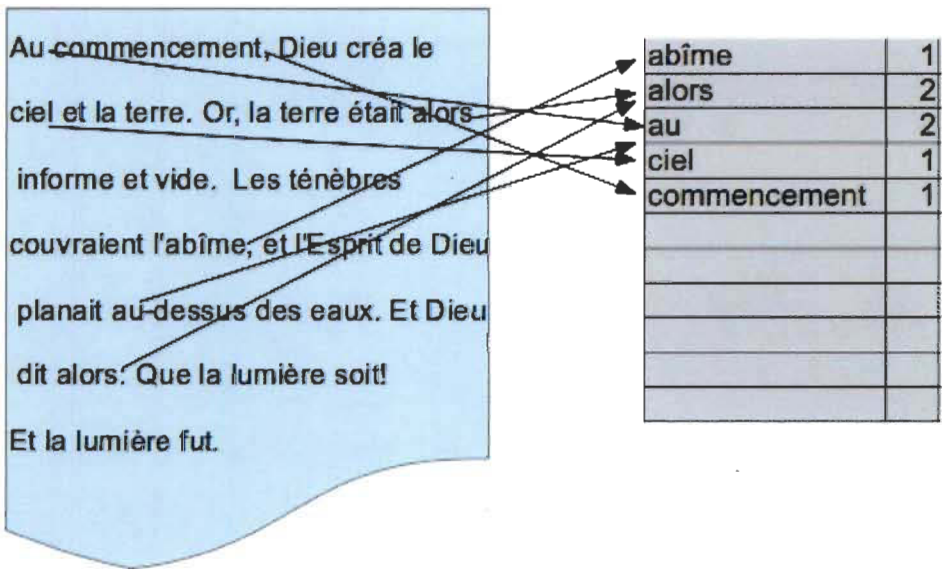
3.3 La suppression de mots fonctionnels (stop words remotion).

Il existe certains mots, appelées fonctionnels, qui apparaissent trop fréquemment dans tout type de texte. Cette particularité fait en sorte que leur présence n'apporte aucune information utile pour la classification du texte. La présence de ces mots peut, au contraire, produire du bruit qui complique la classification précise. C'est la raison pour laquelle il est préférable de supprimer ces mots pour ainsi améliorer la capacité de classification du modèle qui sera postérieurement utilisé. Ce type de mots inclut les connecteurs, les conjonctions, les causes déterminantes, ainsi que des verbes qui figurent fréquemment dans toutes les catégories de classification (par exemple le mot « permet »). Il existe d'ailleurs une liste de mots fonctionnels du français publiée par [24].

3.4 La représentation vectorielle du texte.

Le texte original peut être vu comme une séquence de mots. Ce type de représentation est actuellement incompréhensible pour les algorithmes d'apprentissage automatique qui ont besoin de recevoir des représentations vectorielles numériques des entités à classer. La représentation vectorielle consiste à transformer chaque document en une séquence de nombres, dans laquelle chaque nombre correspond à un mot du vocabulaire de l'ensemble des documents ou corpus. Pour transformer les documents de texte en vecteurs, on produit d'abord un vocabulaire avec tous les mots contenus dans les textes de l'ensemble d'entraînement. On produit ensuite une

matrice numérique dans laquelle chaque ligne correspond à un des documents de texte et chaque colonne correspond à un mot du vocabulaire du corpus. Si le mot n'apparaît pas dans le document, on lui assigne le nombre 0. Par contre, s'il apparaît, on peut lui assigner le nombre 1, ou celui correspondant au total de fois que le mot apparaît dans le document. Cette dernière matrice s'appelle la matrice de fréquences. La matrice numérique résultante peut être passée alors à l'algorithme de classification qui sera capable de l'interpréter et de travailler avec elle. Cette sorte de représentation est aussi appelée le sac de mots (bag of words). La figure 3.2 illustre ce processus.



$$\begin{pmatrix} & \downarrow & & & & & & \\ & t_1 & t_2 & t_3 & t_4 & t_5 & \cdots & t_d \\ \mathbf{x}_1 & 1 & 2 & 2 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & & & & & & \\ \mathbf{x}_n & 0 & 1 & 1 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

FIGURE 3.2 – Représentation vectorielle du texte.

3.5 La transformation des caractéristiques.

D'une part, on peut penser qu'il pourrait être judicieux de faire une représentation numérique qui accorde plus d'importance aux mots dont la fréquence est haute dans la catégorie à laquelle ils appartiennent et basse dans les autres catégories, en vu de pondérer la valeur numérique de chaque mot selon l'information qu'elle apporte pour la classification. Aussi, c'est l'effet produit par la pondération tf-idf, introduite par [72], qu'on utilisera dans la partie pratique de notre recherche. Il y a par ailleurs d'autres pondérations possibles comme le χ^2, le χ^2_P, le gini index, et le gain d'information, expliquées par [62], qui permettent aussi de capturer cette sorte de relations entre mots et documents.

D'autre part, étant donné que certains des documents peuvent être beaucoup plus longs que d'autres, faisant en sorte que ces vecteurs présentent des fréquences de mots plus grandes, il peut être nécessaire de normaliser les vecteurs pour éliminer l'influence de la taille des documents. Normaliser un vecteur signifie le changer d'échelle, de telle sorte que sa norme ou longueur soit égale à 1. Pour normaliser n'importe quel vecteur $\mathbf{v}$, il faut tout simplement le diviser par sa norme, c'est-à-dire :

$$\mathbf{v} = \frac{\mathbf{v}}{\| \mathbf{v}_p \|}$$

où, pour un vecteur $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ la norme est, :

$$\| \mathbf{v} \| = \left(\sqrt{| v_1 |^p + | v_2 |^p + \cdots | v_n |^p} \right)^{\frac{1}{p}} = \left(\sum_{i=1}^{n} | v_1 |^p \right)^{\frac{1}{p}} \tag{3.1}$$

appelée la distance de Minkowski ou norme p. La norme constitue une mesure de la longueur et il est possible d'utiliser différents types de distance pour la calculer. L' une des distances les plus utilisées est la distance euclidienne, appelée aussi distance

$L2$, qui est obtenue avec la valeur de $p = 2$, dans l'équation 3.1, c'est-à-dire,

$$\| \mathbf{v} \| = \left(\sqrt{| v_1 |^2 + | v_2 |^2 + \cdots | v_n |^2} \right)^{\frac{1}{2}} = \left(\sum_{i=1}^{n} | v_1 |^2 \right)^{\frac{1}{2}} \tag{3.2}$$

D'autres distances, comme la distance $L1$, aussi appelée cityblock, taxicab, ou Manhattan, illustrée dans la figure 3.3, obtenue en faisant $p = 1$ dans l'équation 3.1, peuvent aussi être utilisées. Il existe encore autres types de normes, qu'on ne mentionnera pas dû au fait qu'elles ne sont pas très utilisées dans le type d'applications réalisées dans notre recherche.

La distance euclidienne et de taxi (taxicab). Dans la géométrie taxicab chacune des trois lignes décrites (rouge, pourpre et bleu) ont la même longueur $a + b$ tandis que la ligne verte, correspondant à la distance euclidienne, qui est de longueur $\sqrt{a^2 + b^2}$ est le plus court chemin unique.

FIGURE 3.3 – Distance euclidienne et Manhattan.

Une fois terminé le processus de vectorisation du texte, on peut finalement passer à l'étape suivante, aller à l'algorithme d'apprentissage.

3.6 Résumé.

Dans ce chapitre, on a vu les différentes étapes pour faire la représentation vectorielle des documents de texte (extraction de termes, lemmatisation, suppression de mots fonctionnels, vectorisation et transformation). Cette représentation sera utilisée lors du processus d'optimisation de la fonction objectif du modèle choisi. En conséquence, dans les deux prochains chapitres, on va détailler les caractéristiques des machines à support de vecteurs et de la forêt de décision.

Chapitre 4

Les machines à vecteurs de support (SVM).

4.1 Le cas quand les données sont linéairement séparables.

On considère un ensemble d'observations d'entraînement $\mathbf{x}_{1:n}$ qui peut être considéré comme un ensemble de points dans un espace vectoriel de dimension d dans lequel chaque observation $\mathbf{x}_i$ est un vecteur dans l'espace $\mathbb{R}^d$ avec son étiquette associé de classe, y_i. Chaque y_i peut prendre une des deux valeurs, $+1$ ou -1, si le point observé appartient à une catégorie ou à l'autre, c'est-à-dire, $y = \{-1, 1\}$. Si les points sont linéairement séparables dans l'espace de caractéristiques, il est toujours possible de construire un hyperplan H qui sépare les exemples des deux catégories en permettant d'assigner l'étiquette 1 ou -1 selon que le point observé se trouve d'un côté ou l'autre de l'hyperplan. La SVM pourra trouver l'**Hyperplan de marge maximale** (Maximum Marginal Hyperplane MMH), c'est-à-dire, l'hyperplan donnant une séparation maximale des points appartenant aux différentes catégories. Cet hyperplan sera construit par les vecteurs d'entraînement qui sont les plus difficiles

à classifier, et qui seront à la limite de la marge de séparation en étant une espèce de support de la marge construite. Ils sont pour cette raison appelés les vecteurs de support.

Un exemple en deux dimensions est montré dans la figure.4.1 (a), où l'hyperplan est, dans ce cas-ci, une ligne qui divise un ensemble linéairement séparable des données conformées par des données (vecteurs) de dimension 2, ce qui veut dire que chaque instance de l'ensemble d'entraînement possède deux attributs ou caractéristiques. Les vecteurs de support sont tracés avec une ligne plus épaisse.

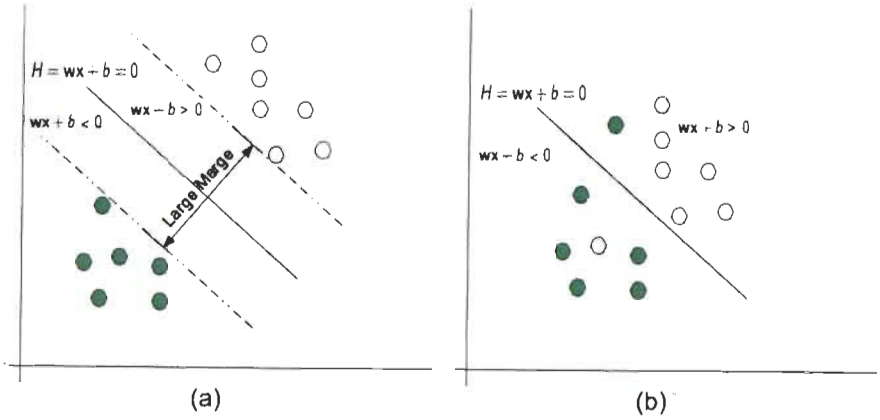


FIGURE 4.1 – SVM exemples séparables et non séparables.
(a) HMM avec des données séparables. (b) Données non séparables dans l'espace original.

Par définition, un hyperplan de séparation peut être écrit comme suit :

$$\mathbf{wx} + b = 0 \tag{4.1}$$

où $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ est un exemple dans l'espace de caractéristiques de dimension d, $\mathbf{w}$ est un vecteur de poids, à savoir, $\mathbf{w} = (w_1, w_2, ..., w_d)$ et b est une scalaire, aussi appelée le biais ou seuil. Il est p1.2ossible de classer les exemples par l'expression 4.1 en leur assignant la valeur 1 ou -1 si leur distance à l'hyperplan est non nulle, comme il est montré dans la figure 4.1.

On admet que ρ est la longueur de la marge de séparation générée par un hyperplan placé juste à la moitié de la distance entre les deux exemples les plus proches de l'une et de l'autre catégorie. Alors, pour chaque exemple $\mathbf{x}_i$, nous avons

$$\begin{array}{l} \mathbf{w}^T\mathbf{x}_i + b \leq \rho/2 \text{ si } y_i = -1 \\ \mathbf{w}^T\mathbf{x}_i + b \geq \rho/2 \text{ si } y_i = 1 \end{array} \Leftrightarrow y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq \rho/2 \tag{4.2}$$

Étant donné que $\mathbf{w}$ est perpendiculaire à l'hyperplan, la distance de l'hyperplan à n'importe quel point $\mathbf{x}_i$, en termes de la taille de $\mathbf{w}$ est

$$\frac{y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)}{\| \mathbf{w} \|}$$

Si on note $\mathbf{x}_s$, aux points les plus proches à l'hyperplan, c'est-à-dire, ceux qui se trouvent dans la limite de la marge, puis en redimensionnant $\mathbf{w}$ et b par $2/\rho$, dans l'expression 4.2 l'on obtient :

$$\rho = \frac{y_s\left(\mathbf{w}^T\mathbf{x}_s + b\right)}{\| \mathbf{w} \|} = \frac{1}{\| \mathbf{w} \|}$$

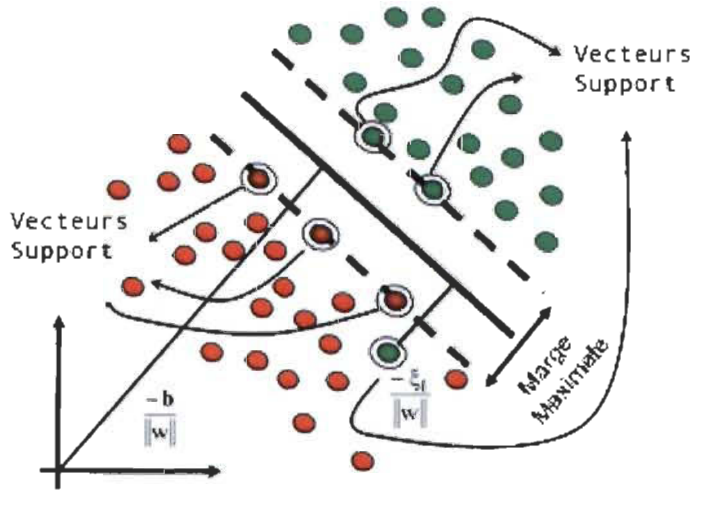


FIGURE 4.2 – Les points d'une catégorie sont représentés en rouge, ceux de l'autre sont représentés en vert. [81]

On trouve donc que la distance de l'hyperplan de séparation aux points les plus proches d'un côté de la marge est $\frac{1}{\| \mathbf{w} \|}$, où $\| \mathbf{w} \|$ est la norme euclidienne de $\mathbf{w}$, qui est $\sqrt{\mathbf{ww}}$.[1] Par conséquent, la longueur de la marge est $\frac{2}{\|w\|}$. On observe que, pour maximiser cette expression, il faut minimiser $\| \mathbf{w} \|$.

Minimiser $\| \mathbf{w} \|$ est l'équivalent de minimiser $\| \mathbf{w} \|^2$ mais, cette dernière a l'avantage d'être dérivable partout tandis que $\| \mathbf{w} \|$ n'est pas dérivable en $\| 0 \|$.

De plus, pour s'assurer que les exemples soient bien classés, on ajoute les contraintes :

$$\gamma_i = y_i \left(\mathbf{wx} + b \right) - 1 \geq 0, \forall i \in \{1, \ldots, n\}$$

On fait face au problème de minimiser $\| \mathbf{w} \|^2$ sujet aux γ_i dont la solution peut être trouvée, par la méthode des multiplicateurs de Lagrange qui sert à transformer un problème d'optimisation de fonction avec des contraintes en un problème d'optimisation de fonction sans contraintes.

Pour cela, on exprime le lagrangien *Lp* comme somme de la fonction à minimiser (la fonction objectif dans ce cas-ci) et de l'opposé de chaque contrainte γ_i multiplié

1. $\sqrt{\mathbf{ww}} = \sqrt{\mathbf{w}_1^2 + \mathbf{w}_2^2 + \cdots + \mathbf{w}_n^2}$

par une constante $\alpha_i \in \mathbb{R}^+$. Les α_i constituent les « multiplicateurs de Lagrange » .

$$\begin{aligned} L_P = L_P(\mathbf{w}, b, \alpha) &= \frac{1}{2} \parallel \mathbf{w} \parallel^2 - \sum_{i=1}^{n} \alpha_i y_i (\mathbf{w}\mathbf{x}_i + b) + \sum_{i=1}^{n} \alpha_i \\ &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i (\mathbf{w}\mathbf{x}_i + b) + \sum_{i=1}^{n} \alpha_i \end{aligned} \tag{4.3}$$

Lp doit être minimisé par rapport à $\mathbf{w}$.

Le gradient de LP devant être nul par rapport à $\mathbf{w}$ et b, ainsi qu'en dérivant et égalant à zéro, cela devient :

$$\begin{cases} \dfrac{\partial Lp}{\partial \mathbf{w}} & = 0 \\ \dfrac{\partial Lp}{\partial b} & = 0 \end{cases} \Rightarrow \begin{cases} \mathbf{w} & = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \\ \sum_{i=1}^{n} \alpha_i y_i & = 0 \end{cases}$$

De la formulation de L_P et de ces deux équations, on arrive à la formulation duale du lagrangien :

$$L_D = L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \tag{4.4}$$

Cette fonction n'est pas non plus fonction de $\mathbf{w}$ et doit être maximisée. Le maximum de L_D et le minimum de L_P correspondent aux mêmes valeurs de $\mathbf{w}$, b, et α_i.

Pour que $\mathbf{w}, b$ et les α_i existent, L_D doit vérifier les conditions de Karush-Kuhn-Tucker (KKT), référées par [68] :

i. $\dfrac{\partial L_P}{\partial w_\nu} = w_\nu - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_{i,\nu} = 0 \ \forall \nu = 1, \ldots, d$

ii. $\dfrac{\partial L_P}{\partial_b} = -\sum_{i=1}^{n} \alpha_i y_i = 0$

iii. $y_i (\mathbf{w}\mathbf{x}_i + b) - 1 \geq 0 \ \forall i = 1, \ldots, n$

iv. $\alpha_i \geq 0 \ \forall i = 1, \ldots, n$

v. $\alpha_i (y_i (\mathbf{w}\mathbf{x}_i + b) - 1) = 0 \ \forall i = 1, \ldots, n$

Ces conditions sont satisfaites, donc le problème a une solution.

Les α sont des multiplicateurs de Lagrange. La solution de l'équation (4.4) peut être trouvée par un algorithme de programmation quadratique (Quadratic Programming Package [1]) dont le résultat sera les valeurs α des multiplicateurs de Lagrange qui optimisent la fonction 4.4. La majorité d'elles auront une valeur de zéro, tandis que quelques unes auront des valeurs positives (conformément aux conditions 4. et 5. de KKT). On observe que la condition KKT 5 est vraie pour les valeurs α positives, si et seulement si,

$$(y_i (\mathbf{w}\mathbf{x} + b) - 1) = 0$$

Ceci signifie qu'ils sont les seuls vecteurs qui participent dans la construction de l'hyperplan en étant eux-mêmes dans la frontière de la marge maximale et en y constituant un sorte de «support». C'est la raison pour laquelle ils sont appelés des **vecteurs de support**.

4.2 La classification d'une nouvelle donnée.

Une fois construite la frontière de marge maximale, à l'aide de l'ensemble de données d'entraînement, on a une SVM entraînée et prête à classer des nouvelles données encore non observées.

Basée sur la formulation lagrangienne mentionnée ci-dessus, la frontière de décision peut être réécrite ainsi :

$$d(\mathbf{x}) = sgn\left(\sum_{i=1}^{N_s} y_i \alpha_i (\mathbf{s}_i \mathbf{x} + b)\right) \tag{4.5}$$

où y_i est l'étiquette de classe (1 où -1) du vecteur support $\mathbf{s}_i$; $\mathbf{x}$ est une donnée d'entraînement, α_i et b sont les paramètres numériques qui ont été déterminés automatiquement par l'optimisation de la fonction 4.4 et N_s est le nombre de vecteurs

de support.

Soit une donnée de test $\mathbf{x}$, mise dans l'équation (4.5), on vérifie alors le signe résultant, positif ou négatif, afin de savoir de quel côté de la frontière la donnée se trouve et, par conséquent, sa catégorie d'appartenance. Si le signe est positif, la SVM prévoit que $\mathbf{x}$ appartient à l'une des classes. Si elle est négative, elle appartient à l'autre.

4.3 Le cas quand les données ne sont pas linéairement séparables.

Si les données ne sont pas séparables dans l'espace de caractéristiques $X = \mathbb{R}^d$, il est toujours possible de faire une transformation des données vers un autre espace, généralement à plus haute dimension, dans lequel les données sont seront linéairement séparables.

Soit la transformation :

$$\Phi : \mathbf{X} \rightarrow \mathcal{Z} \tag{4.6}$$

C'est-à-dire, une transformation de l'espace des données en un espace des caractéristiques $\mathcal{Z}$, avec $\mathbf{X} < \mathcal{Z}$. où chacune des caractéristiques ajoutées de $\mathcal{Z}$ est une combinaison non linéaire des dimensions de $\mathbf{X}$.

Pour trouver la solution dans le nouvel espace, il est tout simplement nécessaire de remplacer les vecteurs originaux $\mathbf{x}_i$ par les nouveaux vecteurs $\mathbf{z}_i$ du nouvel espace $\mathcal{Z}$, autant dans l'équation 4.4 que dans les restrictions correspondantes.

Une fois la solution trouvée, il est possible d'identifier les vecteurs de support dans l'espace de caractéristiques original correspondant aux vecteurs $\mathbf{x}_i$ dont les valeurs α sont non nulles en permettant de classer des nouveaux exemples par l'équation 4.5.

$$(x_1, x_2) \overset{\Phi}{\mapsto} (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

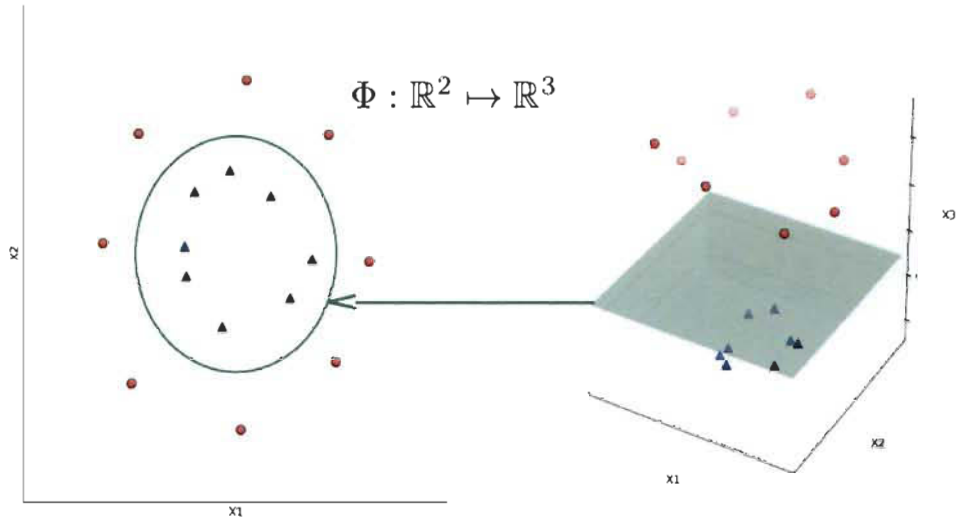


FIGURE 4.3 – Projection des points vers $\mathbb{R}^3$ pour trouver la frontière de séparation.

4.4 La marge souple.

La SVM peut toujours trouver une frontière qui sépare correctement toutes les données d'entraînement. Toutefois, il est possible que ce type de frontière n'obtienne pas une bonne généralisation si les données ne sont pas linéairement séparables dans l'espace original. Il est donc nécessaire d'utiliser la SVM de marge souple qui, en ayant une surface moins sinueuse et aussi en permettant la mauvaise classification de certaines des données d'entraînement, améliorera la capacité de généralisation du classeur. La marge souple résout en même temps l'influence de données atypiques[33, 56], comme le montre la figure 4.4.

Pour procéder, un jeu de variables mesurant l'erreur est introduit. Ces nouvelles variables sont notées ξ_i et appelées variables ressort (slack variables).

Si $\mathbf{x}_i$ dépasse la marge de séparation, alors $\xi_i \geq 0$. Ainsi, ξ_i indique à quel point l'exemple $\mathbf{x}_i$ est loin de la marge : plus $\mathbf{x}_i$ sera loin de la séparatrice, plus ξ_i sera grand, comme illustré à la figure 4.4.

Le problème devient la recherche de l'hyperplan de séparation maximale impliquant une pénalisation pour les exemples qui violent la marge. La fonction objectif à minimiser devient :

$$\frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_{i=1}^{N} \xi_i \tag{4.7}$$

où C est une constante, appelée le terme de régularisation, qui permet de donner plus ou moins d'importance aux points qui violent la marge de séparation en permettant de faire cette marge plus ou moins souple au besoin. En résumé, on résout maintenant le problème :

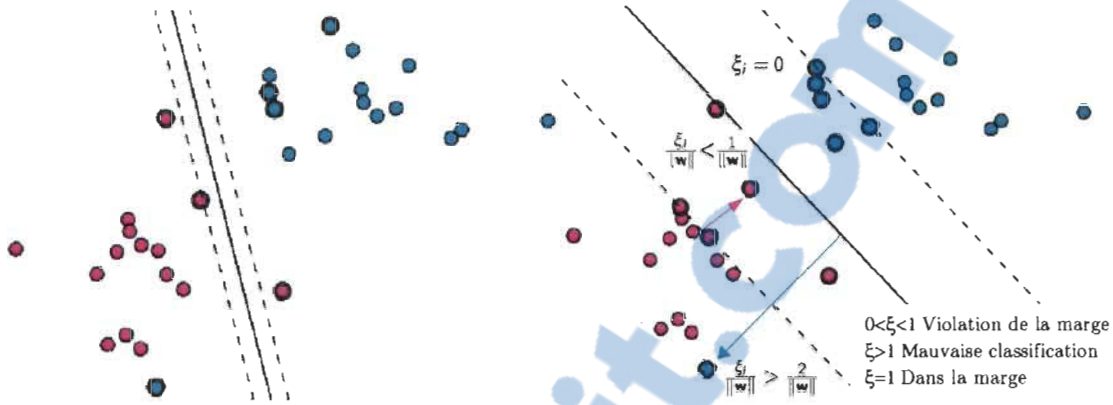
$$\begin{cases} \text{minimiser } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i \\ \text{sujet à } \gamma_i \ \xi_i \geq 0, \ \forall i \end{cases} \tag{4.8}$$

En appliquant la méthode de Lagrange comme précédemment on obtient les multiplicateurs α_i.

La valeur optimale de C est trouvée en essayant différents modèles candidats, ayant chacun une différente valeur de C, et en mesurant sa précision par la procédure de validation croisée, pour choisir, finalement, le modèle ayant la meilleure performance.

On peut considérer que le paramètre C a la même fonction que le paramètre de régularisation λ mentionné dans la section 2.4.6 ; toutefois, dans le cas des SVM, la complexité du classeur diminuera en réduisant la valeur de C, c'est-à-dire que, comme le mentionne [57], on peut considérer que le paramètre C joue un rôle semblable à $\frac{1}{\lambda}$.

Une fois entraînée la SVM avec marge souple, la classification de nouvelles données est faite précisément de la même manière que celle qui a été exposée précédemment dans la section 4.2.



À gauche, sensibilité de la SVM aux points atypiques. À droite, la marge souple permet d'éviter l'influence d'une donnée atypique.

FIGURE 4.4 – SVM Marge souple.

4.5 L'astuce de la fonction noyau.

Le calcul du produit point des vecteurs dans l'espace d'ordre supérieur, qu'il faut calculer dans l'équation 4.4 peut être énormément demandeur de ressources computationnelles, rendant ce calcul très difficile, voire impossible. Pour résoudre ce problème, une astuce, connue sous le nom de l'astuce de la fonction noyau (the kernel trick), est utilisée.

Si on a une fonction K, appelée de fonction noyau ou kernel :

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = z_i \cdot z_j$$

Il est possible alors de faire tous les calculs concernant la SVM en utilisant K au lieu de transformer les données par la fonction ϕ, même sans nécessairement connaître cette fonction ϕ . Comme résultat, le calcul se fait dans l'espace d'origine et ceci est beaucoup moins coûteux qu'un produit scalaire en grande dimension.

Lors de l'optimisation du problème quadratique, on remplace les produits scalaires $\mathbf{z}_i \cdot \mathbf{z}_j$ (qui remplaceraient $\mathbf{x}_i \cdot \mathbf{x}_j$ dans l'équation 4.4) par $K(\mathbf{x}_i, \mathbf{x}_j)$:

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{4.9}$$

Pour classer une nouvelle donnée $\mathbf{x}$, on doit calculer :

$$sgn\left(\sum_{i=1}^{N_s} y_i \alpha_i (\mathbf{s}_i \mathbf{x} + b)\right) = \sum_{i=1}^{N} \alpha_i y(\mathbf{s}_i) K(\mathbf{s}_i, \mathbf{x}) \tag{4.10}$$

La solution est encore la même que celle employée dans le cas des données linéairement séparables (eq 4.5), en remplaçant le produit de vecteurs par la fonction noyau.

4.5.1 Les fonctions noyau.

Pour qu'une fonction puisse servir de noyau, il faut qu'elle satisfasse la condition de Mercer (pour s'assurer que le problème quadratique possède une solution[68, 58]) :

> « Une fonction $K(x, y)$ respecte cette condition si pour toute fonction $g(x)$ telle que $\int g(x)^2 dx$ est finie, on a
>
> $$\int \int K(x, y) g(x) g(y) dx dy \geq 0 \text{ . »[68]}$$

Si on note le vecteur d'entraînement comme $\mathbf{x}$, certaines des fonctions noyau connues sont [59, 35] :

- Linéaire : $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x} \cdot \mathbf{x}' \rangle$
- Fonction de base radiale gaussienne(RBF) :
  - $K(\mathbf{x}, \mathbf{x}') = exp(-\gamma \parallel \mathbf{x} - \mathbf{x}' \parallel^2)$;[2]avec $\gamma > 0$.
- Polynômiale : $K(\mathbf{x}, \mathbf{x}') = (\gamma \langle \mathbf{x} \cdot \mathbf{x}' \rangle + r)^d > 0$
- Sigmoide : $K(\mathbf{x}, \mathbf{x}') = tanh(\gamma \langle \mathbf{x} \cdot \mathbf{x}' \rangle + r)$; $\gamma > 0$

2. γ peut aussi être aussi paramétrée comme $\gamma = \frac{1}{2\sigma^2}$, où σ^2 correspond à la variance de la fonction gaussienne [59]

4.6 La prévision de la capacité de généralisation.

Une des principales caractéristiques des SVM, mentionnée par [1], est que la proportion des vecteurs de support relative à l'ensemble d'entraînement peut servir à calculer, la capacité de généralisation des SVM comme suit :

Si on appelle l'erreur de généralisation E_{out} celle attribuée à la proportion de données de test classées de façon erronée par une SVM entraînée, alors cette erreur a la cote supérieure suivante :

$$E_{out} \leq \frac{\#\text{Vecteurs de support}}{N-1}$$

où N-1 représente le total des données d'entraînement. Ainsi si, par exemple, on a un ensemble d'entraînement de 5000 vecteurs (données) et que par l'optimisation du SVM on a 250 vecteurs de support, alors, la capacité de généralisation de la SVM sera d'au plus près $100\% \left[1 - (\frac{250}{5000})\right] = 95\%$.

4.7 La SVM pour plus de deux catégories.

Tel qu'écrit tout au long de ce chapitre, la conception des machines à support de vecteurs est faite pour classer seulement dans deux catégories. Toutefois, il est possible de classer des données dans de multiples catégories en utilisant l'une des méthodes suivantes.

4.7.1 Une contre une (One versus one).

Cette méthode pour classer en k catégories, consiste à entraîner $\binom{k}{2}$ SVM chacune capable de classer entre deux catégories. Pour classer une donnée de test, celle-ci est classée par tous les $\binom{k}{2}$ classeurs construits et la catégorie assignée sera celle dans laquelle elle a été assignée davantage de fois.

4.7.2 Une contre tous (One versus all).

Pour classer en k catégories, cette méthode définit une des catégories comme étant la catégorie intérêt pour identifier si les données appartiennent, ou non, à celle-ci. On entraîne ensuite un classeur SVM pour classer les données d'entraînement dans la catégorie d'intérêt ou dans l'autre. On répète ce processus k fois avec chacune des catégories, c'est-à-dire qu'on entraîne k classeurs chacun capable d'identifier l'appartenance à une des catégories. Finalement, pour classer un exemple de test, on utilise la formule 4.5 k fois, une avec chaque classeur. La catégorie assignée sera celle dont la valeur résultant dans l'équation est la plus grande (c'est-à-dire toute la valeur, non seulement le signe).

4.8 Contrôler la complexité du classeur.

D'après le guide présenté par [58] [3] :

Pour varier la complexité du classeur selon l'abondance et la complexité des données (voir section 2.4) il est possible d'utiliser le paramètre $C \approx \frac{1}{\lambda}$, de la fonction objectif, ou le γ de la fonction noyau comme suit :

Long C : Haute variance.

Petit C : Bas variance.

Long γ : Haute variance.

Petite γ : Petite variance.

3. Étant donné que l'auteur utilise le paramètre $\gamma = \frac{1}{2\sigma^2}$, où σ^2 correspond à la variance de la fonction gaussienne. on a écrit la variation avec les expressions des kernels de la section 4.5.

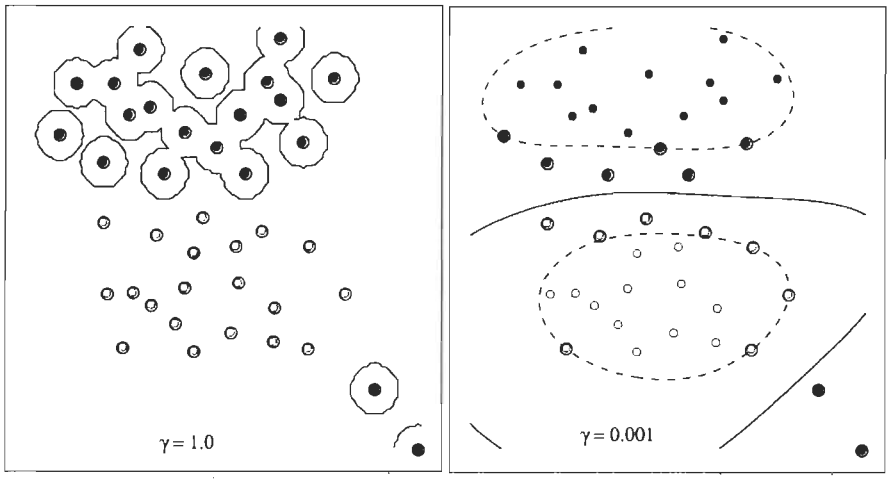


FIGURE 4.5 – Effet de la variation du paramètre γ, en laissant fixe C = 1.0. Dans le graphique à gauche, le classeur sur-adapte les données tandis que, dans celui à droite, on obtient un meilleur ajustement en utilisant une valeur plus petite de γ, en diminuant la variance du classeur.

4.9 Les avantages et les inconvénients des SVM.

4.9.1 Les inconvénients.

— Les SVM peuvent avoir des problèmes dans le cas où beaucoup des termes sont non significatifs pour la discrimination de classe [54].

— Elles ne servent pas à calculer des probabilités d'appartenance à la classe [33].

— L'utilisation des fonctions noyaux ne permet pas de sélectionner des caractéristiques importantes pour la classification [33].

4.9.2 Les avantages.

— Étant donné que la complexité du modèle ne dépend pas de la dimension de l'espace de caractéristiques, les SVM sont particulièrement bonnes à traiter avec des données à dimensions élevées (telles que les représentations vectorielles de texte).

— En utilisant seulement les vecteurs de support dans la frontière de décision, il est possible d'obtenir des résultats de faible densité en traitant avec des grands ensembles de données.

— La complexité du classeur peut être contrôlée par le paramètre C en rendant la marge plus lisse ou plus sinueuse au besoin.

— Il est possible d'avoir une idée de la capacité de généralisation du classeur même sans avoir vu qu'une seule donnée de test.

— Avec la SVM linéaire, c'est-à-dire, sans noyau, la frontière de décision construite correspond à un hyperplan dont les coefficients peuvent être interprétés comme l'importance que le classeur donne aux caractéristiques d'après leur importance pour la classification [8].

En plus du premier et du quatrième avantage mentionnés en haut, l'utilisation de la SVM linéaire est particulièrement bonne pour la classification des documents de texte pour les raisons suivantes, mentionnées par [43] :

— La plupart des problèmes de catégorisation des textes sont linéairement séparables.

— Il y a peu de caractéristiques insignifiantes.

Pour ces raisons, et du fait les objectifs particuliers de notre recherche, on utilisera la SVM linéaire afin d'identifier les vecteurs (mots) les plus significatifs pour la classification dans chaque catégorie, permettant ainsi l'analyse automatique du discours des documents qu'ils les composent.

4.10 Résumé.

Dans ce chapitre, nous avons présenté le développement de la fonction objectif à optimiser dans le modèle de machines à support de vecteurs tant pour le cas d'exemples séparables, que non séparables sur l'espace de caractéristiques. Nous

avons présenté, dans ce dernier cas, le paramètre de régularisation C qui permettra de modifier la complexité du modèle. Nous avons décrit aussi comment, par l'astuce de la fonction noyau (kernel), il est possible de construire des frontières de décision plus complexes qui permettront de séparer les points, même si ceux-ci sont très mélangés dans l'espace vectoriel. On a, finalement, insisté sur les avantages et les inconvénients de ce modèle ainsi que sur les raisons justifiant le choix de la SVM linéaire dans le cadre de notre recherche.

Maintenant, nous verrons en détail le deuxième modèle qu'on appliquera : le modèle de forêts de décision et on regardera comment la nature de ses caractéristiques servent nos objectifs de recherche.

Chapitre 5

Les arbres et forêts de décision

5.1 Introduction.

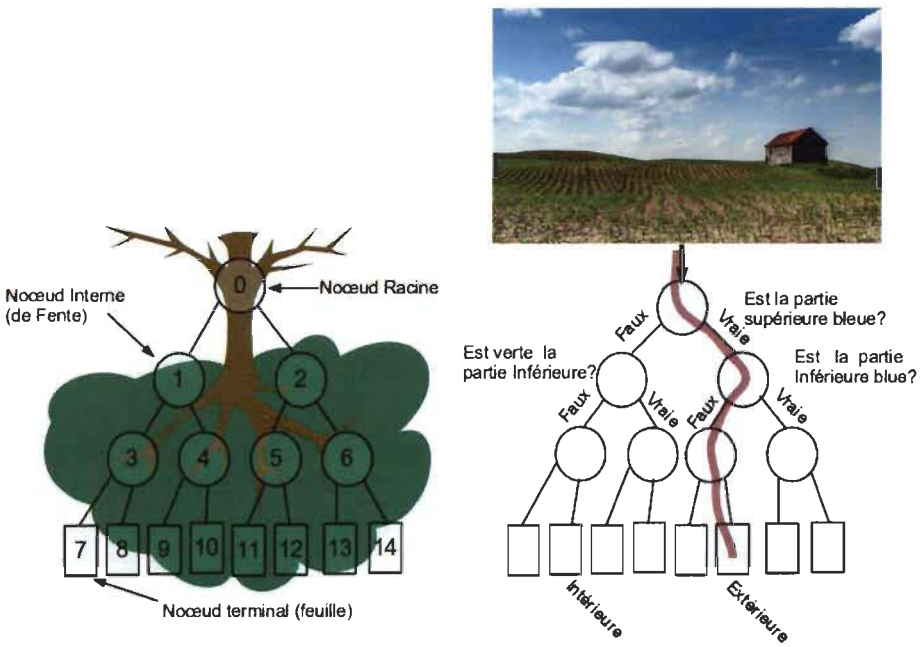
D'après [42], à la fin des années 70s et début des 80s deux groupes de chercheurs ont développé, de manière indépendante l'un de l'autre, la technique de classification d'arbres de décision. J. Ross Qinlan a développé un algorithme d'arbres de décision appelé ID3 (Iterative Dichotomiser). Plus tard, en améliorant plusieurs caractéristiques de l'ID3, il a développé et présenté un autre algorithme appelé C4.5. De leur côté, en 1984, L. Breiman, J. Friedman, R. Olshen, et C. Stone, un groupe de statisticiens, ont développé un algorithme pour produire des arbres de décision binaires appelé CART (Classification and Regression Trees). Ces algorithmes ont été le réel début de la recherche et ils ont produit le développement de nouveaux et très intéressants algorithmes de classification basés sur des arbres de décision.

Dans ce chapitre, nous expliquerons les algorithmes arbres et forêts de décision.

5.2 Les arbres de décision.

Comme il a été mentionné dans le chapitre 2, le modèle d'arbres de décision fait partie des modèles d'apprentissage statistique automatique basés sur une approche

géométrique pour segmenter l'espace de caractéristiques. L'algorithme consiste à construire un ensemble de classeurs, appelés classeurs faibles, qui sont séquentiellement introduits avec l'intention de faire une segmentation de l'espace de caractéristiques permettant de classer les données. Cette séquence de classeurs est ensuite représentée comme une série de nœuds connectés et placés hiérarchiquement. Le résultat final est un arbre qui représente la séquence de classeurs en plaçant dans la partie supérieure le nœud correspondant au classeur dont la capacité de classification est la plus grande entre tous les classeurs et, en allant vers le bas des autres nœuds, en représentant d'autres classeurs faibles, en ordre d'importance décroisssante. Le résultat final, est un arbre renversé comme celui représenté dans la figure 5.1 .



(a) (b)

Arbre de décision. (a) Structure générale d'un arbre de décision. (b) Exemple en montrant un arbre de décision illustratif utilisé pour décider si une photo représente une scène d'intérieur ou extérieure.

FIGURE 5.1 – Arbre de décision. Inspiré de [11]

5.2.1 L'entraînement des arbres.

On est face à une technique supervisée d'apprentissage automatique. Il est donc nécessaire, pour entraîner le classeur, d'utiliser un ensemble de données d'entraînement dont les étiquettes soient connues.

On présente, comme on l'a mentionné dans la section 2, un exemple d'entraînement comme le vecteur

$$\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$$

et on cherche à connaître le risque ou la perte conditionnelle :

$$P(\omega_i \mid \mathbf{x})$$

Le processus d'entraînement consiste à ajouter, de manière séquentielle, des classeurs faibles, ou frontières, qui segmenteront l'espace de caractéristiques. Ces classeurs faibles peuvent avoir une multitude de forme. Toutefois, leur élection affectera la complexité du classeur final construit et, partant, sa capacité de généralisation. Pour faciliter l'explication, on considérera que des droites perpendiculaires à l'une des dimensions ou caractéristiques, c'est-à-dire, à l'un des axes. On aura donc, seulement des régions rectangulaires.

Pour la classification [1], l'objectif est de trouver J régions rectangulaires, mutuellement exclusives, $R_1, \cdots, R_j$ qui minimisent le taux d'erreur de classification [31] :

$$\sum_{j=1}^{J} = 1 - \max_k (\hat{p}_{jk}) = \sum_{j=1}^{J} E_j \tag{5.1}$$

où, $\hat{p}_{jk}$ représente la proportion de données d'entraînement dans la j-ème région qui appartiennent à la classe k-ème. L'objectif est de produire une segmentation donnant le plus petit nombre d'erreurs de classification.

Toutefois, comme le mentionne [31], il est computationnellement impossible de

1. Le lecteur intéressé aux arbres pour la prédiction de valeurs continues, peut regarder [31]

calculer toutes les partitions possibles de l'espace de caractéristiques en J rectangles. Il est donc nécessaire d'utiliser un approche séquentielle connue telle la division récursive binaire, expliquée comme suit :

Chacun des classeurs faibles essayera de faire une classification complète, c'est-à-dire, correcte des données, et ce, en segmentant l'espace de caractéristiques. Chaque nœud y sera représenté par un nœud dans l'arbre. Si un classeur faible n'est pas capable de classer correctement toutes les données, il rendra alors une segmentation de l'espace de caractéristiques permettant la meilleure classification possible des données résultants du classeur faible précédent. Cette segmentation rendra des régions plus petites et nouvelles contenant des sous-groupes de données qui seront plus homogènes et, par conséquent, plus facilement classables par le classeur faible suivant.

Chaque classeur faible est introduit en regardant toutes les caractéristiques $\mathbf{x}_j$, et en ajoutant un point de coupe s qui produit des sous-régions

$$R_1(j,s) = \{\mathbf{x} \mid \mathbf{x}_j < \mathrm{s}\} \text{ et } R_2(j,s)\{\mathbf{x} \mid \mathbf{x}_j \geq \mathrm{s}\}$$

et en choisissant j et s tels qu'ils minimisent la proportion d'exemples mal classés dans les sus-régions générées.

Chaque fois qu'on ajoute un nouveau classeur faible, on ajoutera un nouveau nœud dans l'arbre. Le premier nœud, qui correspond au classeur faible qui fait la meilleure classification possible sur l'ensemble de données d'entraînement au complet, est placé dans la partie supérieure de l'arbre et s'appelle le nœud racine. Les nœuds intermédiaires sont appelés nœuds internes de fente et les nœuds finaux s'appellent les nœuds feuille. Ces derniers sont les derniers classeurs faibles ajustés et ils sont chargés d'assigner la classe à des exemples reçus en générant des régions finales de l'espace de caractéristiques produits par l'algorithme. La figure 5.1 (a) montre les types de nœuds.

Le processus d'ajustement de classeurs faibles, et de ses nœuds correspondants dans l'arbre, est répété jusqu'à arriver à générer des régions disjointes

$$R_1, R_2, \cdots, R_j$$

telles que :

i. Toutes les données contenues dans la région R_i sont de la même classe. Dans ce cas, le dernier classeur ajouté, générant la région R_i, sera un nœud feuille qui assignera la classe des données contenues dans cette région.

ii. Il y a un nombre minimal de données contenues, par exemple 5 dans la région R_i. En ce cas, le dernier classeur ajouté, générant la région R_i, sera un nœud feuille qui assignera la classe de la majorité des données contenues dans cette région.

La figure 5.2 montre ce processus.

Toutefois, la procédure qu'on vient de décrire peut produire des arbres qui suradaptent énormément les données d'entraînement. Ceci est dû au fait que pour diminuer le taux d'erreur dans l'équation (5.1), il est possible, avec les données d'entraînement, que soitent générées trop de régions, en produisant autant de régions que de données. L'arbre résultant s'adapterait parfaitement aux données d'entraînement mais sa capacité de généralisation serait très pauvre. Pour résoudre ce problème, dans la définition de la fonction objectif J à minimiser on ajoute un terme de pénalisation au nombre total de classeurs faibles (et nœuds), à l'équation (5.1). On définit donc la fonction objectif à optimiser comme suit :

$$J = \sum_{j=1}^{|T|} = 1 - \max_k (\hat{p}_{jk}) + \lambda \mid T \mid \tag{5.2}$$

où $\mid T \mid$ indique le nombre de nœuds feuille dans l'arbre. On observe que plus grand est le nombre de nœuds terminaux, plus grande sera la pénalisation. Le para-

mètre de pénalisation permet ainsi d'« élaguer » l'arbre pour varier sa complexité, en permettant d'obtenir une taille optimale qui rend la meilleure classification possible des nouveaux exemples. La valeur λ, permettant d'arriver à une taille optimale de l'arbre, peut être trouvée par validation croisée [32].

5.2.2 L'optimisation des nœuds.

Étant donné que le taux d'erreur de classification n'est pas une mesure suffisamment sensible pour bien faire croître les arbres, on utilise à sa place, à chaque étape du grandissement de l'arbre, l'une des deux mesures de la variance totale au long des k classes qui sont numériquement très semblables [32] : L'entropie et le Gini index .

5.2.2.1 Le Gini index.

Le Gini index est défini comme suit :

$$G = \sum_{k=1}^{K} \hat{p}_{jk} \left(1 - \hat{p}_{jk}\right) \tag{5.3}$$

5.2.2.2 L'entropie.

L'entropie est définie ainsi :

$$D = -\sum_{k=1}^{K} \hat{p}_{jk} log \left(\hat{p}_{jk}\right) \tag{5.4}$$

Ces mesures sont aussi appelées des mesures de « pureté », parce qu'une petite valeur indique qu'une région, ou son nœud associé, contient principalement des observations d'une même classe.

5.2.3 La phase de test.

Une fois qu'on a fait grandir l'arbre, la phase de test consiste simplement à faire passer chacune des données de test par l'arbre de décision. À partir du nœud racine, chaque nœud appliquera une fonction de triage à l'exemple $\mathbf{x}$ qui sera envoyé, selon le résultat de cette fonction, vers un des nœuds suivants dans la hiérarchie. Ce processus est répété jusqu'à arriver à un nœud feuille qui assignera sa classe d'appartenance.

La figure 5.2 montre l'arbre de classification produit avec le célèbre ensemble de données de fleurs iris d'Anderson [2] et Fisher [26] en 1936. On utilise seulement deux des caractéristiques : la longueur et la largeur des pétales pour pouvoir représenter cet arbre graphiquement dans deux dimensions. On entraîne l'algorithme pour classer des données de fleurs dans trois types possibles à partir de ces deux caractéristiques. Dans la partie à gauche on montre les données d'entraînement et les régions produites en introduisant séquentiellement les classeurs faibles θ_i et, dans la partie droite, leurs nœuds correspondant dans l'arbre sont visibles. Si une nouvelle donnée $\mathbf{x} = (x_1, x_2)$ est passée par l'arbre, et que celle-ci arrive finalement à un nœud feuille qui est associé avec la région 1 (R_1), alors la donnée sera classée comme de l'espèce Setosa. Si le nœud feuille final est associé avec la région 3 (R_3), la donnée sera classée comme de l'espèce Versicolor. Finalement si le nœud feuille final place la donnée dans l'une des régions 2 ou 4 (R_2 où R_4), alors elle sera classée comme de l'espèce Virgínica.

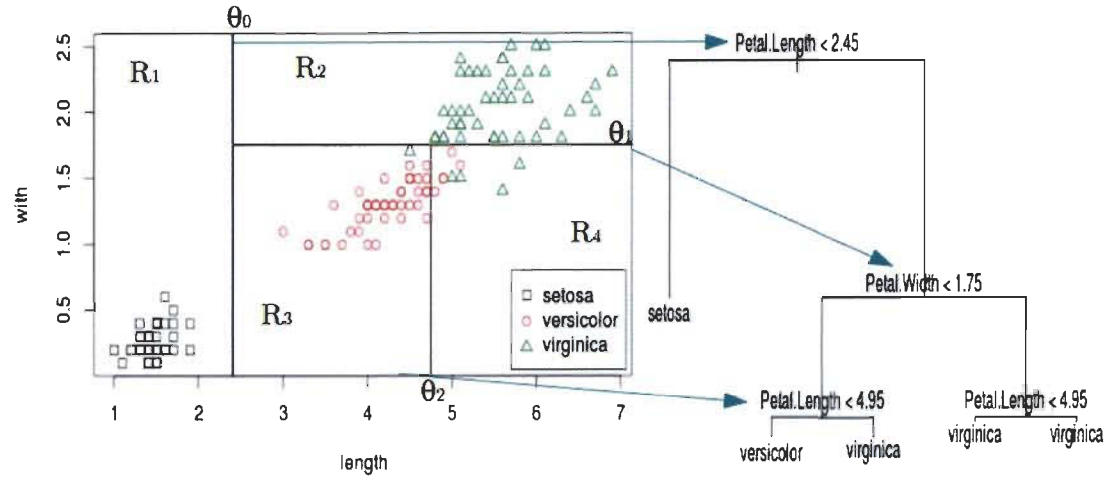


FIGURE 5.2 – Exemple d'arbre de classification en utilisant l'ensemble de données iris de Fisher.

5.3 Les avantages et les inconvénients des arbres de décision.

L'avantage principal des arbres de décision est qu'ils permettent de faire une représentation graphique du processus de décision facile à interpréter pour toute personne et que, comme le mentionne [32], beaucoup de personnes considèrent comme un « miroir » du processus humain de prise de décision. Toutefois, en accord avec [32], les arbres ont, en général, une performance plus mauvaise que d'autres techniques d'apprentissage statistique. C'est pour cette raison qu'ils ont été peu utilisés jusqu'à l'apparition d'une nouvelle technique, basée néanmoins sur eux, appelée les forêts de décision, qui forme ce jour l'une des techniques d'apprentissage automatique de pointe.

5.4 Le modèle de forêt de décision.

La principale cause de la plus ou moins bonne capacité de généralisation des arbres de décision est qu'ils produisent des classeurs avec un petit biais mais avec une grande variance. Ceci fait que les arbres tendent à sur-adapter les données d'entraînement. Pour résoudre ce problème, L. Breiman[6] a introduit une nouvelle technique : les forêts de décision. Ce modèle se base sur l'utilisation d'une des méthodes de comité ou ensemble, mentionnées dans la section 1.2.8. Le modèle utilise un ensemble d'arbres afin de calculer la valeur « moyenne » de prévision ainsi que le bagging pour réduire la variance, rendant ainsi un modèle qui aura une bien meilleure capacité de généralisation que les arbres individuels.

Si en construisant les nœuds, on utilise toutes les caractéristiques pour choisir l'une d'elles comme le classeur faible, le résultat sera que les arbres produits seront tous identiques, et la moyenne sera équivalente à chacun des arbres individuels. Ceci est dû au fait que les arbres seront complètement corrélés. La solution est la randomisation dans la construction des nœuds, c'est-à-dire, choisir au hasard un sous-ensemble $d_j < d$ dans la construction de chaque nœud. Ceci permettra de décorréler les arbres en réduisant la variance et en générant un modèle qui ne soit pas équivalent à l'un des arbres individuels.

Une autre manière de réduire la variance consiste à utiliser toutes les caractéristiques dans la construction des arbres et à introduire le caractère aléatoire par la stratégie d'échantillonnage bootsrap ou bagging que l'on va expliquer maintenant.

5.4.1 Le Bagging.

Le Boostrap aggregation ou Bagging est une procédure générale utilisée pour réduire la variance d'une méthode d'apprentissage statistique. Cette procédure se base sur l'idée que faire la moyenne d'un ensemble d'observations en réduit la variance.

Ainsi, s'il était possible de produire B différents ensembles de données d'entraînement et de produire un arbre avec chacun ainsi que calculer la moyenne des arbres produits. L'arbre « moyenne » résultant aurait une variance beaucoup plus petite que chacun des arbres individuels. Toutefois, étant donné qu'on dispose habituellement que d'un seul ensemble de données, le bagging nous permet de produire B_i nouveaux ensembles de données d'entraînement en prenant des échantillons, uniformément avec remise, de B. Étant donnée que ces sont des échantillons avec remise, quelques observations seront répétées dans chaque ensemble B_i. Il peut être démontré que chaque B_i aura approximativement $(1 - 1/e) \approx 0.63$, c'est-à-dire, deux tiers d'exemples uniques ou non répétés. Cette proportion de données est appelée l'échantillon dans la bourse « in bag sample » tandis que les exemples non inclus, dans un B_i particulier, approximativement un tiers de B, c'est-à-dire, $B/3$, forment ce qu'on appelle l'échantillon hors de la bourse «out of bag sample » (OOB).

En construisant ainsi les arbres pour faire la moyenne au moyen du Bagging, ceux-ci peuvent être construits de grande taille sans avoir à « les élaguer » en permettant le calcul de la moyenne d'un ensemble d'arbres dont la variance sera petite.

5.4.2 La randomisation de nœuds.

Soit p est l'ensemble de toutes les caractéristiques. Pour introduire le caractère aléatoire dans la construction des nœuds, il n'est pas nécessaire de disposer que d'un sous-ensemble $p_j \subset p$ de caractéristiques à chaque nœud. Pour la classification, si on utilise des lignes perpendiculaires à une des dimensions comme classeurs faibles, il sera nécessaire de tester exhaustivement toutes les caractéristiques en calculant le gini index ou l'information gain résultant pour choisir, finalement, celle dont la valeur soit la plus petite. On aura donc $p_j = d$, où d est le total de caractéristiques ou de dimensions de données. Toutefois, comme on vient de mentionner, il sera possible de disposer seulement $\sqrt{d}$ des caractéristiques en entraînant chaque nœud. Ainsi, par

exemple, si l'ensemble de données est de dimension 10000, il sera nécessaire, pour l'entraînement de chaque nœud, de sélectionner uniformément au hasard seulement $\sqrt{10000} = 100$ des caractéristiques des données en les testant exhaustivement comme nœuds candidats.

On obtient donc que la fonction objectif à optimiser à chaque nœud est :

$$\theta_j^* = \underset{\theta_j \in p_j}{argmax}\ I_j \tag{5.5}$$

ou I_j est l'une des mesures d'information, soit le gini index ou l'entropie, θ_j sont les possibles frontières de décision qui seront exhaustivement testées en choisissant la meilleure d'elles comme le classeur faible θ_j^* finalement ajouté,à l'étape j.

La randomisation des nœuds permet de contrôler le caractère aléatoire des arbres qui constituent la forêt [11, 32]. Une valeur plus petite de p_j, habituellement choisie de façon identique pour tous les arbres, implique moins de corrélation entre les arbres de la forêt. Si $p_j = p$, alors on aura une corrélation totale entre tous les arbres faisant en sorte que tous soient identiques, l'un par rapport à l'autre, et que la capacité de la forêt soit la même que celle d'un arbre individuel; mais si $p_j = 1$ on obtient le caractère aléatoire maximal et des arbres totalement non corrélés. Il faut, donc, trouver la bonne valeur p_j. D'après [32], une quantité habituellement utilisée et qui offre des bons résultats est $p_j = \sqrt{p}$; sinon, la valeur optimale peut également être trouvée par validation croisée.

5.4.3 L'entraînement de la forêt.

Pour entraîner la forêt, on produit d'abord un ensemble de T arbres $t \in \{1, \cdots, T\}$ de manière indépendante, et en faisant ainsi la randomisation des nœuds. Par la suite, on doit pousser par ces derniers les données d'entraînement, dès la racine, jusqu'à ce qu'elles atteignent une feuille, en combinant toutes les prévisions dans une seule,

par le calcul de la moyenne simple :

$$P(\omega \mid \mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} p_t(\omega \mid \mathbf{x}) \tag{5.6}$$

où $p_t(\omega \mid \mathbf{x})$ est la prévision faite par l'arbre t pour l'exemple $\mathbf{x}$.

En utilisant la procédure précédente pour avoir des prévisions ou des classifications de l'ensemble de données d'entraînement, il est aussi possible d'utiliser la validation croisée, expliquée dans la section 2.4.7, pour optimiser la performance du classeur de forêt de décision, en variant ces paramètres, afin d'obtenir la meilleure capacité de généralisation possible.

5.4.4 La prévision et la classification des données.

Pour prévoir ou classer une donnée $\mathbf{x}$, il faut simplement la passer par les arbres de la forêt et calculer la moyenne par l'équation 5.6.

La moyenne obtenue correspond à la valeur prévue par la forêt de décision pour des valeurs continues. Pour l'assignation des valeurs catégoriques, c'est-à-dire pour la classification, le résultat de l'équation 5.6, retourne une distribution de probabilité d'appartenance à chacune des classes.

Donc pour classer, il faut simplement choisir la classe dont la probabilité est la plus grande. Cette catégorie sera celle qui a été assignée le plus grand nombre de fois par les arbres de la forêt, c'est-à-dire, pour la plupart des arbres. Ainsi, par exemple, si on a généré une forêt de $T = 100$ arbres afin de faire la classification en trois classes $(\omega_1, \omega_2, \omega_3)$, chacune d'elles donnera une catégorie d'appartenance de $\mathbf{x}$; on admet que le premier et deuxième arbres classent la donnée dans la première et deuxième classe respectivement, et le dernier arbre classe dans la troisième classe. Ces classifications peuvent être représentées par des vecteurs en ayant 1 dans la

catégorie de classification et de 0 dans les autres, c'est-à-dire :

$$\begin{aligned} p_1(\omega \mid \mathbf{x}) &= (1,0,0) \\ p_2(\omega \mid \mathbf{x}) &= (0,1,0) \\ &\vdots \\ p_{100}(\omega \mid \mathbf{x}) &= (0,0,1) \end{aligned}$$

Si on suppose qu'une nouvelle donnée est classée par 10 arbres dans la classe 1, 70 dans la classe 2 et 20 dans la classe 3, alors le résultat de l'équation 5.6 est

$$\frac{1}{100}\sum_{t=1}^{100} p_t(\omega \mid \mathbf{x}) = \frac{1}{100}(10,70,20) = (0.10,0.70,0.20)$$

et la classification de la nouvelle donnée, assignée par la forêt, serait celle dont la probabilité est la plus grande, c'est-à-dire, la classe 2.

La figure 5.3 illustre le processus de prévision et de classification de données.

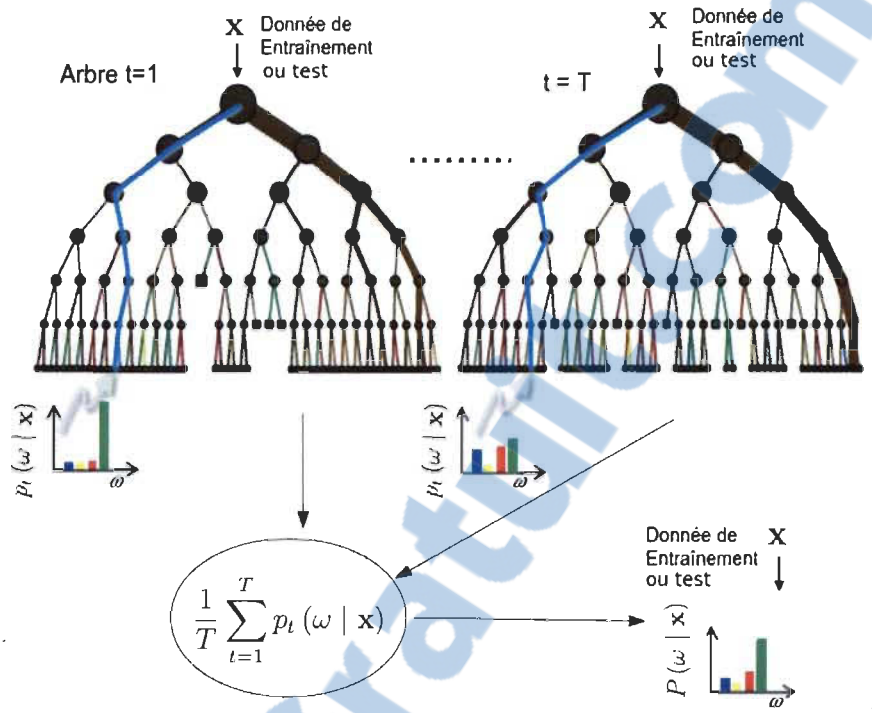


FIGURE 5.3 – Entraînement de la forêt de décision ou classification. Inspiré de [11]

5.4.5 Le bagging et la capacité de généralisation.

Si on utilise le bagging pour la construction de la forêt, il est possible d'estimer la capacité de généralisation de la forêt de décision en utilisant les données hors de la bourse. L'erreur hors de la bourse (OOB error) sera alors simplement calculée comme la proportion de données hors de la bourse dont la prévision finale est incorrecte. Cette erreur est un estimateur valable de la capacité de généralisation de la forêt de décision. Cette procédure s'avère nécessaire avec des ensembles de données très grands avec lesquels l'utilisation de la validation croisée pour estimer la capacité de généralisation serait computationnellement coûteuse.

5.4.6 L'effet des paramètres du modèle de forêt de décision.

Les paramètres qui influencent le comportement d'un modèle de forêt de décision sont :

— La taille T de la forêt.

— Le type du modèle de décision faible.

— La profondeur maximale d'arbre D permis.

— Le montant de randomisation (contrôlé par p_j) ;

On regardera ensuite comment ces paramètres influencent la performance du classeur de forêt de décision.

5.4.6.1 Le nombre d'arbres T de la forêt.

D'après [32] et [41], augmenter le nombre d'arbres dans la forêt améliorera seulement sa capacité de généralisation mais jusqu'à arriver à un point où sa capacité de généralisation est optimale celle-ci ne s'améliore plus considérablement, étant impossible pour le modèle de sur-ajuster les données d'entraînement en ajoutant plus d'arbres. Par conséquent, utiliser quand même un nombre trop grand d'arbres n'affecte pas la capacité de généralisation du modèle, mais il faut tojours considérer son coût computationnel. Par contre, l'augmentation du nombre d'arbres T de la forêt produit des frontières de décision plus souples, qui permettent de bien modeler l'incertitude, tel qu'il est illustré sur la figure 5.4.

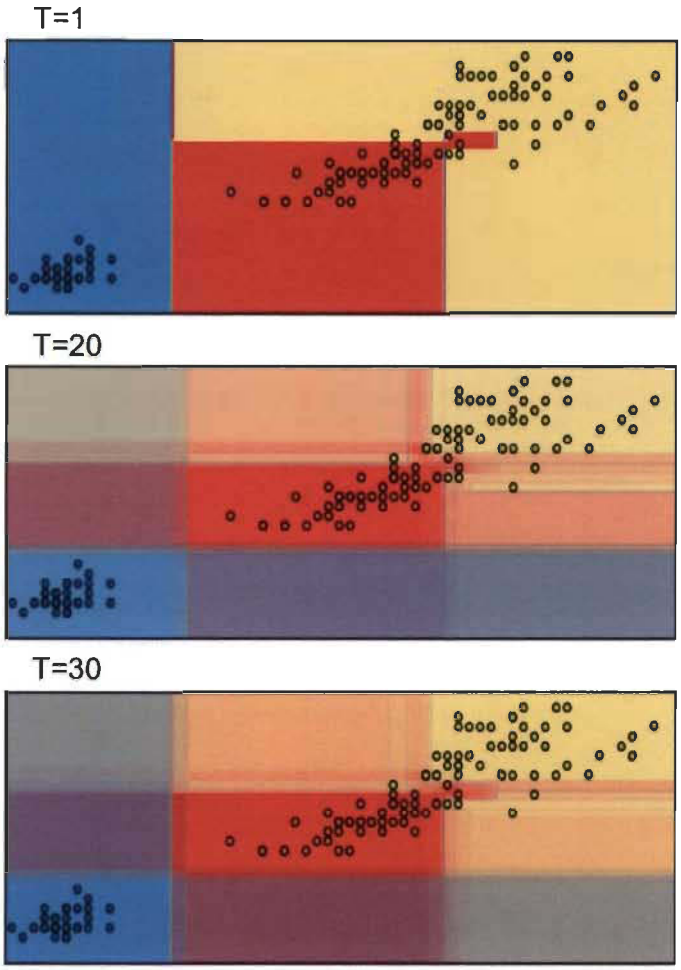


Illustration de l'influence du nombre d'arbres T dans la forêt en utilisant les données des fleurs iris. L'augmentation du nombre d'arbres améliore la capacité de généralisation du modèle, en permettant en outre de capturer l'incertitude.

FIGURE 5.4 – Influence du T.

5.4.6.2 L'effet du type de classeur faible.

[11] mentionne que le type de classeur faible choisi peut influencer fortement la performance de la forêt de décision ; on améliore sa performance en utilisant des classeurs faibles complexes comme des lignes orientées [55, 52] ou des modèles non linéaires, tandis que les modèles de droites perpendiculaires à l'un des axes, comme ceux qu'on a utilisés dans l'exemple de la figure 5.2, produisent des régions rectangu-

laires qui peuvent mener à une mauvaise généralisation. Toutefois, d'après le même auteur, en augmentant le nombre d'arbres T de la forêt, comme il est montré dans la figure 5.4, le résultat sera plus lisse en fixant ce problème. Par ailleurs, les tests alignés à l'un des axes sont computationnellement extrêmement efficaces. De plus, dans les logiciels informatiques, c'est souvent le type de classeur faible qui est implémenté et c'est celui qui sera utilisé par la suite dans notre application d'analyse de texte.

5.4.6.3 La profondeur maximale d'arbre.

Le paramètre D de profondeur d'arbre contrôle la quantité de sur-ajustement. Des arbres trop profonds tendent à sur-ajuster les données d'entraînement, tandis que des arbres trop peu profonds peuvent ne pas adapter suffisamment les données d'entraînement pour offrir une bonne généralisation. D'après [11], bien que l'utilisation de multiples arbres allège le sur-ajustement aux données d'entraînement, il ne le fait pas complètement et, étant donnée que la valeur appropriée de la profondeur D varie selon l'application particulière du modèle, il faut être soigneux en le choisissant.

Il est important de remarquer que la taille de l'arbre peut aussi être contrôlée en établissant le nombre minimal de données d'entraînement nécessaires pour produire un nœud feuille. Par exemple, en fixant le nombre minimal dans 5, l'arbre produit sera moins grand qu'en fixant ce nombre dans 1.

5.4.6.4 Le montant de randomisation (contrôlé par p_j).

Selon l'exposé dans la section 5.4.2, ce paramètre permet de réduire ou d'augmenter la complexité du classeur en diminuant ou en augmentant sa valeur, respectivement. Si la valeur $p_j = \sqrt{p}$, suggérée par [32] et [41], ne permet pas une performance suffisamment bonne, la valeur optimale peut être obtenue par validation croisée.

5.5 Les avantages et les inconvénients des forêts de décision.

5.5.1 Les Avantages.

Selon [11], les forêts de décision peuvent produire, à certaines conditions, des frontières de séparation de marge maximale, comme celles obtenues par les machines à support de vecteurs.

Les forêts de décision permettent de trouver l'importance des caractéristiques pour la classification ou la prévision. Il faut simplement obtenir, pour chaque caractéristique, la somme du décroissement total d'entropie ou le gini index dans chaque arbre de la forêt, puis faire la moyenne des T arbres de la forêt. Ainsi les caractéristiques qui obtiennent une plus grande réduction moyenne de l'hétérogénéité seront les plus importantes pour la classification.

Il n'est pas possible de sur-ajuster les données d'entraînement en mettant trop d'arbres dans chaque forêt, cela aurait seulement un coût computationnel.

Il est possible d'utiliser le même modèle de classification tant avec des problèmes de classification binaire ou de classes multiples ainsi que de valeurs continues.

Ce modèle est efficace pour travailler avec des données à haute dimension (telles que les représentations vectorielles de texte) en améliorant la vitesse de traitement dû au fait qu'il choisit des sous-ensembles de caractéristiques dans la formation de chaque arbre au lieu d'utiliser toutes les caractéristiques existantes.

Il donne un résultat à faible densité contenant seulement les caractéristiques les plus importantes utilisées et, en plus, il permet de bien modéliser l'incertitude.

5.5.2 Les inconvénients.

Malgré tous les avantages que nous venons de mentionner, « Il n'y a pas de déjeuner gratuit (there is no free lunch) dans la statistique : aucune méthode ne dépasse toutes les autres pour tous les ensembles possibles de données » [41].

5.6 Résumé.

Dans ce chapitre, nous avons présenté le modèle d'arbre de décision qui est utilisé comme bloc fondamental de la fonction objectif du modèle de forêt de décision. Nous avons vu en outre, que pour optimiser la fonction objectif, les deux paramètres fondamentaux sont le nombre d'arbres de la forêt et le nombre de caractéristiques échantillonnées dans la construction de chaque nœud des arbres. Ce dernier est l'hyper-paramètre qui nous permettra, dans la pratique, de contrôler la complexité du modèle.

Pour les buts de notre recherche, les caractéristiques significatives du modèle sont, d'une part, sa capacité d'identifier l'importance des mots pour la classification, ce qui nous permettra de faire l'analyse automatique du contenu des documents de texte de chaque classe et, d'autre part, son efficience dans des espaces de haute dimension, comme c'est le cas pour la classification de tels documents. Finalement, étant donné l'utilisation de petits sous-ensembles de caractéristiques pour la construction de nœuds, le modèle de forêt de décision est très efficace et extensible à de grandes quantités de données.

Avec toute la théorie nécessaire, nous pouvons maintenant aller à la mise en œuvre et à l'application des modèles choisis dans notre projet de recherche.

Chapitre 6

L'implémentation

Étant donné que l'un des objectifs du développement du logiciel est son intégration, à titre d'outil plus disponible dans l'application REGASS [17], on a développé deux versions différentes du logiciel de classification : l'une est utilisée dans l'étape de classification du REGASS et, on l'appellera la version 1 ; l'autre est dans l'étape d'analyse et on la qualifiera de version 2. Les deux versions utilisent le même noyau de la fonction de classification qui reçoit les segments de texte, les vectorisent et optimise le modèle choisi, soit la SVM ou la forêt de décision, en montrant un rapport de classification et en permettant de faire un graphique du processus d'optimisation du modèle utilisé.

La version 1 reçoit les segments de texte contenus dans le logiciel REGASS, puis, en utilisant des données externes d'entraînement, situées dans le dossier indiqué par l'utilisateur, elle entraînera le classeur en montrant le rapport et le graphique d'optimisation des données d'entraînement, en classant finalement les données de REGASS avec le classeur optimal construit.

La version 2 est utilisée dans l'étape d'analyse du logiciel REGASS et elle utilise les segments de texte et les étiquettes assignées par certains de ses classeurs disponibles : SOM, ART ou k-means. Contrairement à la version 1, la version 2

sert à classer une des classes, appelée la classe cible, contre le reste des classes, afin d'identifier les caractéristiques (mots) plus importantes pour la classification des segments de cette classe-ci, en offrant, en outre, une représentation graphique de ces termes. Cette version permet, finalement, d'explorer les segments proches à certains des termes plus importants, lesquels peuvent être choisis par l'utilisateur.

6.1 La lecture et le découpage des données.

Dans la version 1 les données utilisées pour optimiser le classeur sont des données externes à REGASS, qui doivent alors se trouver dans un dossier indiqué par l'utilisateur.

Dans la version 2, les seules données utilisées sont celles contenues dans le logiciel REGASS, et elles sont prises de façon automatique par le logiciel.

Après la lecture, les données sont découpées comme suit :$7/10 = 0.70$ comme données d'entraînement et de $3/10 = 0.30$ comme données de test.

6.2 La fonction de classification.

Dans cette section on décrit le développement de la fonction qui fait la classification des documents dans les différentes catégories dont le code d'appel est le suivant..

Appel de fonction :

```
def Classeur(trains_path = "", outf_path = "",
    hyperparam_dict = {} , classeurchoix = 3,
    n_iter_search = 2, seed = 13, **kwargs):
```

6.2.1 Les paramètres.

trains_path : dans la version 1, c'est l'adresse du dossier qui contient des sous-dossiers contenant les fichiers correspondent aux documents ou aux segments de texte, à utiliser pour entraîner le classeur, c'est-à-dire, l'ensemble d'entraînement. Le dossier spécifié est fourni par l'utilisateur et doit contenir des sous-dossiers qui contiennent, chacuns, les documents de texte d'entraînement d'une des catégories dans lesquelles on les classera. Les étiquettes son générées automatiquement pour chaque sous-dossier. Dans la version 2, l'adresse utilisée est une direction fixe interne où se trouvent les segments et les étiquettes de classe assignées par REGASS.

outf_path : dans la version 1 c'est l'adresse où se trouvent les segments du REGASS dont la classe sera retournée. Dans la version 2, l'adresse est passée mais non utilisée.

hyperparam_dict **:** c'est le dictionnaire contenant les valeurs des hyperparamètres qui seront utilisés pour optimiser le classeur, c'est-à-dire les valeurs disponibles pour produire différentes hypothèses (classeurs candidats ayant différentes combinaisons d'hyperparamètres) et ce, afin de choisir celle ayant la meilleure performance.

Classeurchoix **:** c'est la valeur numérique entière correspondant au type de classeur à utiliser. Ce paramètre peut prendre des valeurs d'entiers entre 0 et 3, correspondant à la forêt de décision randomisée, la forêt de décision par recherche exhaustive des hyperparamètres, la SVM randomisée et la SVM par recherche exhaustive, respectivement. Ce paramètre est utilisé par un dictionnaire défini dans la partie inférieure de la fonction, et qui agit comme un switch qui mènera vers l'exécution du code du classeur choisi.

n_iter_search **:** c'est le nombre d'hypothèses à tester en faisant une recherche

randomisée des valeurs optimales des hyperparamètres (voir section 6.2.5.5).

Seed **:** c'est la valeur semence du générateur des valeurs aléatoires.

** ***kwargs*** : il permet de passer des arguments facultatifs, par exemple :

n_e_dct **:** qui est le nombre d'arbres à produire dans les classeurs de forêts de décision.

6.2.2 L'importation et la vectorisation des données.

Les segments de texte sont lus puis placés dans une liste appelée corpus, dont les éléments correspondent aux segments de texte chargés. Par ailleurs, les étiquettes de classe sont mises dans une liste appelée target_l [1]

Commence alors la programmation des tâches d'apprentissage automatique. Pour cela on utilise le module sci-kit learn du langage de programmation python[66]. Le choix de ce module est dû au fait qu'il y a une grande quantité de modèles mise en œuvre ainsi que de tâches de préparation données et de rapport de résultats de disponibles pour l'utilisateur.

On divise les données en deux parties : un ensemble de données d'entraînement, avec 70% des données, et un ensemble de données de test, avec le 30% restant. Pour réaliser ceci, on utilise la fonction cross_validation.train_test_split(), de sci-kit learn, dont le paramètre test_size permet de définir la proportion de l'ensemble de données de test. On obtient ainsi l'ensemble d'entraînement X_train avec ses étiquettes y_train et celui de test X_test avec les siennes y_test.

```
X_train, X_test, y_train, y_test = cross_validation.
    train_test_split(corpus, target_l, test_size=0.30,
    random_state=0)
```

1. Dans la version 2 on utilise une liste additionnelle, appelée target_lbool dans laquelle les étiquettes de classe, différentes de la classe objectif, sont mises à 0.

Pour faire la représentation vectorielle des documents de texte, il est d'abord nécessaire de créer le vocabulaire, c'est-à-dire le sac de mots, à utiliser à partir de l'ensemble d'entraînement ; pour ceci, on crée un objet, de la classe CountVectorizer, nommé vectorizer. Ensuite par la méthode fit_transform() le vectorizer produira un dictionnaire de fréquences des n-grammes utilisés par la méthode fit_transform, afin de produire la matrice de fréquences de termes, ou matrice de fréquences.

Par défaut, la génération du vocabulaire est faite en prenant des n-grammes de taille 1, c'est-à-dire, des mots individuels, toutefois, il s'avère parfois utile de choisir davantage de n-grammes de taille plus grande, (2 par exemple).

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(ngram_range=(1,1))
freq_term_matrix = vectorizer.fit_transform(X_train)
```

Si l'utilisateur choisit d'utiliser la pondération tf-idf, mentionnée dans la section 3.5, il faut créer un nouvel objet de la classe TfidfTransformer(), avec le paramètre norm= "l2" qui fait la pondération tf-idf et qui normalise les vecteurs en utilisant la distance euclidienne.

On a, finalement, la représentation vectorielle de l'ensemble de données d'entraînement, appelée X_train.

```
if kwargs['tf_idf'] == True:
    tfidf = TfidfTransformer(norm="l2")
    tf_idf_matrix = tfidf.fit_transform(freq_term_matrix)
    X_train = tf_idf_matrix
else:
    X_test = freq_term_matrix
```

On doit maintenant vectoriser l'ensemble de données de test. Pour ce faire, il

est nécessaire d'utiliser le vocabulaire qui a déjà été produit avec les données d'entraînement en excluant, par conséquence, les mots de l'ensemble de test qui ne sont pas dans l'ensemble d'entraînement. On doit donc utiliser le même objet vectorizer qui a été construit avec les données d'entraînement pour obtenir la représentation vectorielle de fréquences de termes de l'ensemble de test. Si l'utilisateur souhaite faire la transformation tf-idf, il faut répéter la procédure de pondération faite avec l'ensemble d'entraînement. Finalement, seulement dans la version 1, les segments de texte dedans REGASS sont vectorisés de la même façon que l'ensemble de données de test.

```
freq_term_matrix = vectorizer.transform(X_test)
if kwargs['tf_idf'] == True:
    tfidf = TfidfTransformer(norm="l2")
    tf_idf_matrix = tfidf.fit_transform(freq_term_matrix)
    X_test = tf_idf_matrix
else:
    X_test = freq_term_matrix
```

Les données sont déjà prêtes à être utilisées par le classeur.

6.2.3 Le choix du classeur.

On vérifie, selon la valeur du paramètre Classeurchoix, lequel des quatre classeurs sera utilisé. Pour ceci, on définit un dictionnaire, au moyen duquel il est possible de choisir le bloc de code correspondant au classeur à utiliser.

```
    #Définition des options de classeurs disponibles
options = {0 : rfrandomisee,
           1 : rfgrids,
           2 : svmrandomisee,
           3 : svmgrids,}
rapport, y_pred = options[classeurchoix]()
```

0 : *rfrandomisee* correspond au classeur de forêt de décision randomisée, c'est-à-dire, celle dont les hyperparamètres optimaux sont cherchés de façon aléatoire et non exhaustive (voir sec. 6.2.5.5).

1 : *rfgrids* correspond au classeur de forêt de décision dont la recherche des hyperparamètres optimaux est faite par une recherche exhaustive de toutes les combinaisons possibles des hyperparamètres (grid search).

2 : *svmrandomisee* correspond au classeur de machines à support de vecteurs (svm) en faisant une recherche randomisée des hyperparamètres optimaux (voir sec. 6.2.5.5).

3 : *svmgrids* correspond au classeur de machines à support de vecteurs (svm) dont la recherche des hyperparamètres optimaux est effectué de façon exhaustive (grid search).

6.2.4 Les classeurs.

6.2.4.1 La structure.

Une fois déterminé le classeur à utiliser, on passe à la section de code dans laquelle il est exécuté. Les classeurs ont la structure commune suivante :

i. Définition de la grille de recherche d'hyperparamètres qui contient les valeurs possibles des hyperparamètres entre lesquels sera cherchée la combinaison de

valeurs donnant la performance optimale du classeur.

ii. Création d'une instance du classeur à utiliser avec les paramètres d'initialisation nécessaires.

iii. Entraînement du classeur en utilisant l'ensemble de données d'entraînement.

La procédure consiste à générer des classeurs candidats (hypothèses) pour chacune des différentes combinaisons des valeurs des hyperparamètres, en évaluant ses performances par une validation croisée de trois couches (3 folds cross validation) et, finalement, en choisissant l'hypothèse dont la combinaison des hyperparamètres rend la meilleure performance d'après la mesure de précision.

iv. Test du classeur.

Ceci est fait en classant les données de l'ensemble de test pour bien évaluer la performance du classeur avec des données encore non vues. Dans la version 1 les données du REGASS sont également classées par le classeur optimal.

v. Retour des objets suivants :

(a) Rapport d'optimisation et de classification de l'ensemble de test.

(b) Vecteur de classification des données du REGASS (Version 1) ou vide (Version 2).

(c) Classeur optimal.

(d) Liste des caractéristiques plus importantes.

6.2.4.2 La création d'une instance du classeur à utiliser.

On crée un objet classeur. Pour ce faire, on utilise les modules de sci-kit learn de forêts de décision ainsi que la librairie de machines à vecteurs de support.

Dans le cas des classeurs de forêt de décision, on utilise le constructeur RandomForestClassifier() avec le paramètre random_state =13 pour répéter les résultats

obtenus, autrement l'élection aléatoire de caractéristiques dans la construction des nœuds, produira des résultats différents même si on choisit les mêmes hyperparamètres. Le modèle de machines à vecteurs de support est créé avec le constructeur clf = svm.SVC (). Les deux classeurs font partie du module sci-kit learn, d'où ils doivent être importés.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
clf = RandomForestClassifier(random_state =13)
clf = svm.SVC()
```

6.2.5 L'optimisation du classeur.

6.2.5.1 La grille d'hyperparamètres.

L'optimisation du classeur correspond aux processus décrits dans la section 2. On cherche la combinaison d'hyperparamètres permettant un ajustement optimal du classeur aux données d'entraînement par l'optimisation de la fonction objectif mais en évitant le sur-ajustement. Chaque combinaison d'hyperparamètres testée correspond à un classeur candidat, chacun ayant une complexité différente. Le classeur optimal sera le classeur candidat ayant la meilleure capacité de classification, en utilisant comme mesure de performance la précision moyenne de classification des données d'entraînement par une validation croisée de 3 couches.

Dans le code, il faut d'abord définir l'espace d'hyperparamètres contenant toutes les valeurs possibles que les modèles candidats peuvent utiliser. En sci-kit learn cette définition est faite par un dictionnaire représentant la grille des valeurs possibles. Ce dictionnaire est construit par le logiciel à l'aide de l'interface graphique, en prenant les valeurs possibles des hyperparamètres choisis par l'utilisateur et en étant passé comme paramètre de la fonction de classification Classeur().

On décrit ensuite les hyperparamètres possibles de chaque classeur (Prises de la documentation de[66]). On montre seulement ceux qui peuvent être modifiés par l'utilisateur du logiciel, le reste est laissé avec des valeurs par omission. Plus d'information peut être trouvée dans le site web du sci-kit learn [66].

6.2.5.2 Les hyperparamètres de la forêt de décision.

Les hyperparamètres pouvant être variés dans les classeurs de forêts de décision sont désignés ainsi :

Les hyperparamètres spécifiques de la forêt :

n_estimators **:** int. Le nombre d'arbres de décision dans la forêt.

max_features **:** int, float, string ou aucun, optionnel (défaut="auto"= $sqrt\,(n\_features)$). Le nombre de caractéristiques à tester dans la construction de chaque nœud.

Les hyperparamètres spécifiques par arbre :

criterion : string, optional "gini" où "entropie" (default="gini").

max_depth **:** int ou None (default= None). La profondeur maximale de l'arbre. Si aucune n'est spécifiée, les nœuds sont construits jusqu'à ce que toutes les feuilles soient pures ou jusqu'à ce que toutes les feuilles contiennent moins d'exemples que le nombre minimal spécifié par le paramètre min_samples_split.

min_samples_split **:** int (défaut=2). Le nombre minimal d'exemples requis pour diviser un nœud interne.

min_samples_leaf : int, optionnel (défaut=1). Le nombre minimal d'exemples dans la création des nouvelles feuilles. Une fente est dite écartée si, dans la suite, une des feuilles contient moins de min_samples_leaf exemples.

bootstrap : boolean, optionnel (défaut=True). Si des échantillons pris par la stratégie bootsrap sont employés en construisant des arbres.

6.2.5.3 Les hyperparamètres de la SVM.

Les hyperparamètres pouvant être variés, dans les classeurs de SVM de décision sont :

kernel : string, optionnel (défaut = 'rbf'). Il spécifie le type de noyau à utiliser dans l'algorithme. Valeurs disponibles au logiciel : «linear », « rbf ».

C : float, optionnel (défaut = 1.0) Paramètre de pénalisation C.

***gamma* :** float, optionnel (défaut = 0.0) Coefficient du noyau pour la SVM « rbf ».

6.2.5.4 La recherche exhaustive des paramètres optimaux.

Dans ce type de recherche, toutes les combinaisons possibles des valeurs, définies dans la grille d'hyperparamètres, sont testées jusqu'à trouver celle qui permet d'obtenir la meilleure performance. La performance peut être mesurée avec les mesures mentionnées dans la section 2.4.8. La fonction sklearn.grid_search.GridSearchCV() utilise, par défaut, l'exactitude (accuracy) ou, plus précisément, son complément appelé l'erreur de classification, avec une stratégie de validation croisée de trois couches pour trouver ces valeurs optimales des hyperparamètres.

Cette type d'optimisation peut être computationnellement coûteuse, toutefois, elle offre la meilleure performance possible pour les combinaisons d'hyperparamètres permises.

Voici des exemples des grilles de valeurs des hyperparamètres pour les classeurs de forêt de décision et de SVM.

```
hyperparam_dict = {"max_depth": None,
"max_features": [50, 75, 100],
"min_samples_split": [2,5],
"min_samples_leaf": [1,3],
"bootstrap": [True, False],
"criterion": ["gini", "entropy"]}
```

```
hyperparam_dict = {'kernel': ['rbf', 'linear'],
                   'C': [.001,.01,.1, 1,10,100, 1000]
                  'gamma': [0.5,0.25,0.125,0.0625,0.0325] }
```

6.2.5.5 La recherche randomisée d'hyperparamètres optimaux.

On cherche la combinaison optimale d'hyperparamètres dans un sous-ensemble de combinaisons d'hyperparamètres possibles, créées de façon aléatoire à partir de la grille des hyperparamètres.

Ce processus de génération aléatoire de combinaisons d'hyperparamètres sera effectué de manière itérative le nombre de fois souhaité par l'utilisateur, disons k, avec $k < n$, où n est le nombre total de possibles combinaisons d'hyperparamètres. Le résultat sera la meilleure combinaison trouvée dans le nombre total d'itérations. L'utilisateur passera, à l'aide de l'interface graphique, le nombre de combinaisons d'hyperparamètres à tester dans la recherche aléatoire par le paramètre n_iter_search $= k$.

Cette type de recherche peut être plus rapide que la recherche exhaustive; bien que la performance du classeur optimal obtenu est un peu inférieure à celle de la recherche exhaustive, la différence est généralement d'importance moindre.

6.2.5.6 L'entraînement du classeur.

Une fois défini l'espace d'hyperparamètres, et créé l'instance du classeur on les utilise pour créer l'objet (random_searh ou grid_search, selon la stratégie d'optimisation à utiliser) qui contiendra toutes les directives permettant d'effectuer l'optimisation du classeur, incluant le classeur à utiliser et même, dans le cas de l'optimisation par stratégie de recherche randomisée, le nombre de recherches randomisées à faire (n_iter_search).

On utilise ensuite l'objet pour optimiser notre classeur avec les données d'entraînement.

Stratégie d'optimisation par la recherche randomisée :

```
clf_srch = RandomizedSearchCV( clf, param_distributions =
    hyperparam_dict, n_iter = n_iter_search)
clf_srch.fit(X_train.todense(), y_train)
```

Stratégie d'optimisation de recherche exhaustive :

```
clf_srch = GridSearchCV(clf, param_grid = hyperparam_dict)
clf_srch.fit(X_train.todense(), y_train)
```

6.2.5.7 Le test du classeur.

Une fois optimisé le modèle, on utilise le modèle optimal pour classer les données de test et produire un rapport de classification. Dans la version 1, les données contenues en REGASS sont aussi classées par le modèle et les valeurs assignées sont passées à REGGAS comme le vecteur numérique y_out.

```
y_true, y_pred, y_out = y_test, clf_srch.predict(X_test.
    todense()), clf_srch.predict(X_out.todense())
```

6.2.5.8 Les objets retournes.

Finalement la fonction retourne les objets mentionnés dans la section 6.2.4.1.

Ces objets retournés serviront au logiciel pour effectuer d'autres tâches, comme la représentation graphique de la courbe d'apprentissage et des mots les plus importants pour la classification.

6.3 Résumé.

Dans ce chapitre, on a présenté la mise en œuvre du cœur du logiciel pour la classification automatique de documents de texte. Nous nous sommes limitées à exposer la programmation de la fonction qui fait la classification des données et qui est attachée à la sélection et à l'optimisation des modèles de décision. Ceci été essentiel car c'est la partie véritablement significative dans le cadre de notre recherche.

On a aussi détaillé les deux versions qui ont été intégrées au logiciel REGASS, lesquelles partagent le même noyau et qui ont des différences minimales. La version 1 permet d'entraîner le modèle avec des documents de texte externes à REGASS, situés dans un répertoire fourni par l'utilisateur, puis elle classe les documents de texte dedans REGASS en utilisant le modèle entraîné. La version 2 classe les documents de texte internes de REGASS avec les catégories produites au moyen de certains des algorithmes non supervisés déjà mis en œuvre dans ce logiciel.

On a aussi décrit comment, pour optimiser les modèles, on prend au hasard 70% des données disponibles comme données d'entraînement et 30% comme données de test. La fonction de classification a besoin de savoir quel modèle sera utilisé, ainsi que quelle stratégie d'optimisation sera mise en marche : la recherche exhaustive ou la recherche randomisée. En outre, elle doit savoir, selon le modèle choisi, les hyperparamètres à tester. Toutes ces options sont à la disposition de l'utilisateur ; il

s'aide d'une interface graphique que l'on présente dans le prochain chapitre.

Chapitre 7

L'interface de l'Utilisateur

Dans ce chapitre, on décrit le développement d'une interface graphique qui permet l'utilisation facile du logiciel de classification qui a été intégrée au logiciel REGASS [17]. Étant donné que le projet a été développé sur un ordinateur ayant un système d'exploitation basé sur la plate-forme Linux, et que l'intégration au logiciel REGASS devait être faite sur Windows, le développement de l'interface a été fait en utilisant les librairies wxPython, « une liaison (binding) de la bibliothèque de la classe wxWidgets C++ avec le langage de programmation python » [83]. Cette librairie a été choisie parce qu'elle permet la compilation multiplate-forme de l'interface graphique « donnant aux applications un aspect et une sensation véritablement natifs parce qu'elle emploie l'API native de la plate-forme plutôt qu'imiter le GUI »[84] ; elle est en outre gratuite et de code ouvert (open source).

7.1 La fenêtre principale.

FIGURE 7.1 – Fenêtre principale lors de l'exécution du logiciel sur Linux.

Fonctions :

— Dans la version 1, ce logiciel permet de choisir le dossier principal en appuyant sur le bouton dossier ou en écrivant la route vers lui. Dans la version 2, le bouton s'appelle catégorie et permet de choisir la classe cible qui sera classée contre le reste des classes.

— Après l'optimisation du modèle, la fenêtre montre le rapport d'optimisation contenant :

i. Le temps qu'a pris l'entraînement du classeur.

ii. Les paramètres du classeur optimal.

iii. Le rapport de classification qui montre la matrice de confusion pour la classification des données de test en utilisant les mesures de performance mentionnées dans la section 2.4.8.

iv. Après la réalisation des graphiques des mots plus importants, est affichée la liste des mots au-dessus du rapport de classification.

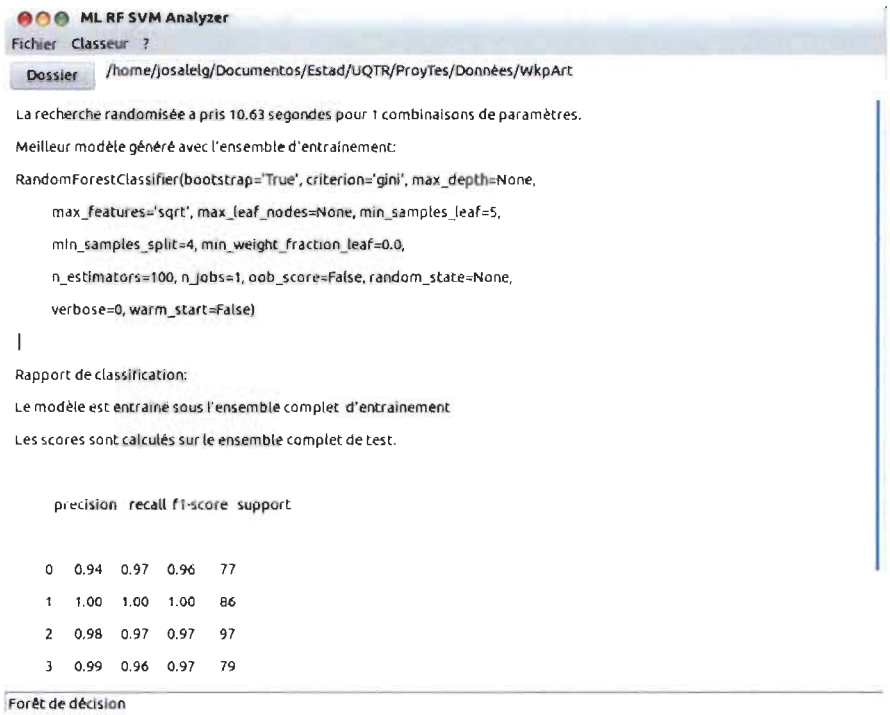


FIGURE 7.2 – Fenêtre principale lors de la présentation du rapport d'entraînement.

7.2 Les options pré-traitement et transformation du texte.

La seule fonction active au démarrage du logiciel est le Classeur>Options, ce qui permet à l'utilisateur de choisir certaines des options de pré-traitement et de transformation du texte, mentionnées au chapitre3 ; ceci se fait à l'aide du formulaire montré dans la figure 7.3 [1].

1. Les boutons du logiciel sont affichés dans la langue du sytème d'explotation (linux en español dans l'image).

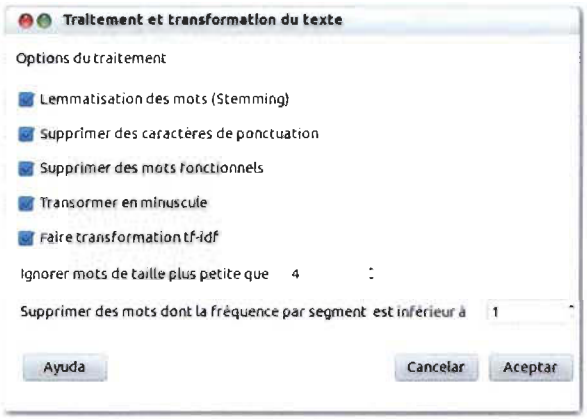


FIGURE 7.3 – Formulaire du prétraitement et transformation du texte.

7.3 L'entraînement du classeur.

Une fois choisies les options de pré-traitement et de transformation, pour entraîner le classeur désiré, il faut choisir entre deux options, Random Forest ou SVM, contenues dans l'option Classeur, du menu situé dans la partie supérieure de la fenêtre principale, c'est-à-dire : Classeur>RandomForest ou Classeur>SVM.

Il apparaît alors un formulaire dans lequel il est possible de choisir les hyperparamètres à tester, dans le processus d'optimisation du classeur choisi.

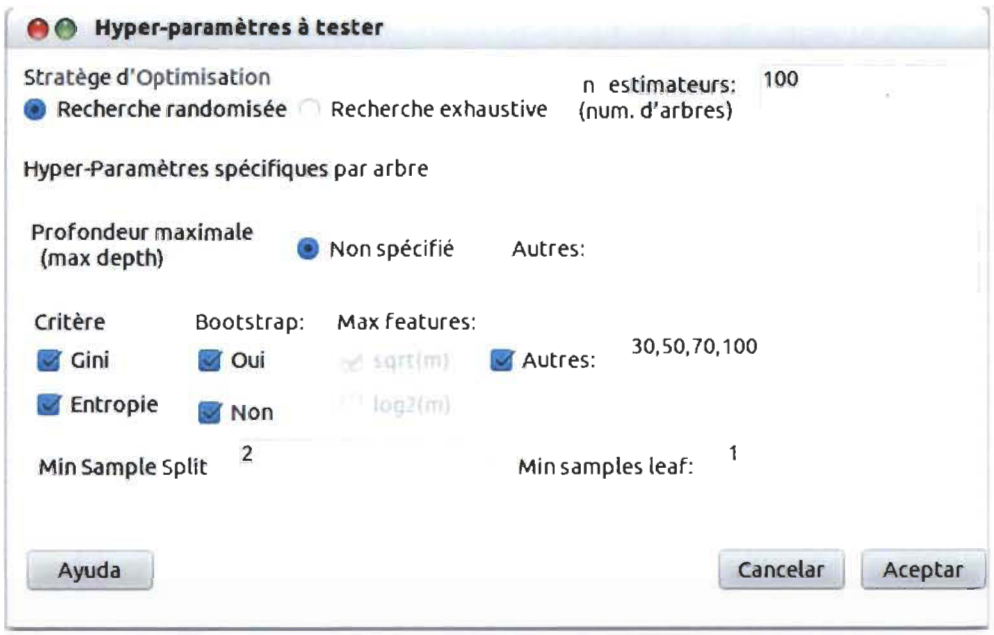
7.3.1 La stratégie d'optimisation.

La première section des formulaires d'élection des hyperparamètres permet de choisir le type de recherche des hyperparamètres optimaux du classeur : exhaustive ou randomisée.

La recherche exhaustive : Là seront testées toutes les combinaisons possibles d'hyperparamètres pour trouver celle qui optimise la performance du classeur.

La recherche randomisée : Là sera testé un sous-ensemble, choisi au hasard, du total des combinaisons possibles d'hyperparamètres pour que soit choisie la com-

binaison permettant d'obtenir la meilleure performance du classeur. Le nombre de combinaisons à tester sera défini par l'utilisateur dans une boîte de dialogue qui apparaîtra immédiatement après le formulaire d'hyperparamètres, tel que montré dans la figure 7.5.



Dans l'exemple il y aurait un total de 16 combinaisons possibles.

FIGURE 7.4 – Formulaire d'élection d'hyperparamètres à tester de la forêt de décision.

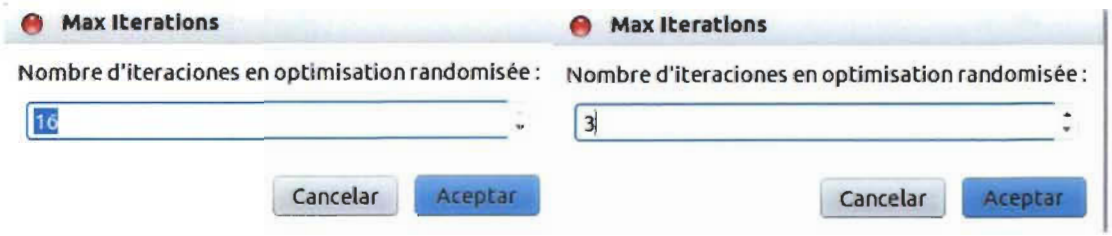


FIGURE 7.5 - Boîte de dialogue permettant de choisir le nombre total de combinaisons d'hyperparamètres à tester par la recherche randomisée.

7.4 Le formulaire d'hyperparamètres de la forêt de décision.

Le formulaire permet, en plus de la sélection de la stratégie d'optimisation, de définir les hyperparamètres à tester disponibles dans le processus d'optimisation du classeur.

Fonctions :

i. Il permet d'établir le nombre d'estimateurs (arbres) qui constitueront la forêt.

ii. Il permet de spécifier la profondeur de l'arbre, et s'il n'est pas spécifié, les arbres seront crus jusqu'à avoir le nombre de données minimal défini dans le champ Min samples leaf.

iii. Il permet de choisir la mesure gini index ou l'entropie.

iv. Il permet d'utiliser, ou non, l'échantillonnage bootstrap.

v. Max features : il permet de définir le nombre de dimensions ou caractéristiques à choisir au hasard dans la construction des nœuds, y compris les options $sqrt(m) = \sqrt{m}$ et de $log_2(m)$, où m c'est le nombre total de dimensions des données d'entraînement. D'autres valeurs peuvent être choisies en les écrivant dans le champ Autres, en les séparant par virgule, par exemple : si m=10,000 et Autres : 200,250, 300, on testerait la prise au hasard de 200, 250 et 300 caractéristiques dans la construction des nœuds des arbres de la forêt.

vi. Min sample split : il permet de définir le nombre minimal d'exemples pour effectuer une division, c'est-à-dire, pour ajouter un nœud.

vii. Min samples leaf : il permet de définir le nombre minimal d'exemples pour introduire une feuille en construisant les arbres.

7.5 Le formulaire d'hyperparamètres de la SVM.

Le formulaire d'hyperparamètres du SVM possède d'autres options qui permettent de définir les hyperparamètres à combiner dans le processus d'optimisation du modèle.

Options :

i. Il permet de choisir le type de noyau (kernel) à tester : linear, Rbf (Radial basis function) ou tous les deux.

ii. Il permet de définir les valeurs à tester du paramètre C.

iii. Si le kernel rbf est choisi, il permet de définir des valeurs à tester du paramètre γ.

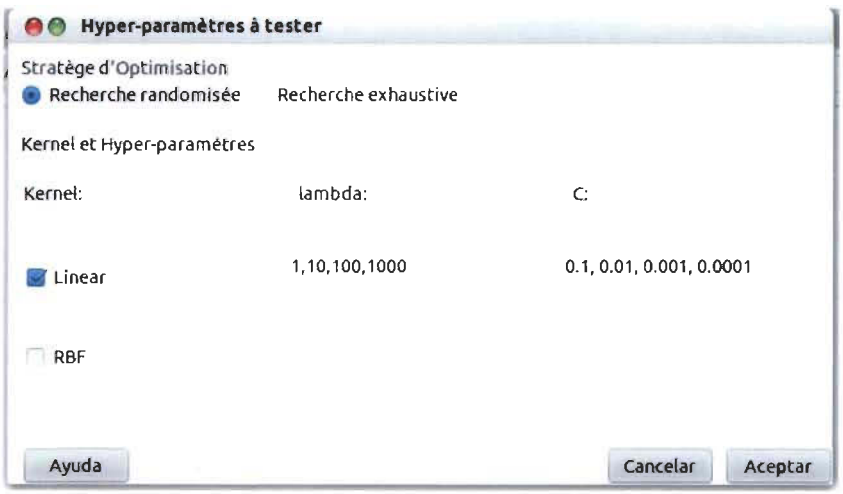


FIGURE 7.6 – Formulaire d'élection d'hyperparamètres à tester de la SVM.

7.6 La graphique de la courbe d'apprentissage.

Le menu Classeur>Courbe d'apprentissage permet à l'utilisateur de faire une représentation graphique du processus d'optimisation du modèle choisi. La représentation graphique est possible seulement quand un seul des hyperparamètres a

plusieurs valeurs à tester dans le formulaire d'hyperparamètres. Le graphique montrera la performance des différentes valeurs testées.

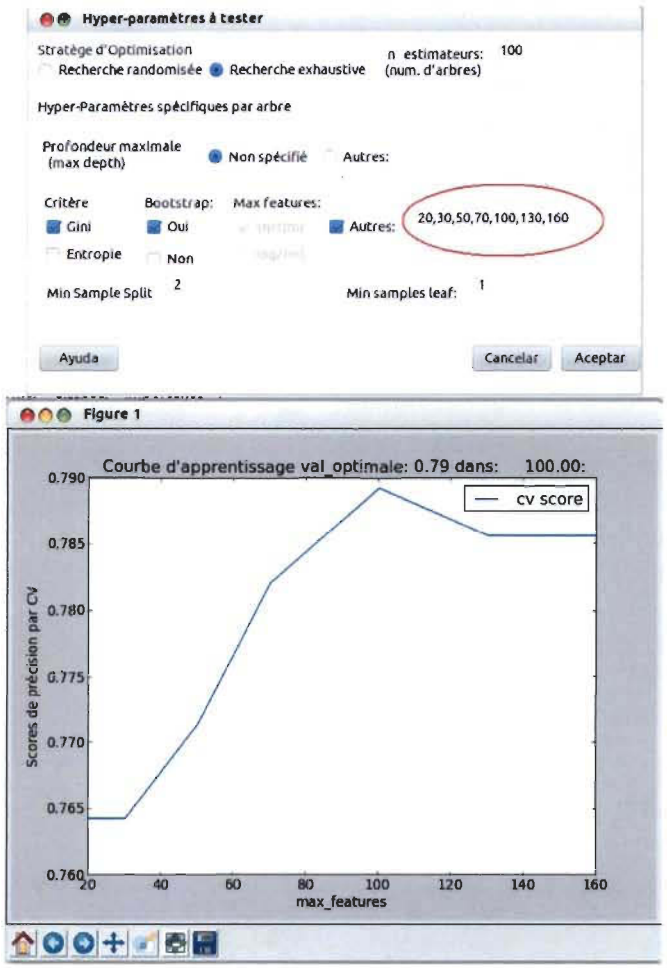


FIGURE 7.7 – Grille et graphique de la Courbe d'apprentissage en variant l'hyperparamètre max features.

7.7 La graphique d'Importance des termes.

L'option Importance>Mots Graphe exécute une représentation graphique des mots plus importants pour la classification. L'utilisateur peut choisir le nombre de mots à montrer entre un et cinquante. La liste sera aussi affichée sous le rapport de classification.

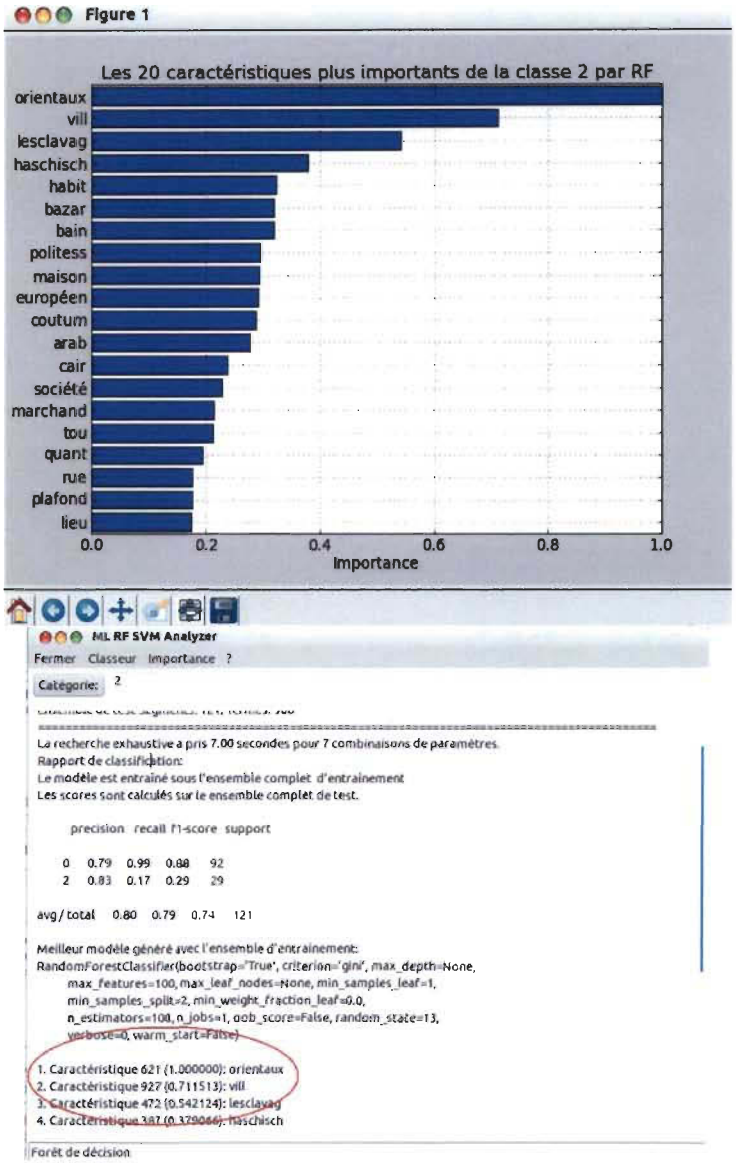


FIGURE 7.8 – Importance des termes. Graphique et liste.

7.8 L'importance des termes.

Une fois que l'utilisateur a identifié les termes les plus importants pour la classification, il peut chercher les segments qui contiennent les mots choisis à partir de l'option Importance>Segments.

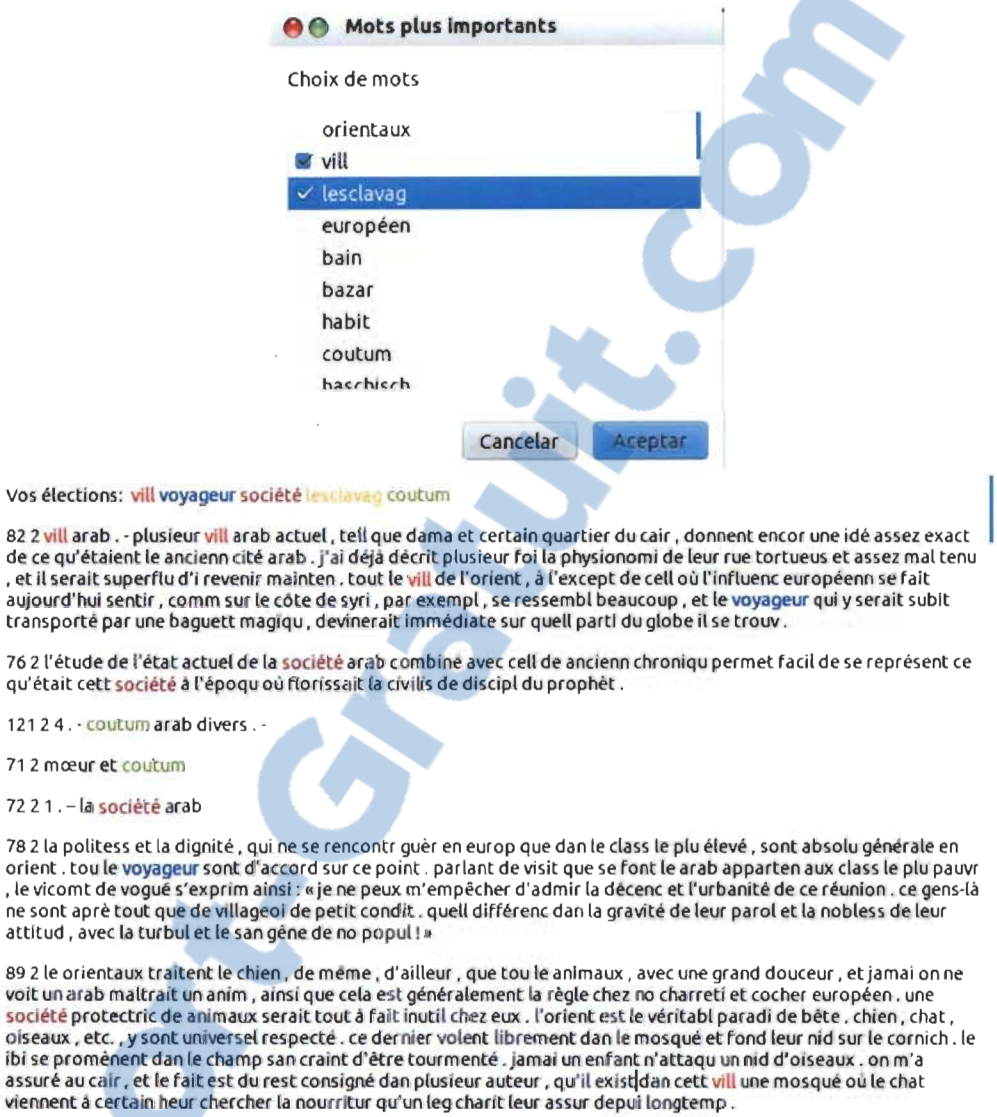


FIGURE 7.9 – Importance des termes. Graphique et liste.

7.9 Résumé.

Dans ce chapitre, nous avons présenté l'interface graphique qui permet à l'utilisateur, par l'intermédiaire du menu situé dans la partie supérieure de la fenêtre principale, d'utiliser les différentes options disponibles. Tout d'abord, il faut choisir les options de prétraitement de texte. Immédiatement après, c'est possible de déter-

miner le modèle à utiliser, que ce soit la SVM ou la forêt de décision. On peut ensuite choisir la stratégie d'optimisation, soit celle de recherche exhaustive, dans laquelle on teste toutes les combinaisons des hyper paramètres, soit celle de recherche au hasard, dans laquelle on teste un sous-ensemble du total de combinaisons afin de trouver le modèle candidat qui rend la meilleure performance. Par la suite, l'utilisateur peut définir les hyperparamètres à tester dans le processus d'optimisation. Il est, en outre, possible de faire la courbe d'apprentissage qui permet à l'utilisateur de voir graphiquement le processus d'optimisation en variant un seul hyperparamètre. Cette fonctionnalité est utile parce qu'elle permet d'identifier les régions qui offrent les valeurs optimales de classification. À la fin du processus d'optimisation, l'interface graphique affiche le rapport de classification qui contient les valeurs des hyperparamètres et la matrice de confusion du modèle optimal faite avec les données de test. Il est possible aussi de visualiser un graphique des mots les plus importants. Finalement, l'utilisateur peut chercher les segments de texte qui contiennent certains des mots les plus importants trouvés par le modèle.

Maintenant que le logiciel est expliqué, passons à l'analyse de texte sur nos données de recherche.

Chapitre 8

L'expérimentation

8.1 Introduction.

Le but de l'expérimentation est de démontrer la pertinence de l'utilisation des arbres de décision et des SVM pour l'identification des mots clés du texte appartenant à une certaine catégorie. Ces mots auraient une importance particulière dans le processus de séparation des segments de la catégorie ciblée du reste des segments en permettant, en plus, d'avoir une idée des principaux sujets abordés dans celle-là.

Pour tester l'hypothèse posée, on essayera de faire ressortir le thème de chaque partie d'un livre, ainsi que les idées développées dans ce thème. Les expériences et les tests ont été effectuées sur le livre « La civilisation des arabes», livre 4 de Gustave Le Bon [49].

8.2 La procédure.

Dans un premier temps, nous avons effectué un traitement manuel sur le texte. Ce traitement consiste à diviser le livre en parties distinctes, chacune correspondant à un des chapitres, et à en épurer le vocabulaire.

Dans un deuxième temps, on a fait une segmentation du livre par paragraphe, de

sorte que chaque segment de texte corresponde à un paragraphe. Puis, on a produit un fichier de texte .txt chaque ligne contenant l'information suivante : numéro de ligne ou segment, numéro de chapitre et segment de texte. Un exemple de la structure du fichier de texte est montré dans la figure 8.1. Ce fichier est utilisé par le logiciel de classification, qui interprète les segments de texte comme les données et le numéro de chapitre comme l'étiquette de classe, c'est-à-dire que, chaque paragraphe appartenant au premier chapitre aura l'étiquette de classe 1, tandis que chaque paragraphe appartenant au chapitre 2, aura l'étiquette de classe 2, etc.

```
1 1 Chapitre I.
2 1 Les Arabes nomades et les Arabes sédentaires des campagnes.
3 1 1. Reconstitution de la vie des anciens Arabes.
```

FIGURE 8.1 - Structure du fichier .csv.

Dans un troisième temps, l'optimisation des classificateurs est effectuée en essayant d'avoir la meilleure précision de classification possible en choisissant une des catégories comme catégorie cible et en utilisant une stratégie de classification d'une contre toutes (one vs all), afin d'identifier les segments de texte comme appartenant à la catégorie cible ou au reste des catégories.

Finalement, on observe et on analyse les caractéristiques les plus importantes en ce qui a trait à la classification par le modèle optimal, fait tant par les forêts de décision que par SVM. Le résultat permettra d'identifier les mots qui contribuent le plus à la séparation de la catégorie cible du reste des catégories, c'est à dire à ceux qui caractérisent possiblement le mieux sa ou ses thématiques.

L'environnement de tests possède les caractéristiques suivantes :

— PC avec microprocesseur double cœur à 2.50 GHz.

— Mémoire vive à 8.00 GB.

— Système d'exploitation Linux Ubuntu 12.04 (precise) à 32-bit.

8.3 Le prétraitement du texte.

8.3.1 L'élimination à main de segments non originaux.

Les segments dits non originaux sont des segments qui correspondent plutôt à l'édition numérique du livre qu'au contenu original. On a enlevé à la main les segments de texte suivants :

— Retour à la table des matières.

— Retour à la table des figures (ordre numérique sur le site web).

— Gustave Le Bon (1884), La civilisation des Arabes. Livre IV (placé en tête de page).

— téléchargeable sur le site web : Les Classiques des sciences sociales, section Auteurs classiques : Gustave Le Bon (1841-1931) :

— La civilisation des Arabes (1884). Dans l'édition papier de 1980 apparaît à la page.

— la figure # nombre.

— Planche couleurs # nombre.

8.3.2 Le nettoyage du texte.

Pour améliorer le résultat de classification, on a réalisé, à l'aide du menu options du logiciel, les opérations suivantes :

— Transformation en minuscule.

— Suppression des mots ayant moins de 3 caractères.

— Élimination des mots fonctionnels.

— Suppression des caractères de ponctuation.

— Lemmatisation des mots.

8.4 La stratégie d'optimisation.

Dans cette section, on décrit la procédure d'optimisation des classeurs pour analyser le contenu de chaque chapitre.

8.4.1 L'optimisation des SVM.

On utilisera le modèle de SVM linéaire, et ce, pour les raisons suivantes :

i. Puisque l'espace vectoriel produit avec des données de texte est généralement à grande dimension, en produisant une matrice à faible densité, il n'est pas généralement nécessaire d'utiliser la SVM avec des noyaux (kernels) pour bien séparer les points dans l'espace.

ii. Notre intérêt est de connaître l'importance des caractéristiques (mots) pour la classification. La SVM linéaire rend possible l'interprétation de l'importance des caractéristiques comme la valeur absolue de chaque coefficient du vecteur de poids $\mathbf{w}$ trouvé. Par contre, l'utilisation de la SVM non linéaire, c'est-à-dire l'utilisation des fonctions noyaux, éviterait la réalisation de cette tâche.

Pour optimiser le classeur, étant donné qu'on utilisera des SVM linéaires, le seul hyperparamètre qu'il faut utiliser est le paramètre de pénalisation C. Pour obtenir la valeur optimale de l'hyperparamètre C, on suivra la stratégie d'optimisation suggérée par [60](en considérant $C = \frac{1}{\lambda}$). On effectue la recherche en utilisant les valeurs suivantes :

λ	C:
327.68	0.0031
163.84	0.0061
81.92	0.0122
40.96	0.0244
20.48	0.0488
10.24	0.0977
5.12	0.1953
2.56	0.3906
1.28	0.7813
0.64	1.5625
0.32	3.1250
0.16	6.25
0.08	12.50
0.04	25
0.02	50
0.01	100
0.005	200
0.0025	400

Pour explorer l'espace d'hyperparamètres conformément à la stratégie mentionnée, on choisira de faire l'optimisation exhaustive pour que le logiciel teste toutes les options introduites en faisant une validation croisée de 3 couches, avec l'ensemble d'entraînement, pour estimer la précision moyenne de classification de chaque hyperparamètre testé (fig 8.2).

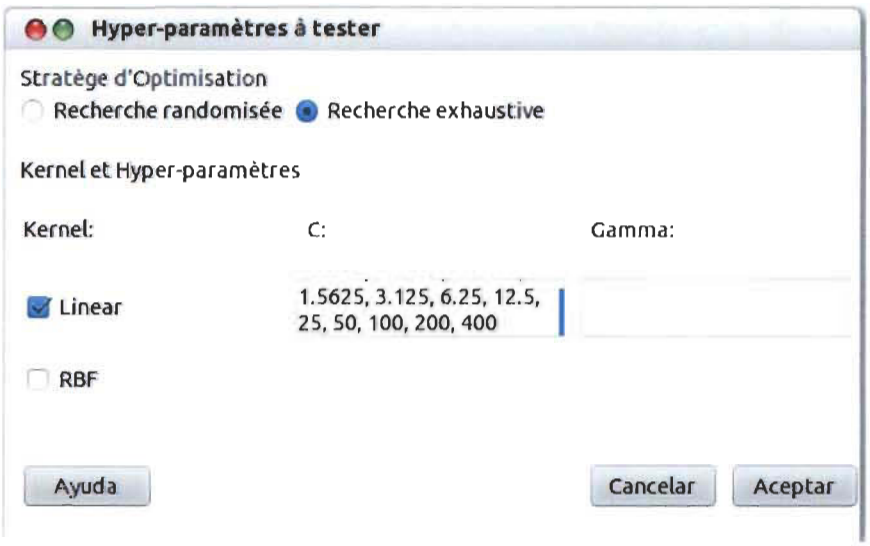


FIGURE 8.2 – Choix d'options d'optimisation du modèle SVM

Lorsque le processus d'optimisation est terminé, le logiciel montrera le rapport de classification de l'ensemble de test résultant et, puisque nous aurons fait varier qu'un seul hyperparamètre, il sera possible de regarder graphiquement le processus d'apprentissage qu'on vient de réaliser, et ce, grâce à l'aide de la fonction courbe d'apprentissage comme il est montré dans la figure8.3, qui correspond à la classe 1. Il est important de noter que la précision affichée par le rapport de classification ne coïncide pas nécessairement avec celle montrée par la courbe d'apprentissage, étant donné que le rapport est calculé avec l'ensemble de données de test tandis que la courbe est calculée avec les données d'entraînement.

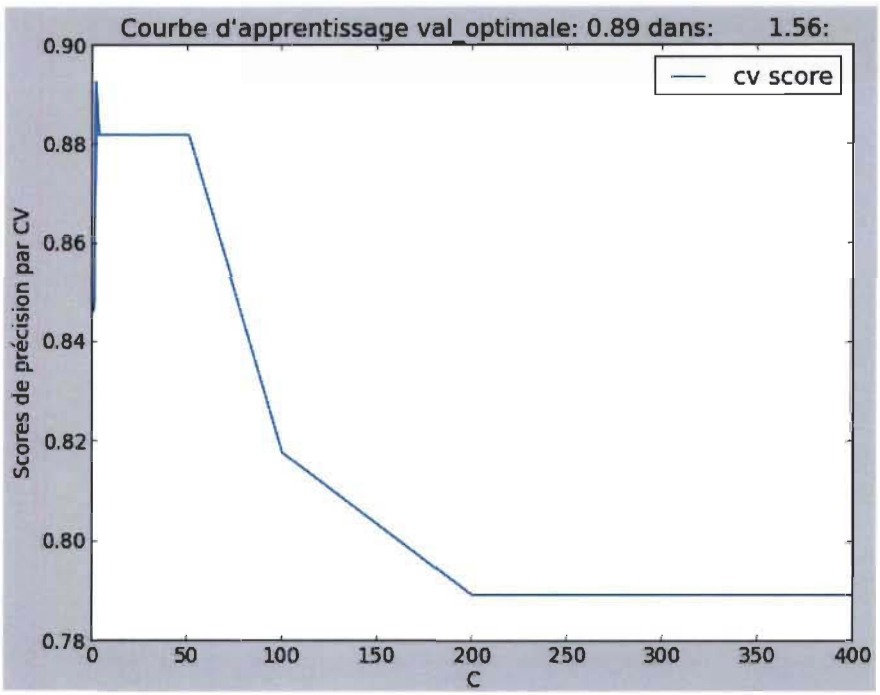


FIGURE 8.3 – Courbe d'apprentissage de la première optimisation

En suite, pour affiner le résultat précédent, on effectuera une seconde recherche parmi les valeurs situées autour de la valeur optimale trouvée par la première recherche, c'est-à-dire, autour de la valeur 1.56 de notre exemple avec la classe 1. Les valeurs à explorer sont :

0.55, 0.65, 0.75, 0.85, 0.95, 1.05, 1.15, 1.25, 1.35, 1.45, 1.55,

1.65, 1.75, 1.85, 1.95, 2.05, 2.15, 2.25, 2.35, 2.45, 2.55

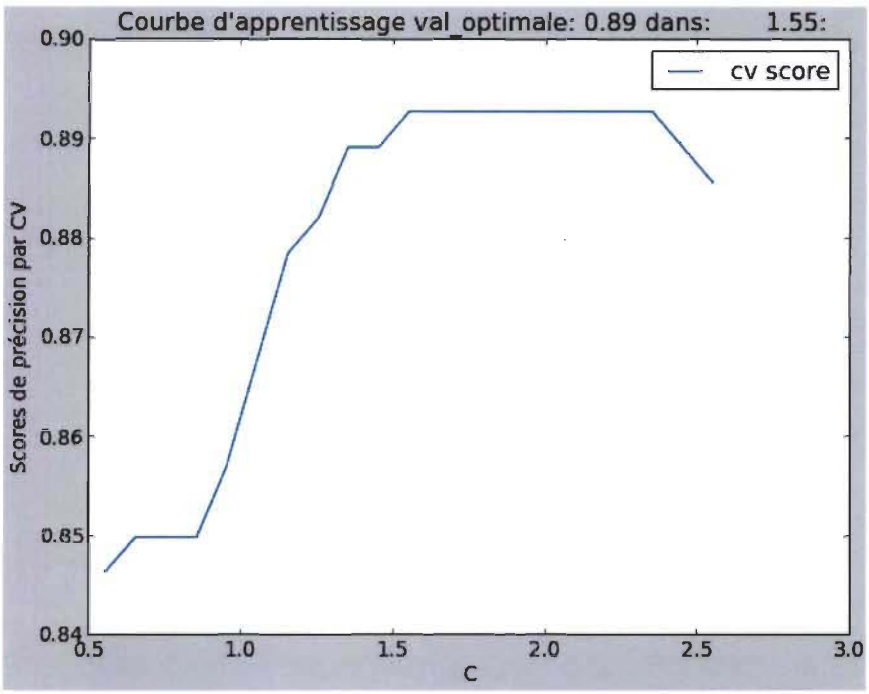


FIGURE 8.4 – Courbe d'apprentissage de la deuxième optimisation

On observe encore la courbe d'apprentissage (fig 8.4) et on remarque que, pour la classe 1, le paramètre de pénalisation optimal C est de 1.55.

8.4.2 L'optimisation des forêts de décision.

Pour optimiser les forêts de décision, nous utiliserons les propriétés mentionnées dans la section 5.4.6, relatives au nombre d'arbres de la forêt et au nombre de caractéristiques utilisées dans la construction de chaque nœud. On basera alors la stratégie d'optimisation du classeur sur le nombre de caractéristiques prises dans la construction de chaque nœud en utilisant, par ailleurs, un nombre suffisamment

grand d'arbres pour améliorer la capacité de prévision. Il est important de mentionner que nous n'utiliserons pas les autres paramètres mentionnés dans la section 5.4.6 étant donné que :

i. L'effet de l'utilisation de la profondeur de l'arbre est similaire à l'utilisation du nombre de caractéristiques pour la construction des nœuds.

ii. Le seul type de classeur faible disponible dans la librairie sklearn, avec laquelle a été conçu le logiciel, utilise uniquement des classeurs faibles des droites parallèles à l'un des axes. À noter que l'utilisation de classeurs faibles plus complexes pourrait éviter l'identification des caractéristiques plus importantes pour la classification (ce qui est l'objectif de la recherche).

La taille de la forêt. Pour déterminer le nombre d'arbres dans la forêt, nous avons classé chaque chapitre contre le reste, en fixant le nombre de caractéristiques par nœud d'arbre dans $\sqrt{p}$ avec p égale au nombre total de caractéristiques, en augmentant le nombre d'arbres et en traçant la courbe de décision pour voir la capacité d'apprentissage obtenue par le modèle. En observant les résultats obtenus, y compris dans l'annexe, il semble qu'il n'y a pas beaucoup de variations à partir de 100 arbres. En considérant que l'ensemble de données n'est pas très grand, nous avons décidé d'utiliser 300 arbres pour assurer un bon rendement, et ce, en variant le nombre de caractéristiques dans la construction des nœuds.

Le nombre de caractéristiques par nœud. En suivant une procédure semblable à celle utilisée dans la section précédente avec les SVM, on testera, en utilisant une recherche exhaustive, des valeurs différentes du nombre de caractéristiques prises dans la construction des nœuds des arbres. À partir du nombre total de caractéristiques qui, après le processus de pré-traitement du texte, est de 969 mots différents, on prend comme base sa racine carrée, c'est-à-dire $\sqrt{968} \approx 31$ et on choisit des nombres qui se situent autour de cette valeur pour varier la complexité du modèle

en choisissant, finalement, celui qui permet d'obtenir la meilleure performance. Les valeurs à essayer sont les suivantes :

5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 140, 160, 180, 200

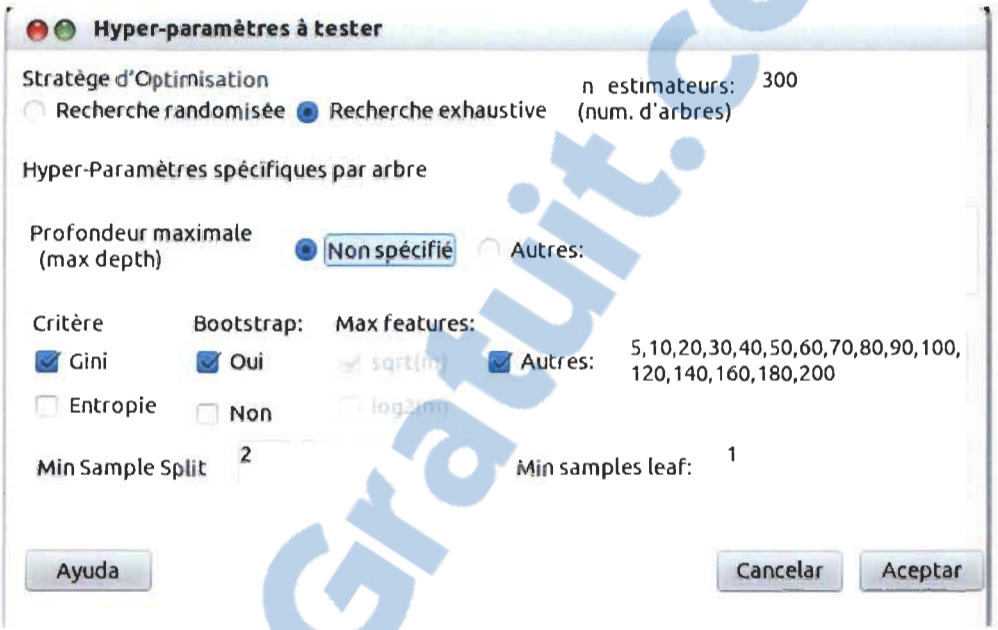


FIGURE 8.5 – Choix d'options d'optimisation du modèle SVM

De la même façon qu'avec les SVM, lorsque le processus d'optimisation est terminé, le logiciel montrera le rapport de classification de l'ensemble de test résultant et, puisque nous aurons fait varier un seul hyperparamètre, il sera possible de regarder graphiquement le processus d'apprentissage. Encore une fois, en observant la courbe d'apprentissage pour la classe 1 (fig 8.6), on détermine que le paramètre optimal de caractéristiques à utiliser à chaque nœud est de 80.

On répétera ces procédures pour analyser le contenu de chacun des chapitres.

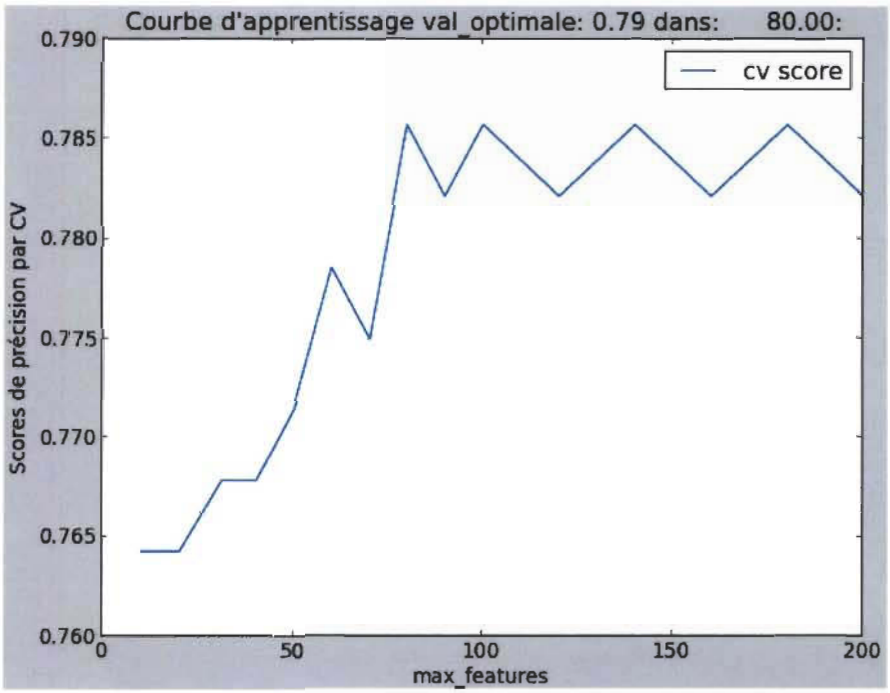


FIGURE 8.6 – Courbe d'apprentissage de la forêt de décision

8.5 La classification et analyse des chapitres.

On analysera séparément le résultat de la classification de chaque chapitre. Vu que notre hypothèse établit que notre procédure permettra de caractériser le contenu de chaque chapitre, il s'avère utile d'avoir l'index du livre comme référence :

- Chapitre I. Les Arabes nomades et Arabes sédentaires des campagnes.
- Chapitre II. Les Arabes des villes. - Mœurs et coutumes.
- Chapitre III. Institutions politiques et sociales des Arabes.
- Chapitre IV. Les femmes en Orient.
- Chapitre V. Religion et morale.

L'analyse de chaque chapitre sera faite à partir des résultats de l'optimisation obtenue pour chaque classe, avec chaque classeur, par la procédure décrite dans la section précédente.

8.5.1 Chapitre I. Les Arabes nomades et Arabes sédentaires des campagnes.

Après le processus d'optimisation, dont les courbes d'apprentissage sont dans la section de l'annexe A.2, on a les rapports de classification suivants :

Pour la forêt de décision :

Gini Index	precision	recall	f1−score	support
0	0.82	0.98	0.90	96
1	0.71	0.20	0.31	25
avg / total	0.80	0.82	0.77	121
Entropie	precision	recall	f1−score	support
0	0.82	0.98	0.90	96
1	0.71	0.20	0.31	25
avg / total	0.80	0.82	0.77	121

Pour la SVM :

	precision	recall	f1−score	support
0	0.85	0.98	0.91	96
1	0.82	0.36	0.50	25
avg / total	0.85	0.85	0.83	121

Dans cette classe, la précision obtenue avec l'ensemble de données de test, par la forêt de décision est, tant par le critère d'entropie que par le gini index, de 80% tandis que la précision par la SVM est de 85%. Nous utilisons maintenant la fonction servant à faire les graphiques des mots les plus importants pour la classification des segments comme appartenant ou non à la classe objectif. Les voici :

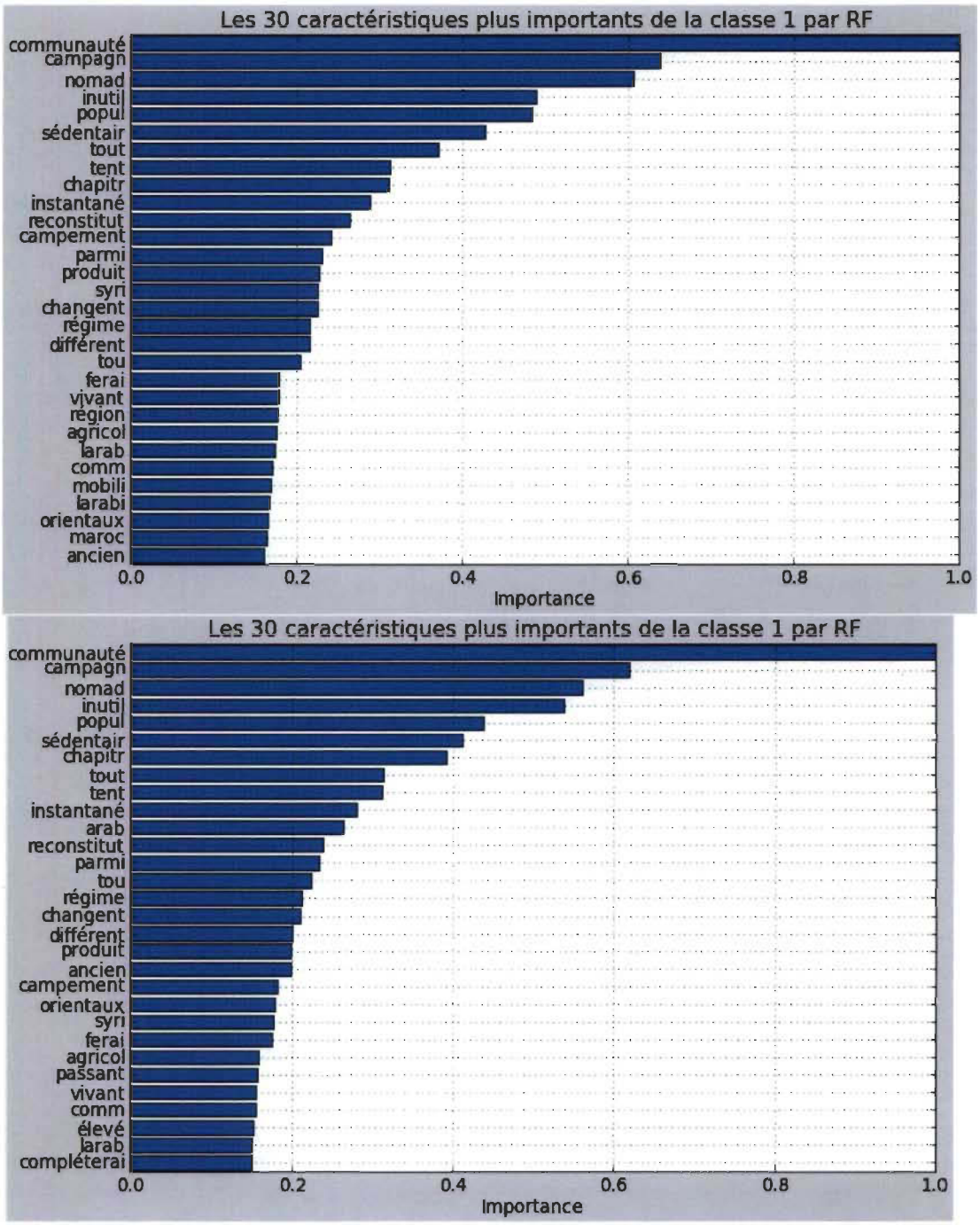


FIGURE 8.7 – Les 30 mots plus importants de la classe 1 par RF avec la mesure d'entropie (en bas) et avec le gini index (en haut).

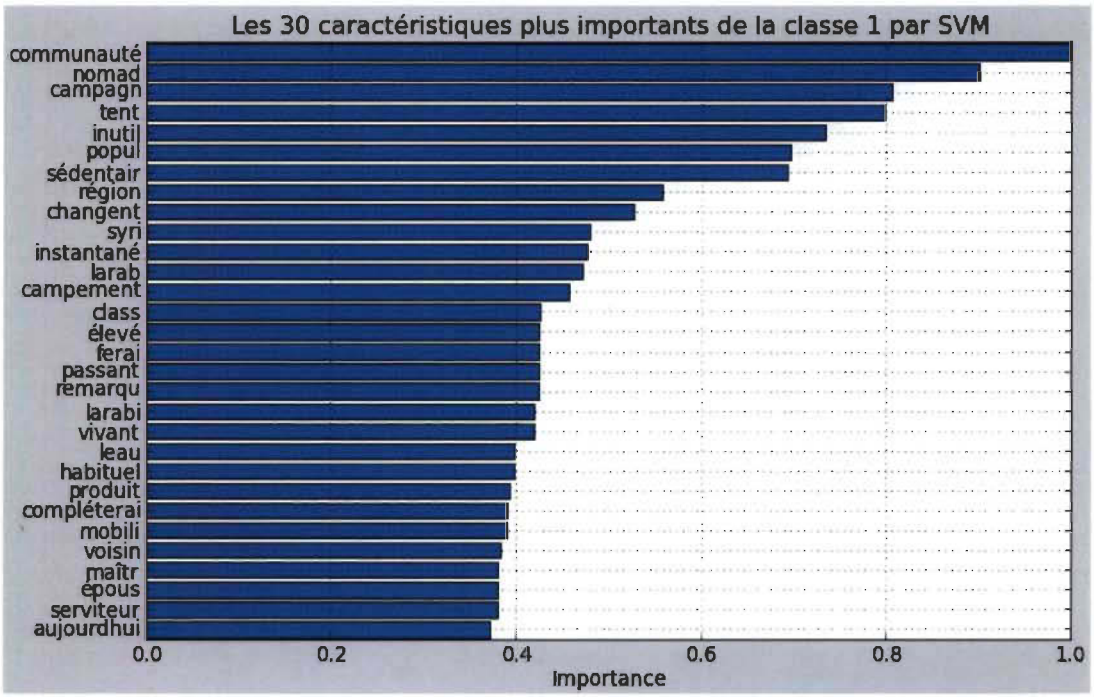


FIGURE 8.8 – Les 30 mots plus importants de la classe 1 par la SVM

On doit rappeler que nous avons lemmatisé les mots en les réduisant à la tige (stemming), donc, en observant les graphiques, nous découvrons que les termes [1]qui contribuent davantage à la classification, tant par des forêts de décision que par la SVM sont : communauté, nomad, campagn, popul, sedentair. La liaison de ces mots avec le chapitre I du livre relatif aux sociétés arabes nomades et de campagne est clairement visible.

D'une part, outre les mots que nous avons mentionnés, on observe qu'il y a d'autres mots communs dans les trois représentations graphiques. Puisque la forêt de décision choisit les mots les plus importants pour la classification dans certaines des classes, mais sans permettre de savoir dans quelle classe ce mot a une forte influence pour la classification, il est possible qu'apparaissent des mots avec une forte influence pour classer dans la classe cible qui soient mélangés avec des mots qui

1. Des termes tronqués correspondant à la lemmatisation des termes originaux qu'on appellera quand même mots ici par facilité pour l'exposé

permettent de classer dans le reste des classes. D'autre part, les mots présentés par la SVM sont des vecteurs trouvés du même côté de l'hyperplan de décision construit et, par conséquent, ils ont tous une forte influence pour classer dans la classe cible et il n'y a pas de mots aidant à classer dans une autre classe comme il arrive avec les forêts de décision. En conséquence, nous obtenons donc un lot de mots dans l'intersection des ensembles de mots obtenus, c'est-à-dire, des mots appartenant aux trois ensembles trouvés. Ces mots de l'intersection sont :

communauté, campagn, nomad, inutil, popul, sédentair, tent, instantané, changent, produit, campement, syri, ferai, vivant, larab.

En utilisant ce dernier résultat et en regardant encore les graphiques des arbres de décision, on peut penser qu'il est possible que des mots tels que : tout, chapitr et orientaux, qui ne sont pas dans l'intersection avec l'ensemble de mots de la SVM, ont une forte influence pour la classification dans certaines des autres classes, c'est-à-dire pour classer dans certains des autres chapitres.

Il est possible finalement de choisir un sous-ensemble de l'ensemble de mots qu'on vient de trouver pour explorer, à l'aide de la fonction dans le menu « importance » > « segments » du logiciel, les segments les plus proches à la liste de mots choisis. On utilisera les mots : communauté, campagn, nomad, popul, sédentair, qui à notre avis, sont ceux qui caractériseraient le mieux le chapitre, en accord avec le titre. Le résultat est montré dans la figure 8.9.

FIGURE 8.9 – Affichage des segments du chapitre 1.

8.5.2 Chapitre II. Les Arabes des villes. - Mœurs et coutumes.

Après le processus d'optimisation, dont les courbes d'apprentissage se trouvent dans la section A.3 des annexes, on a les rapports de classification suivants :

Pour la forêt de décision :

Gini Index	precision	recall	f1-score	support
0	0.80	0.99	0.88	92
2	0.86	0.21	0.33	29
avg / total	0.81	0.80	0.75	121
Entropie	precision	recall	f1-score	support
0	0.80	0.98	0.88	92
2	0.75	0.21	0.32	29
avg / total	0.79	0.79	0.75	121

Pour la SVM :

	precision	recall	f1-score	support
0	0.82	0.96	0.88	92
2	0.71	0.34	0.47	29
avg / total	0.80	0.81	0.78	121

Dans cette classe, la plus grande précision a été obtenue par la forêt de décision avec gini index, 81%, suivie par la SVM avec 80% et par la forêt de décision avec l'entropie avec 79%. Nous procédons à l'exploration des mots qui caractérisent le chapitre 2, en utilisant la fonction servant à faire les graphiques des mots les plus importants pour la classification des segments comme appartenant ou non à la classe cible. Observons les figures 8.10 et 8.11.

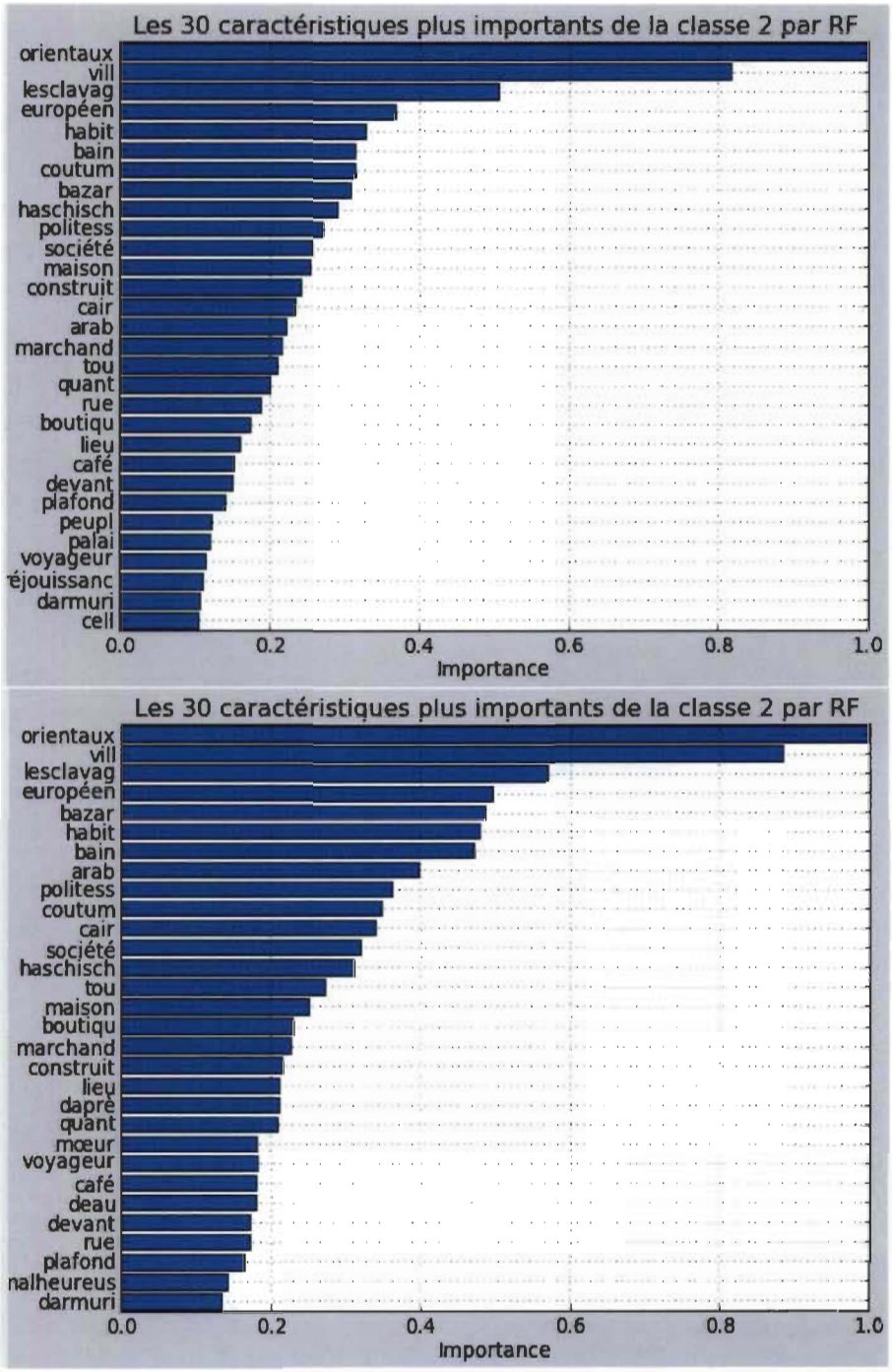


FIGURE 8.10 – Les 30 mots plus importants de la classe 2 par RF avec la mesure d'entropie (en bas) et avec le gini index (en haut).

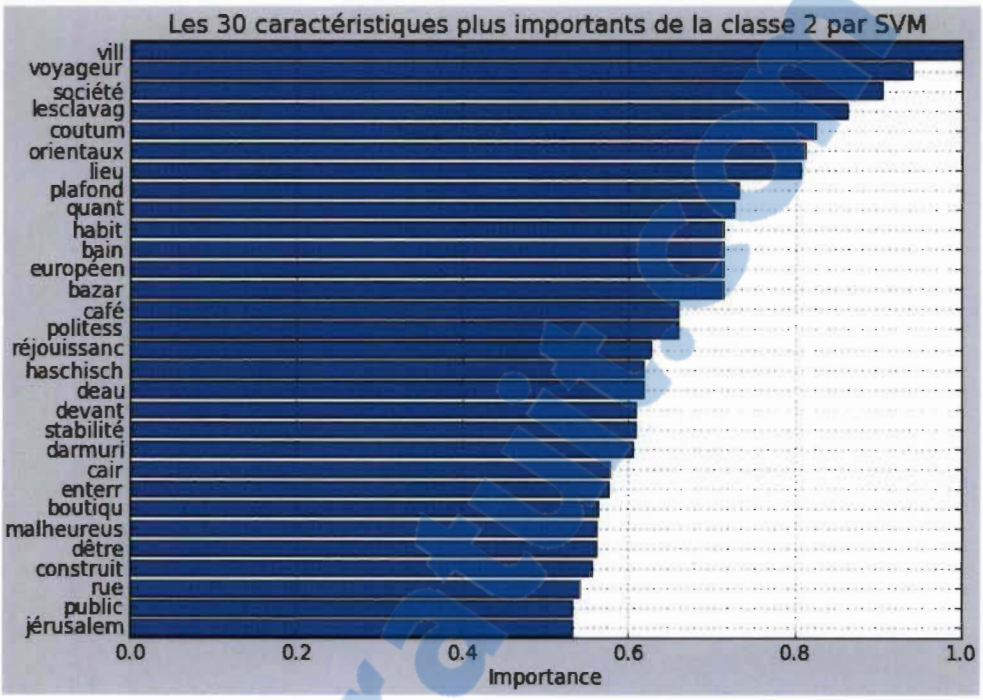


FIGURE 8.11 – Les 30 mots plus importants de la classe 2 par la SVM.

Avant d'analyser le résultat, nous répétons le processus, effectué dans la section précédente, pour choisir dans l'ensemble de mots plus importants présentés par les modèles de forêt de décision ceux qui apparaissent aussi comme les plus importants selon la SVM. L'ensemble trouvé est : orientaux, vill, lesclavag, européen, bazar, habit, bain, politess, coutum, cair, société, haschisch, boutiqu, construit, lieu, quant, voyageur, café, devant, rue, plafond.

En regardant le résultat trouvé, on peut penser que le chapitre traite des villes orientales, de leurs coutumes et activités, ainsi que certaines autres caractéristiques comme les bazars, les boutiques et les bains, ce qui correspond au titre du chapitre. Nous explorons maintenant les segments les plus proches aux cinq mots les plus importants trouvés : vill, voyageur, société, coutum, orientaux et lieu (figure A.5) de l'appendice.

8.5.3 Chapitre III. Institutions politiques et sociales des Arabes.

Après le processus d'optimisation, dont les courbes d'apprentissage se trouvent dans la section A.4 de l'appendice, on a les rapports de classification suivants :

Pour la forêt de décision :

Gini Index	precision	recall	f1-score	support
0	0.85	0.95	0.89	98
3	0.55	0.26	0.35	23
avg / total	0.79	0.82	0.79	121
Entropie	precision	recall	f1-score	support
0	0.85	0.96	0.90	98
3	0.64	0.30	0.41	23
avg / total	0.81	0.83	0.81	121

Pour la SVM :

	precision	recall	f1-score	support
0	0.90	0.88	0.89	98
3	0.52	0.57	0.54	23
avg / total	0.82	0.82	0.82	121

Dans cette classe, la plus grande précision de classification sur l'ensemble de test a été obtenue par la SVM, atteignant un niveau de précision de 82%, suivie par la forêt de décision en utilisant la mesure d'entropie 81% et, finalement la forêt de décision avec le gini index a obtenu 79%. Nous faisons à nouveau le graphe des mots plus importants pour la classification des segments de la classe selon les résultats trouvés. Voici les figures 8.12 et 8.13.

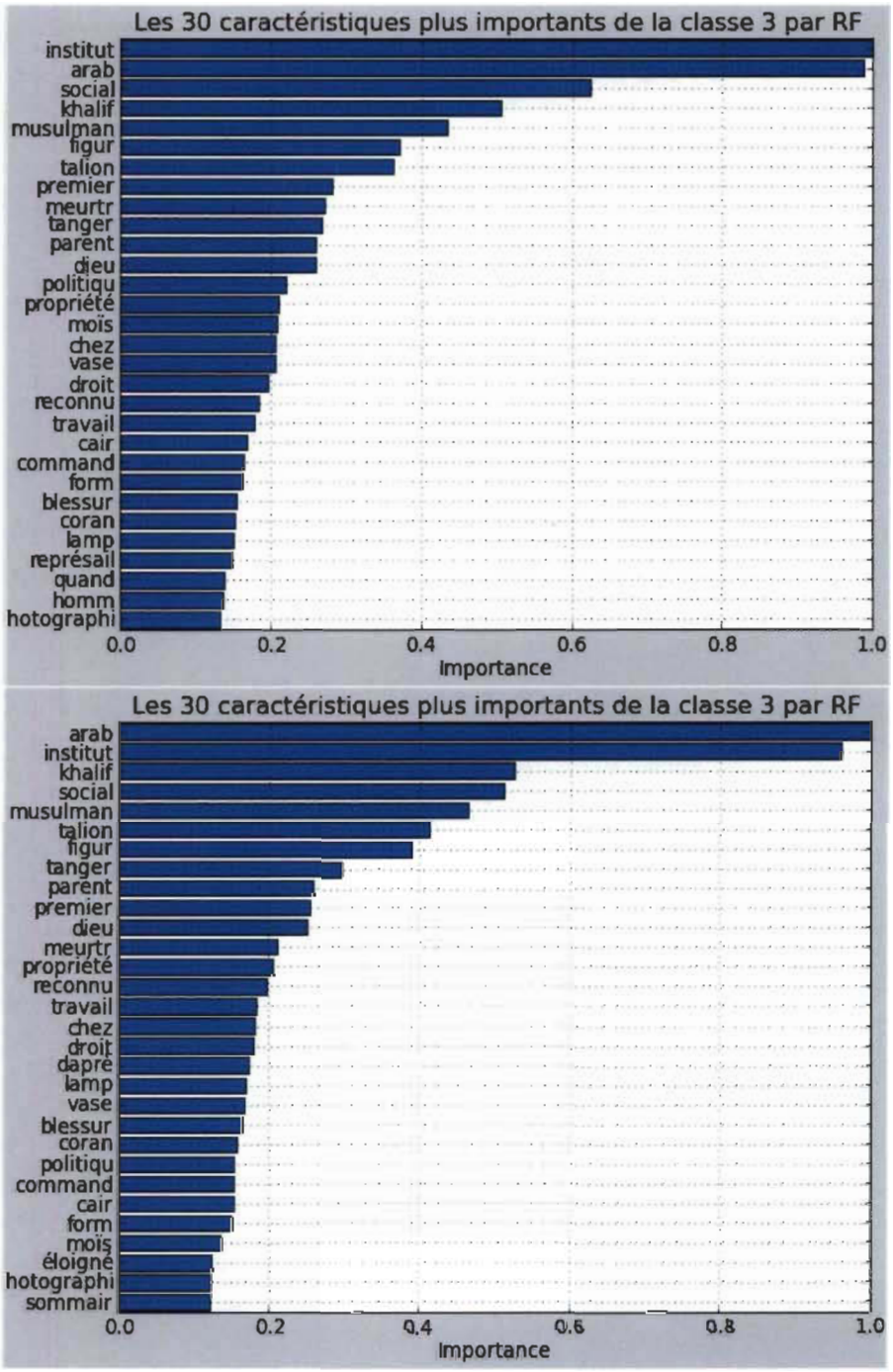


FIGURE 8.12 – Les 30 mots plus importants de la classe 3 par RF avec la mesure d'entropie (en bas) et avec le gini index (en haut).

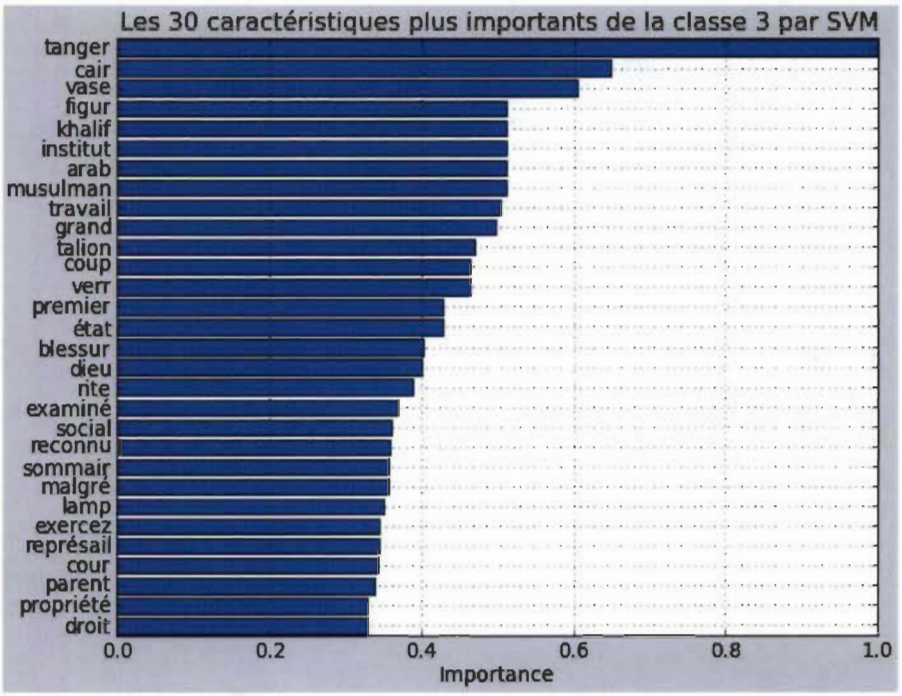


FIGURE 8.13 – Les 30 mots plus importants de la classe 3 par SVM

En répétant le processus de sélection des mots les plus importants trouvés par les modèles de forêt de décision et par la SVM, on obtient les dix-huit mots suivants : arab, institut, khalif, social, musulman, talion, figur, tanger, parent, premier, dieu, propriété, reconnu, travail, lamp, vase, blessur, cair.

Le résultat nous permet de découvrir que le chapitre traite des institutions sociales arabes musulmanes et des sujets relatifs à celles-ci comme le travail, la propriété, le talion, dieu et le khalife. Les segments les plus proches aux cinq mots plus importants trouvés sont : vill, voyageur, société, coutum, orientaux et lieu (figure A.5) de la section des annexes. Il est intéressant de noter que, bien que le titre du chapitre inclut le mot droit, celui-ci a été inclus par les forêts de décision mais non par la SVM, toutefois d'autres mots comme propriété, travail, dieu et talion ont été inclus par les deux modèles et pourraient caractériser cet aspect du discours du chapitre.

Le résultat de l'exploration des segments proches aux mots : khalif, institut, arab, musulman, social sont montrés dans l'annexe A.4.

8.5.4 Chapitre IV. Les femmes en Orient.

Après le processus d'optimisation, dont les courbes d'apprentissage se trouvent de la section des annexes A.5, on a les rapports de classification suivants :

Pour la forêt de décision :

Gini Index	precision	recall	f1-score	support
0	0.90	0.93	0.92	102
4	0.56	0.47	0.51	19
avg / total	0.85	0.86	0.85	121
Entropie	precision	recall	f1-score	support
0	0.90	0.92	0.91	102
4	0.53	0.47	0.50	19
avg / total	0.85	0.85	0.85	121

Pour la SVM :

	precision	recall	f1-score	support
0	0.89	0.91	0.90	102
4	0.44	0.37	0.40	19
avg / total	0.82	0.83	0.82	121

Cette fois, les modèles de forêt de décision ont obtenu, tous les deux, une précision de 85% en dépassant à la SVM qui a atteint 82%. Dans le cas des forêts de décision, il a été nécessaire que les modèles soient plus complexes que ceux utilisés dans tous les chapitres précédents puisque le nombre de caractéristiques pris, pour chaque nœud, dans le modèle optimal, a été avec le gini de 160 et avec l'entropie de 200.

En faisant le graphique des mots plus importants pour la classification des segments du présent chapitre, on trouve les résultats suivants, voir les figures 8.14 et

8.15.

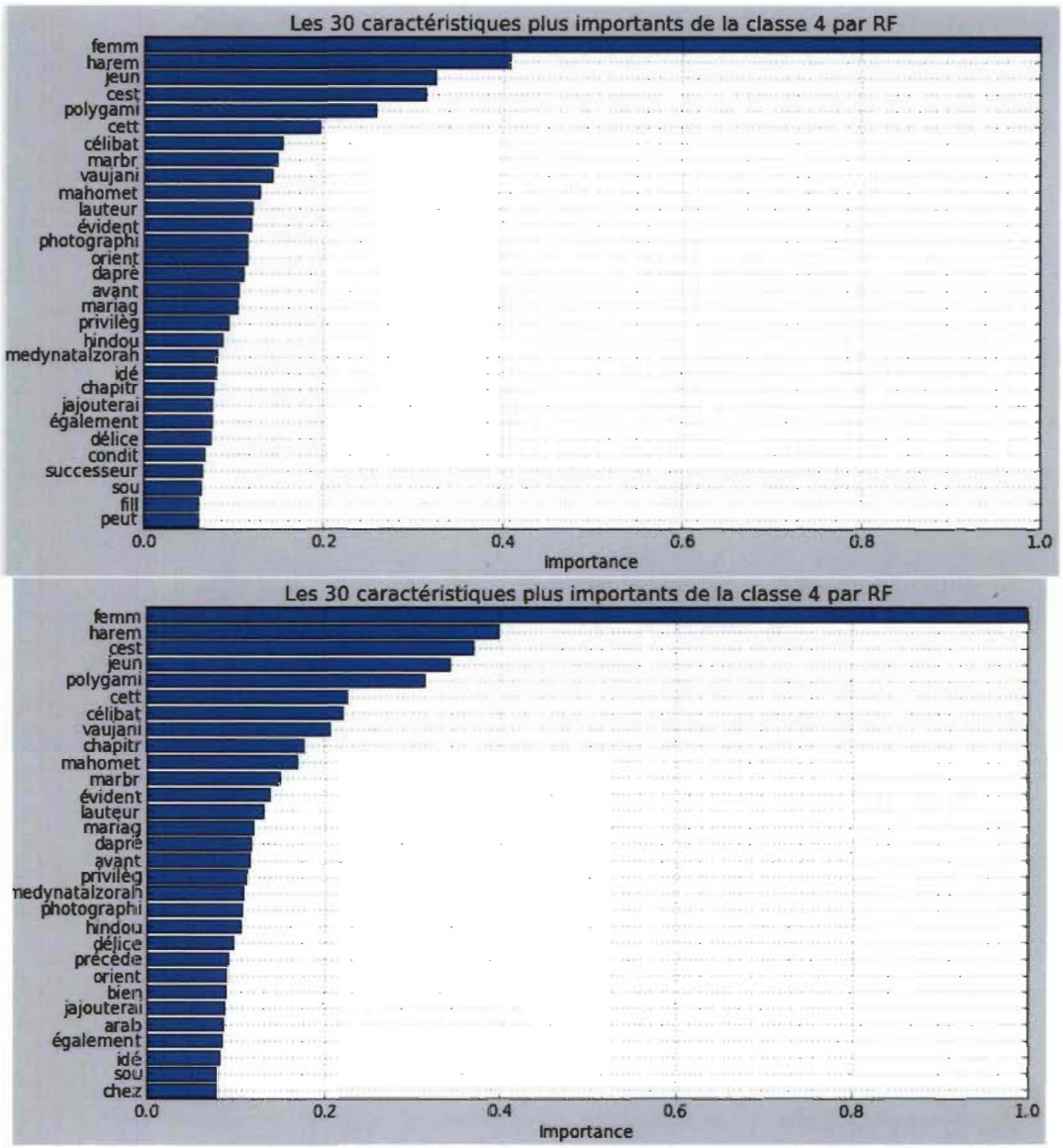


FIGURE 8.14 – Les 30 mots plus importants de la classe 4 par RF avec la mesure d'entropie (en bas) et avec le gini index (en haut).

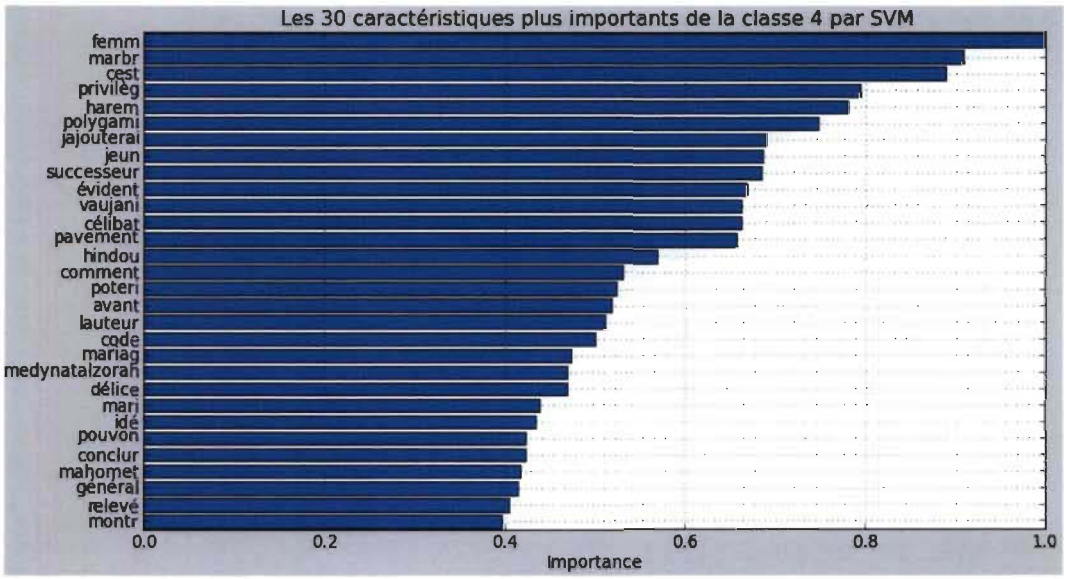


FIGURE 8.15 – Les 30 mots plus importants de la classe 4 par la SVM.

Après le processus de sélection des mots plus importants trouvés par les modèles de forêt de décision et par la SVM, on obtient les dix-neuf mots suivants : femm, harem, cest, jeun, polygami, célibat, vaujani, mahomet, marbr, évident, lauteur, mariag, avant, privilèg, medynatalzorah, hindou, délice, jajouterai, idé.

En étudiant les résultats, tant l'ensemble de mots que les graphiques, l'identification des sujets traités dans ce chapitre, excepté celui de la femme, s'avère plus difficile que pour les chapitres précédents. Il est facile de voir que le sujet principal est la femme (le terme principal dans les trois graphiques) mais l'identification d'un autre sujet important s'avère délicat, bien qu'on trouve quelques mots en rapport avec la condition de et autour de la femme, comme jeun, harem, polygami, célibat, mariag, mari et fil. Toutefois, le reste des mots trouvés ne fournit pas beaucoup d'information claire à propos du contenu du chapitre. Il est possible que le chapitre ait un vocabulaire plus hétérogène et qu'il ne contienne pas beaucoup de mots fortement liés à la classe qui puissent aider à la classification des segments, en rendant plus difficile la classification de ceux-ci. Nous explorons finalement les segments du chapitre proches aux termes : femm harem jeun polygami célibat mariag mari (figure

A.9 de la section des annexes).

8.5.5 Chapitre V. Religion et morale.

On répète une dernière fois le processus d'optimisation afin de trouver les modèles optimaux, dont les courbes d'apprentissage se trouvent dans la section A.6 de l'appendice, en parvenant aux résultats suivants :

Pour la forêt de décision :

Gini Index	precision	recall	f1-score	support
0	0.88	1.00	0.94	96
5	1.00	0.48	0.65	25
avg / total	0.91	0.89	0.88	121
Entropie	**precision**	**recall**	**f1-score**	**support**
0	0.88	1.00	0.94	96
5	1.00	0.48	0.65	25
avg / total	0.91	0.89	0.88	121

Pour la SVM :

	precision	recall	f1-score	support
0	0.89	0.98	0.93	96
5	0.87	0.52	0.65	25
avg / total	0.88	0.88	0.87	121

Bien que les rapports de classification des modèles de forêt de décision soient identiques, le modèle optimal avec gini index utilise 180 caractéristiques dans la construction de nœuds tandis que la forêt avec entropie en utilise 240. Il est important de remarquer que, étant donné qu'une plus grande complexité des modèles de forêt de décision a été nécessaire, particulièrement par le gini index, la recherche a été

étendue en ajoutant à la liste des hyperparamètres à tester, mentionnés dans la section 8.4.2, les valeurs suivantes : 220, 240, 260, 280, 300. Le résultat est montré dans la figure A.6.

Alors qu'il a été nécessaire d'élever la complexité des modèles de forêt de décision, la précision atteinte en classant les segments de cette catégorie a été la meilleure de tout le livre. La meilleure précision obtenue est de 91% par les deux modèles de forêt de décision tandis que la SVM a pu classer correctement 88% des données de test.

En faisant le graphique des mots plus importants pour la classification des segments du présent chapitre, on obtient les résultats suivants qui sont montrés dans les figures 8.17 et 8.16.

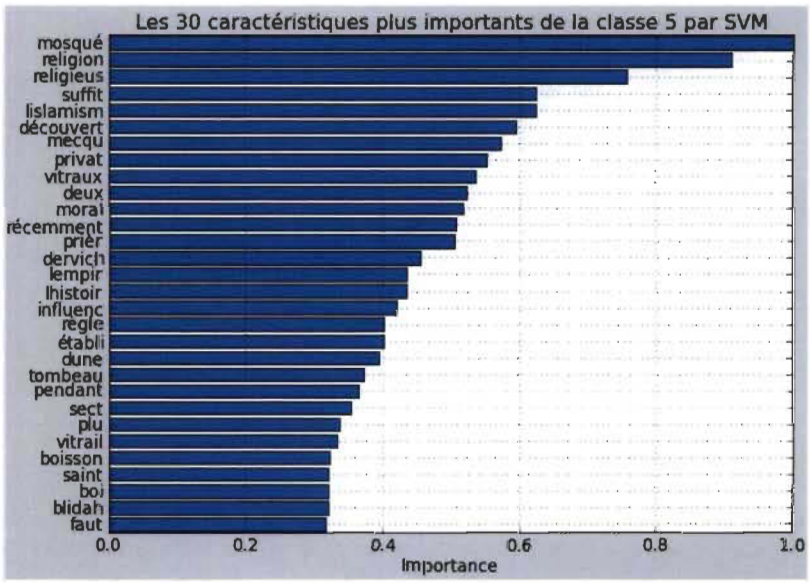


FIGURE 8.16 – Les 30 mots plus importants de la classe 5 par la SVM

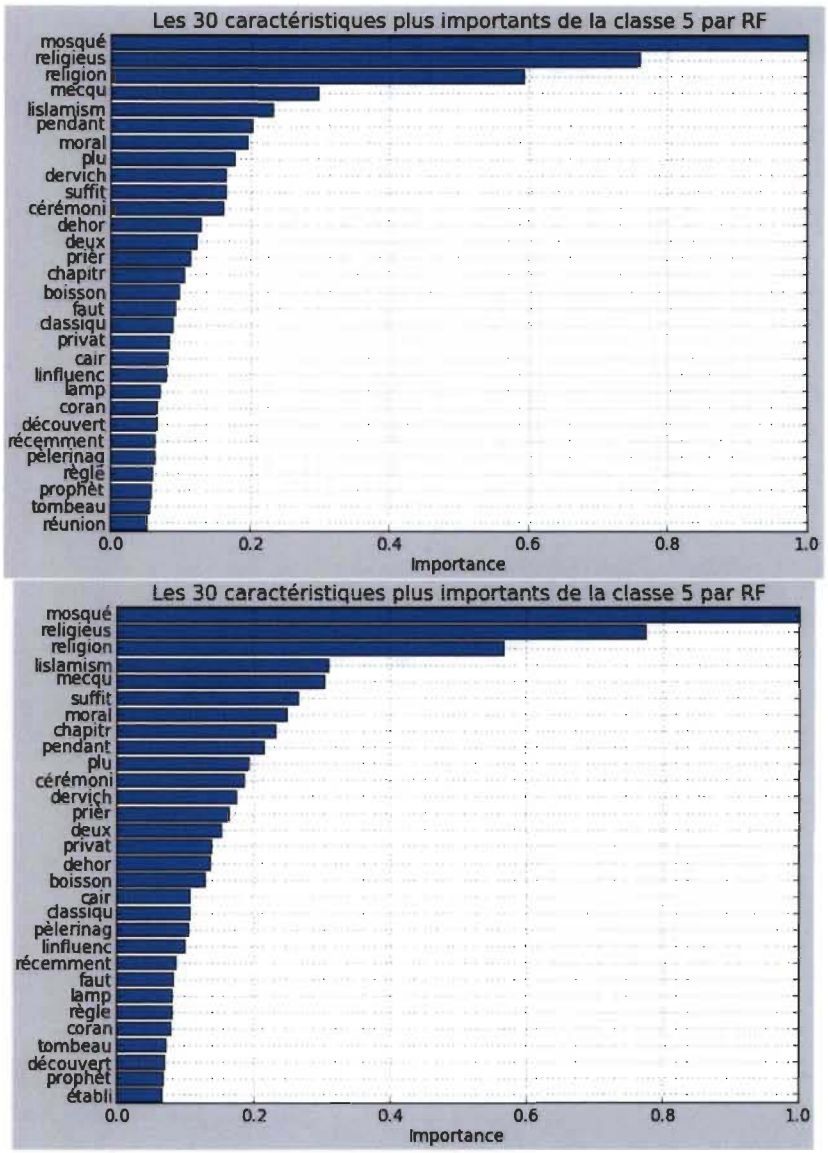


FIGURE 8.17 – Les 30 mots plus importants de la classe 5 par RF avec la mesure d'entropie (en bas) et avec le gini index (en haut).

Après le processus de sélection des mots plus importants trouvés par les modèles de forêt de décision et par la SVM, on obtient les dix-huit termes suivants : mosqué, religieus, religion, lislamism, mecqu, suffit, moral, pendant, plu, dervich, prièr, deux, privat, boisson, récemment, règle, tombeau, découvert.

À la différence du chapitre précédent, dans ce chapitre l'ensemble de mots trouvés par tous les modèles nous donnent une idée claire sur son contenu en nous permettant

de découvrir qu'il est fortement en rapport avec la religion et avec la morale.

On montre, finalement, l'exploration des segments proches aux termes : mosqué religion religieus lislamism mecqu moral prièr (figure A.11 de la section des annexes).

8.6 Résumé

Dans ce chapitre, nous avons effectué l'analyse d'un corpus de texte en utilisant les deux modèles implémentés. À cet effet, nous avons effectué les opérations suivantes : nous avons d'abord fait le nettoyage, le pré-traitement et la vectorisation du texte ; il a été segmenté dans des paragraphes étiquetés selon le chapitre du livre auquel ils appartiennent. Nous avons ensuite effectué une classification des segments appartenant à chacun des chapitres pour identifier les mots qui sont les plus significatifs pour la classification tant par la SVM que par la forêt de décision. Étant donné que l'ensemble de termes les plus significatifs rendus par l'un et l'autre modèle sont différents, nous avons utilisé l'intersection des deux ensembles comme l'ensemble de termes qui nous permettent la meilleure caractérisation de chaque chapitre. Finalement, en utilisant certains des termes les plus significatifs, nous avons exploré les segments de texte qui les contiennent. Les résultats de notre analyse permettent, dans tous les cas, d'identifier de manière automatique et rapide le contenu de chaque chapitre par l'identification des termes les plus significatifs pour la classification, ainsi on peut explorer les segments qui les contiennent. Nos expériences nous ont permis, par ailleurs, de tester les fonctionnalités mises en œuvre pour l'optimisation des modèles.

Chapitre 9

Conclusion

Dans le cadre de notre recherche, nous nous sommes intéressés au développement d'un outil qui nous permettrait d'explorer le contenu des documents ou des segments de texte en utilisant une stratégie de classification de ces derniers au moyen de deux des techniques d'apprentissage automatique d'appoint : les forêts de décision et les machines à vecteurs de support (SVM). La nature de ces modèles d'apprentissage automatique nous a permis d'identifier les caractéristiques ou les mots les plus importants pour la classification qui, d'une certaine manière, sont ceux qui caractérisent le mieux cette classe, en nous permettant d'identifier les sujets principaux des catégories explorées par l'intersection de l'ensemble de mots rendus par les deux modèles.

On a développé un logiciel qui permet de faire cette caractérisation en nous basant sur une stratégie de classification des segments étiquetés. On exécute une classification d'une classe contre toutes les autres classes pour identifier les mots les plus importants pour la classification des segments de la classe cible. Le résultat de la classification est un rapport de classification ainsi qu'un résumé graphique des caractéristiques les plus importantes en permettant l'exploration des segments qui les contiennent. Le logiciel offre également à l'utilisateur, en variant un seul paramètre

dans le processus d'optimisation, un outil pour la visualisation de la courbe d'apprentissage des algorithmes disponibles, en lui autorisant la meilleure optimisation possible de ceux-ci, ce qui lui permettra d'obtenir les meilleurs résultats de classification et, par conséquent, la meilleure caractérisation possible de la catégorie cible. Enfin, le programme permet d'explorer les segments de texte contenant les mots plus importants trouvés ou des sous-ensembles de ceux-ci.

L'outil a été testé en utilisant comme corpus de test le livre Gustave Le Bon (1884) La civilisation des arabes, livre quatrième, Mœurs et institutions[49], dont le contenu a été segmenté en paragraphes qui ont ensuite été étiquetés selon le chapitre auquel ils appartiennent pour pouvoir être utilisés par les algorithmes d'apprentissage. Bien qu'il s'agisse d'un petit corpus de texte, les résultats obtenus nous donnent une grande satisfaction puisque, dans tous les cas, les mots les plus importants trouvés, en plus de faire sens avec le titre du chapitre correspondant, nous ont permis de connaître, de manière rapide, les sujets centraux ainsi que le vocabulaire qui caractérisent le discours de chaque chapitre.

Notre ajout de fonctionnalité au logiciel REGASS permet de :

- Disposer des nouveaux outils dans l'ensemble des modèle disponibles.
- Utiliser un approche différente d'explorer et caractériser le contenu du texte au moyen des deux modèles mis en œuvre.
- Analyser le contenu des conglomérats générés par les algorithmes non supervisés disponibles.

De nombreuses perspectives s'offrent à la suite de notre travail, à savoir :

- L'exploration de contenus dans de grandes quantités de texte, par exemple dans le web. Puisque les techniques utilisées sont très efficaces, spécialement les forêts de décision, celles-ci pourraient être utilisées pour analyser de grandes quantités de texte, pourvu qu'il soit possible de connaître les étiquettes de classe des textes pour pouvoir optimiser les modèles.

— L'utilisation d'une méthode de sélection de variables (feature selection) dont le résultat réduit le nombre de mots significatifs pour la classification.

Bibliographie

[1] Yaser Abu-Mostaf. "Learning from data machine learning course - recorded at a live broadcast from caltech". Web, Jan 2013. SVM Lecture. Acceded May 2013.

[2] Edgar Anderson. The species problem in iris. *Annals of the Missouri Botanical Garden*, (IrisAnder1936) :457–509, 1936.

[3] Gary Belsky. "Why text mining may be the next big thing". *http ://time.com/*, march 2012. Web. Accesed 02 Dic 2014.

[4] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Image classification using random forests and ferns. In *2007 IEEE 11th International Conference on Computer Vision*, number ISSN : 1550-5499, pages 1–8. IEEE, 14-21 Oct. 2007.

[5] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[6] Leo Breiman. Random forests. [6], pages 1–35. University of California Statistics Department.

[7] Jaime Carbonell G, Tom Mitchel M., and Ryszard Michalsky S. Machine learning : A historical and methodological analysis. *AI Magazine*, 4(3) :69–79, 1983.

[8] Yin-Wen Chang. Feature ranking using linear svm. *Causation and Prediction Challenge Challenges in Machine Learning*, 2 :47, 2008.

[9] Paul R. Cohen. *Empirical Methods for Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1995.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3) :273–297, 1995.

[11] J. & Konukoglu E Criminisi, A. & Shotton. Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. *Microsoft Research technical report TR-2011-114*, page 151, 2011. Microsoft manual on random forest.

[12] Nello Cristianini and John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods.* Cambridge university press, 2000.

[13] Durga Bhavani Dasari. Text categorization and machine learning methods : Current state of the art. *Global Journal of Computer Science and Technology*, 12(11-C), 2012.

[14] Pubmed data base. U.s. national library of medicine. Web site, May 2014. http ://www.nlm.nih.gov/.

[15] Nando de Freitas. "Undergraduate machine learning 17 : Linear prediction". Web video lecture., November 2012.

[16] Nando de Freitas. "Machine learning - logistic regression". Web video lecture., March 2013. University of British Columbia.

[17] Steve Descôteaux. Les règles d'association maximale au service de línterprétation des résultats de classification. Master's thesis, Université du Québec à Trois Rivières, April 2014.

[18] L Devroye, L Györfi, and G Lugosi. *A probabilistic theory of pattern recognition.* Number ISBN 0-387-94618-7. Springer-Verlag, Berlin, 1996.

[19] Thomas G. Dietterich. Ensemble learning. In *The handbook of brain theory and neural network* [19], pages 110–125. MIT Press : Cambridge, MA 2002.

[20] Peter Sheridan Dodds, Kameron Decker Harris, Isabel M Kloumann, Catherine A Bliss, and Christopher M Danforth. Temporal patterns of happiness and information in a global social network : Hedonometrics and twitter. *PloS one*, 6(12) :e26752, December 2011.

[21] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3) :103–130, 1997.

[22] Xin Dong and Wu Zhaohui. Speaker recognition using continuous density support vector machines. *Electronics letters*, 37(17) :1099–1101, 2001.

[23] Richard O Duda, Peter E Hart, et al. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973.

[24] Chantal Enguehard. "Mots fonctionnels", 2015. Web. Consulted on 2015-04-03.

[25] Ronen Feldman and James Sanger. *The text mining handbook : advanced approaches in analyzing unstructured data.* Cambridge University Press, 2007.

[26] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2) :179–188, 1936.

[27] Carlos Gershenson. Artificial neural networks for beginners. In *arXiv preprint cs/0308031* [27].

[28] Guodong Guo, Stan Z Li, and Kap Luk Chan. Face recognition by support vector machines. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 196–201. IEEE, 2000.

[29] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining : concepts and techniques.* Morgan kaufmann, 2006.

[30] LIU Han, LIU Ding, Gang ZHENG, LIANG Yanming, and SONG Nianlong. Natural gas load forecasting based on least squares support vector machine [j]. *Journal of Chemical Industry and Engineering (China)*, 5 :026, 2004.

[31] Robert & Friedman Jerome Hastie, Trevor & Tibsharani. *The elements of statistical learning.* In Springier [31], 2nd edition, January 2013.

[32] Robert Hastie, Trevor & Tibshirani. Stanford statistical learning online course lecture "Tree based methods". Web Video, Nov 2013. Accesed 15 Sep 2014.

[33] Robert Hastie, Trevor & Tibshirani. Stanford statistical learning online course SVM lecture. "Example and comparison with logistic regression". Web Video, Nov 2013. Accesed 15 Sep 2014.

[34] Andreas Hotho, Andreas Nürnberger, and Gerhard Paaß. A brief survey of text mining. In *Ldv Forum*, volume 20, pages 19–62, 2005.

[35] http ://scikit learn.org. Kernel functions. Web site, September 2014.

[36] Junling Hu. History of machine learning. Web site, april 2013. http ://www.aboutdm.com/2013/04/history-of-machine-learning.html.

[37] Wei Huang, Yoshiteru Nakamori, and Shou-Yang Wang. Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, 32(10) :2513–2522, 2005.

[38] Klaus Jaffe. *Qué es la ciencia ? Una visión evolutiva.* Empresas Polar, 2007.

[39] Anil K Jain and B Chandrasekaran. 39 dimensionality and sample size considerations in pattern recognition practice. *Handbook of statistics*, 2 :835–855, 1982.

[40] Anil K Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition : A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1) :4–37, January 2000.

[41] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning.* In *Springer* [41], 2013.

[42] Han Jiawei, Micheline Kamber, and Jian Pei. *Data Mining : Concepts and Techniques.* In [42], 2011.

[43] Thorsten Joachims. *Text categorization with support vector machines : Learning with many relevant features.* Springer, 1998.

[44] Sci kit Learn. Examples 3.7. validation curves : plotting scores to evaluate models. Web page. Consulted 2014-12-26. http ://scikit-learn.org/stable/modules/learning_curve.html.

[45] Raymond M Klein. The hebb legacy. In *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale* [45], page 1.

[46] Kevin Knight. Mining online text. *Communications of the ACM*, 42(11) :58–61, 1999.

[47] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9) :1464–1480, 1990.

[48] Pat Langley. The changing science of machine learning. In *Machine Learning* [48], pages 275–279. Springer, 2011.

[49] Gustave Le Bon. *La civilisation des Arabes.* Firmin-Didot et cie, 1884.

[50] Jimmy Lin and Chris Dyer. Data-intensive text processing with mapreduce. In *Synthesis Lectures on Human Language Technologies* [50], pages 1–177. 2010.

[51] D. A. McAllester. What is the most pressing issue facing ai and the aaai today ?. Candidate statement, election for Councilor of the American Association for Artificial Intelligence., 1998.

[52] Bjoern H Menze, B Michael Kelm, Daniel N Splitthoff, Ullrich Koethe, and Fred A Hamprecht. On oblique random forests. In *Machine Learning and Knowledge Discovery in Databases* [52], pages 453–469. 2011.

[53] Melanie Mitchell. *An introduction to genetic algorithms*, volume Mitchell1998. MIT press, 1998.

[54] Dan Munteanu. A quick survey of text categorization algorithms. *Annals of University\ Dunarea de Jos" of Galati*, 1, 2007.

[55] Sreerama K Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. In *Journal of Artificial Intelligence Research* [55], pages 1–32. 1994.

[56] Andrew Ng. Stanford machine learning course lecture : "Large Margin Intuition". Web video, 2011. Consulted : april 2014.

[57] Andrew Ng. Stanford machine learning course lecture : "Support Vector Machines Optimization Objective". Web video, 2011. Consulted : may 2014.

[58] Andrew Ng. Stanford machine learning course lecture : "SupportVectorMachines UsingAnSVM- Machine Learning". Web video, 2011. Consulted : april 2014.

[59] Andrew Ng. Stanford machine learning course lectures : "Kernels I" and "Kernels II". Web video, 2011. Consulted : may 2014.

[60] Andrew Ng. Stanford machine learning course lecture :"Regularization and Bias Variance". Web video, 2011. Consulted : may 2014.

[61] Jeffrey Ng and Shaogang Gong. Composite support vector machines for detection of faces across views and pose estimation. *Image and Vision Computing*, 20(5) :359–368, Elsevier 2002.

[62] Hiroshi Ogura, Hiromi Amano, and Masato Kondo. Distinctive characteristics of a metric using deviations from poisson for feature selection. *Expert Systems with Applications*, 37(3) :2273–2281, 2010.

[63] L Oliveira and Robert Sabourin. Support vector machines for handwritten numerical string recognition. In *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, pages 39–44. IEEE, 2004.

[64] Ajoy K Palit and Dobrivoje Popovic. Computational intelligence in time series forecasting : Theory and engineering applications (advances in industrial control). [64], pages 80–142.

[65] M. T. Pazienza. *Information extraction. Lecture notes in Computer Science.*, volume 1299. Springer, Heidelberg, Germany, 1997.

[66] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn : Machine learning in python. *The Journal of Machine Learning Research*, 12 :2825–2830, 2011. http ://scikit-learn.org.

[67] Rusell Stuart & Norving Peter. *Artificial intelligence a modern aproach*. Prentice hall Englewood Cliffs, 3rd edition, 2003.

[68] Philipe Proux. *Fouille de données Notes de cours.* Université de Lille 3, May 2011.

[69] Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2) :257–286, 1989.

[70] Danny Roobaert and Marc M Van Hulle. View-based 3d object recognition with support vector machines. In *Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop.*, pages 77–84. IEEE, 1999.

[71] Stuart Russel J and Peter Norving. *Artificial intelligence, a modern approach spanish Ed.s.* Prentice Hall, 2 edition, 2004.

[72] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11) :613–620, 1975.

[73] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1) :1–47, 2002.

[74] Kyung-Shik Shin, Taik Soo Lee, and Hyun-jung Kim. An application of support vector machines in bankruptcy prediction model. *Expert Systems with Applications*, 28(1) :127–135, 2005.

[75] S. Joshi Shrijit and Vir V. Phoha. Investigating hidden markov models capabilities in anomaly detection. *Computer Science, Louisiana Tech University*, 1 :98–103, 2006.

[76] Dimitri Siganos and Christos Stergiou. Neural networks. Web site, 2014 Nov. Consulted November 2014.

[77] Param Deep Singh and Jitendra Raghuvanshi. Rising of text mining technique : As unforeseen-part of data mining. *International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)*, 1(3) :pp–139, 2012.

[78] Amlan et al Srivastava, Abhinav & Kundu. Credit card fraud detection using hidden markov model. *IEEE Transactions on dependable and secure computing*, 5 :37- 48, January 2008.

[79] Don R Swanson and Neil R Smalheiser. Implicit text linkages between medline records : Using arrowsmith as an aid to scientific discovery. *Library trends*, 48(1) :48–59, 1999.

[80] PACE University. "Rosenblatt's Contributions". Web site, 2011. Consulted Feb 2014.

[81] Bülent Üstün. "Support vector machines". Radboud University Nijmegen. Web site. Consulted on 2014 Dic 15.

[82] Vincent Wan and William M Campbell. Support vector machines for speaker verification and identification. In *NEURAL NETWORKS SIGNAL PROCESS PROC IEEE*, volume 2, pages 775–784. Citeseer, 2000.

[83] wxPython. http ://www.wxpython.org/. Web site, 2014. Consulted on december 2014.

[84] wxWidgets. https ://www.wxwidgets.org/, 2015. Consulted on january 2015.

[85] Wen Xie, Lean Yu, Shanying Xu, and Shouyang Wang. A new method for crude oil price forecasting based on support vector machines. In *Computational Science–ICCS 2006*, pages 444–451. Springer, 2006.

Annexe A

Détail de l'optimisation.

Dans cette section se trouvent les résultats détaillés d'optimisation qui n'ont pas été inclus dans le chapitre 8.

A.1 Courbes d'apprentissage en variant le nombre d'arbres des forêts de décision.

Les courbes d'apprentissage mentionnées dans la section 8.4.2 sont les suivantes :

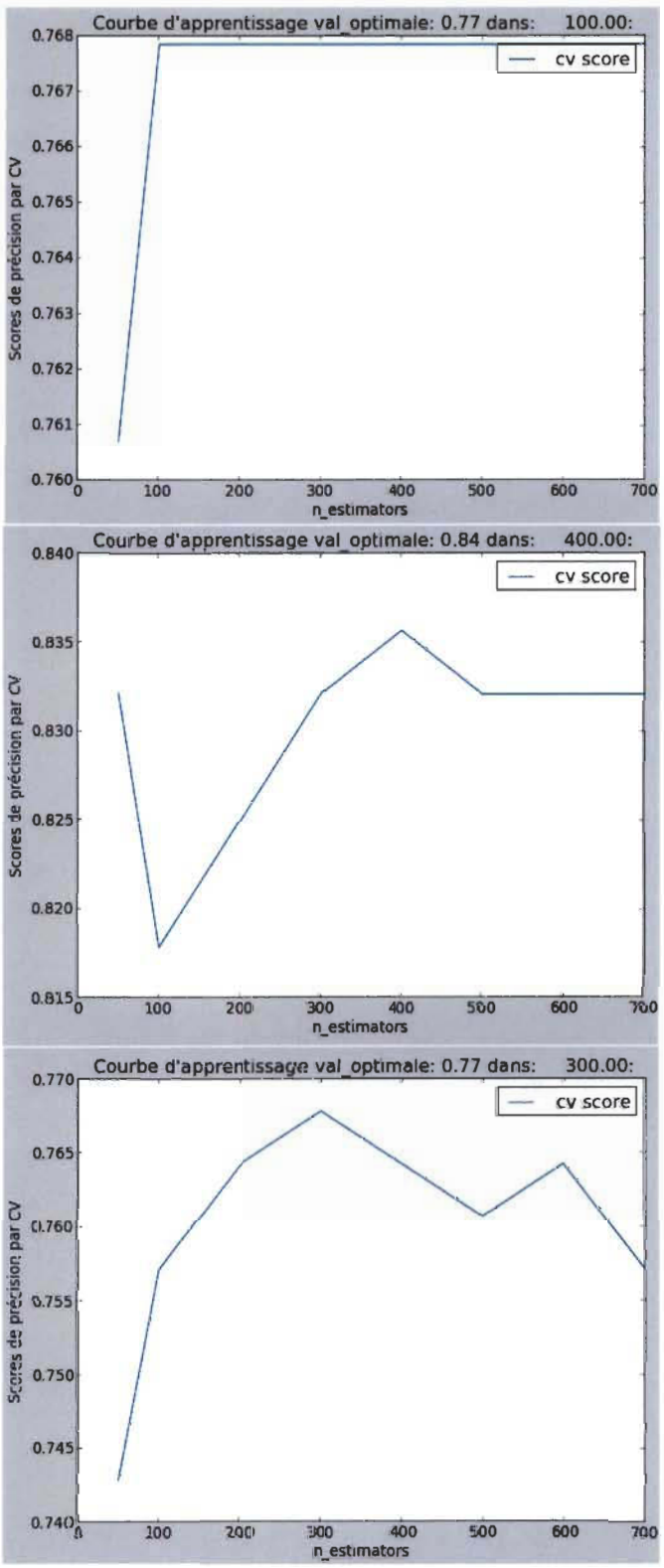


FIGURE A.1 – Courbe d'apprentissage des classes 1 (en haut), 2 (au milieu) et 3 (en bas). Nombre d'arbres de la forêt (n_estimators).

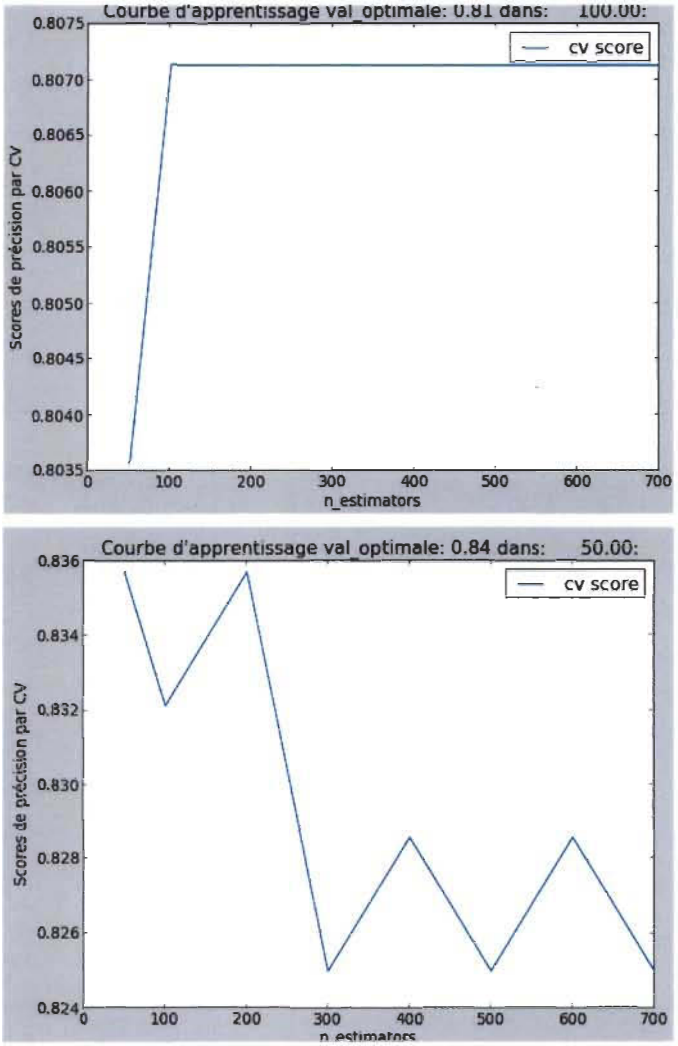


FIGURE A.2 – Courbe d'apprentissage des classes 4 (en haut) et 5 (en bas). Nombre d'arbres de la forêt (n_estimators).

A.2 Compléments d'analyse du chapitre 1.

A.2.1 Courbes d'apprentissage.

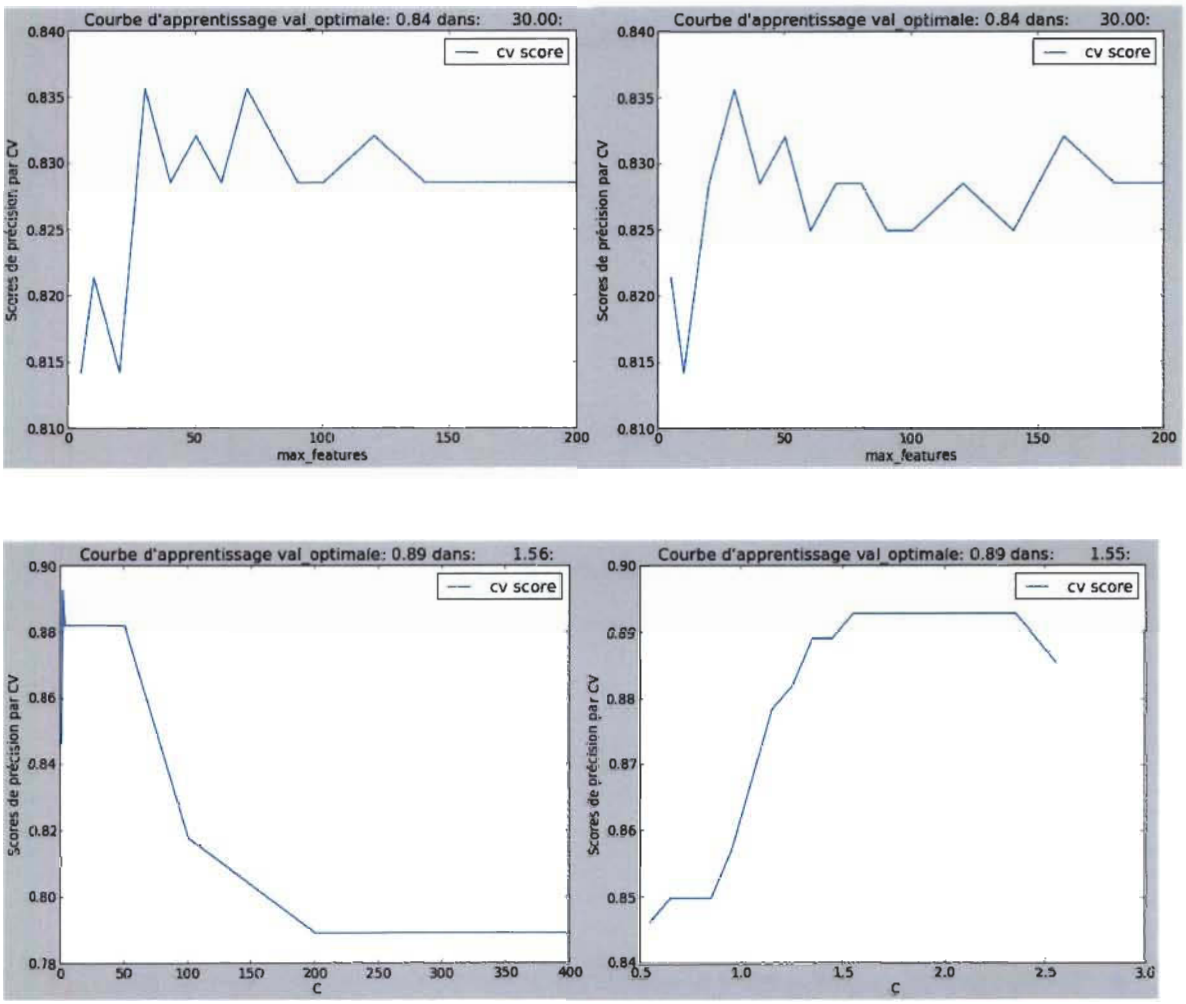


FIGURE A.3 – Courbes d'apprentissage du chapitre (classe) 1. En haut : à gauche RF gini, à droite RF entropie. En bas : à gauche SVM itération 1, à droite itération 2.

A.3 Compléments d'analyse du chapitre 2.

A.3.1 Courbes d'apprentissage.

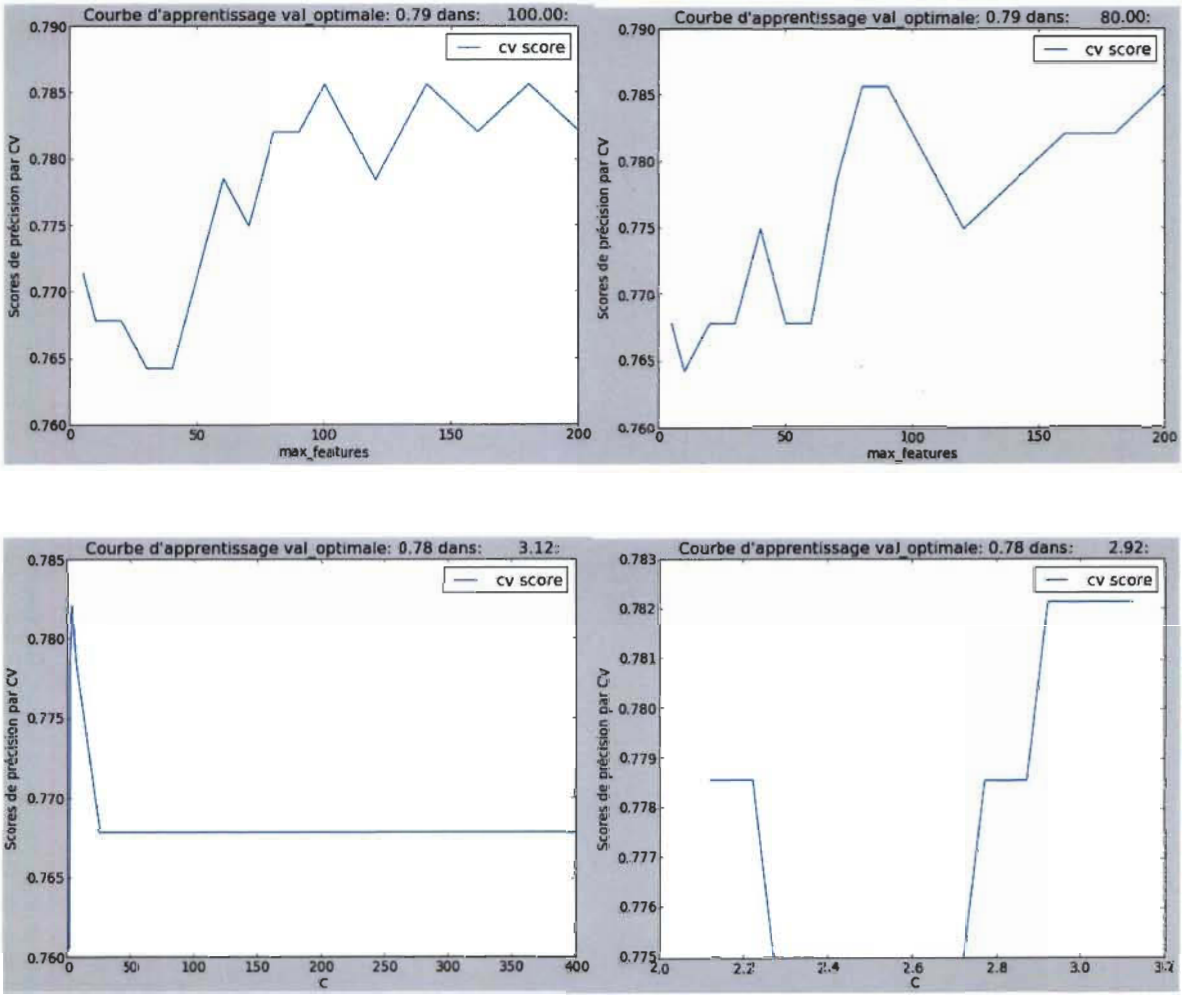


FIGURE A.4 – Courbes d'apprentissage du chapitre (classe) 2. En haut : à gauche RF gini, à droite RF entropie. En bas : à gauche SVM itération 1, à droite itération 2.

A.3.2 Exploration du contenu.

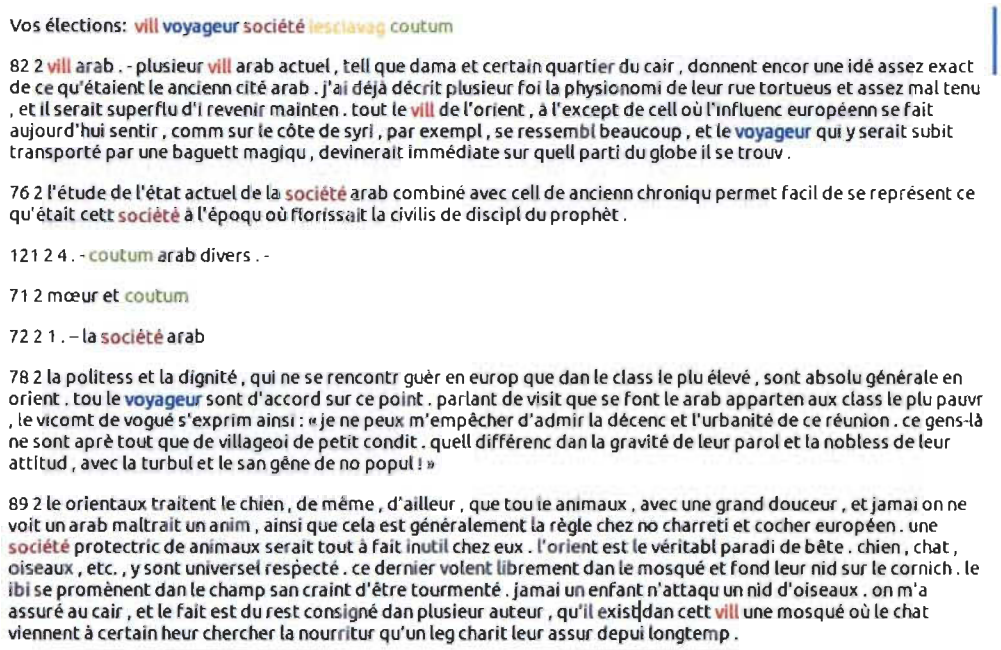


FIGURE A.5 – Affichage des segments du chapitre 2.

A.4 Compléments d'analyse du chapitre 3.

A.4.1 Courbes d'apprentissage.

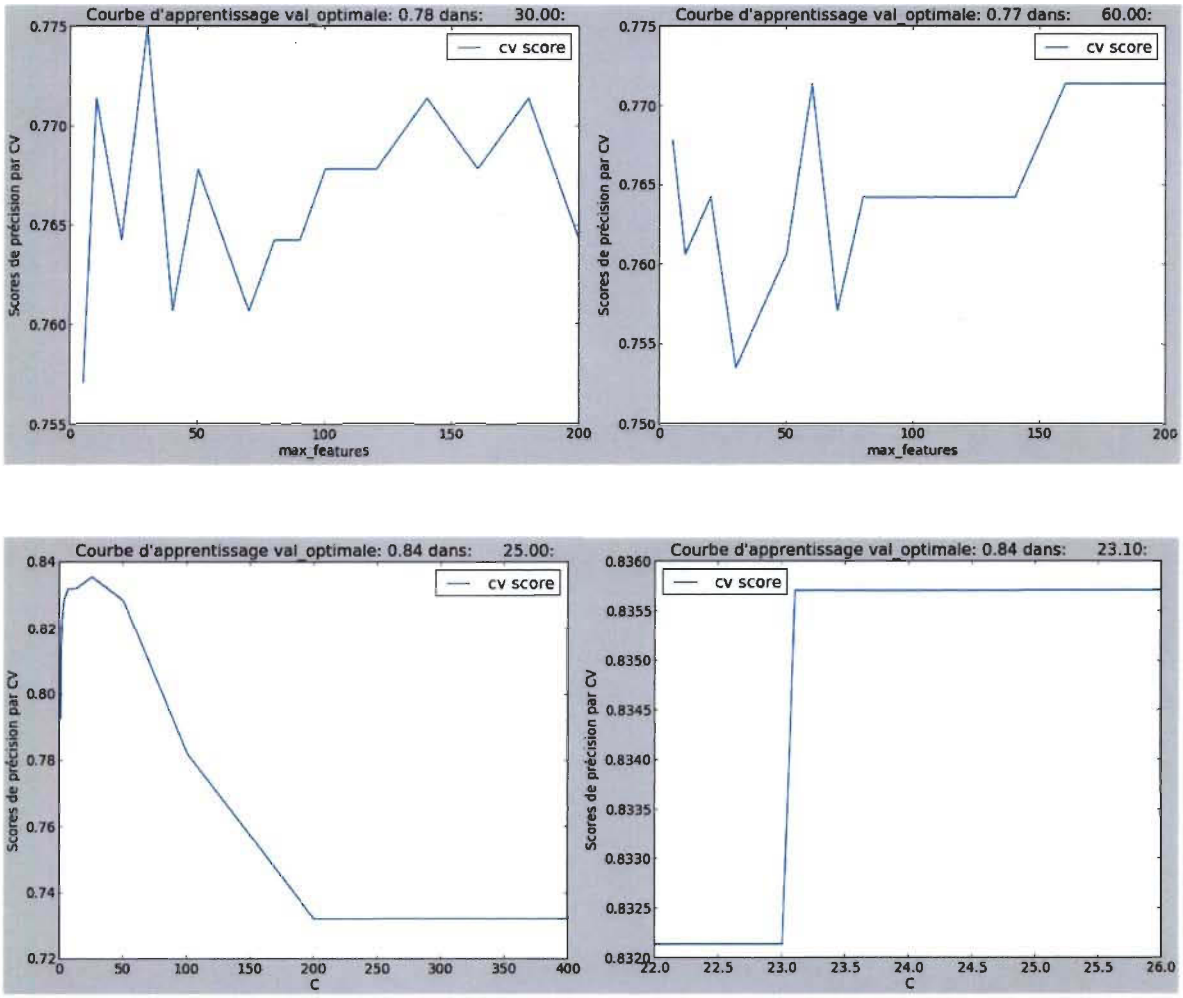


FIGURE A.6 – Courbes d'apprentissage du chapitre (classe) 3. En haut : à gauche RF gini, à droite RF entropie. En bas : à gauche SVM itération 1, à droite itération 2.

A.4.2 Exploration du contenu.

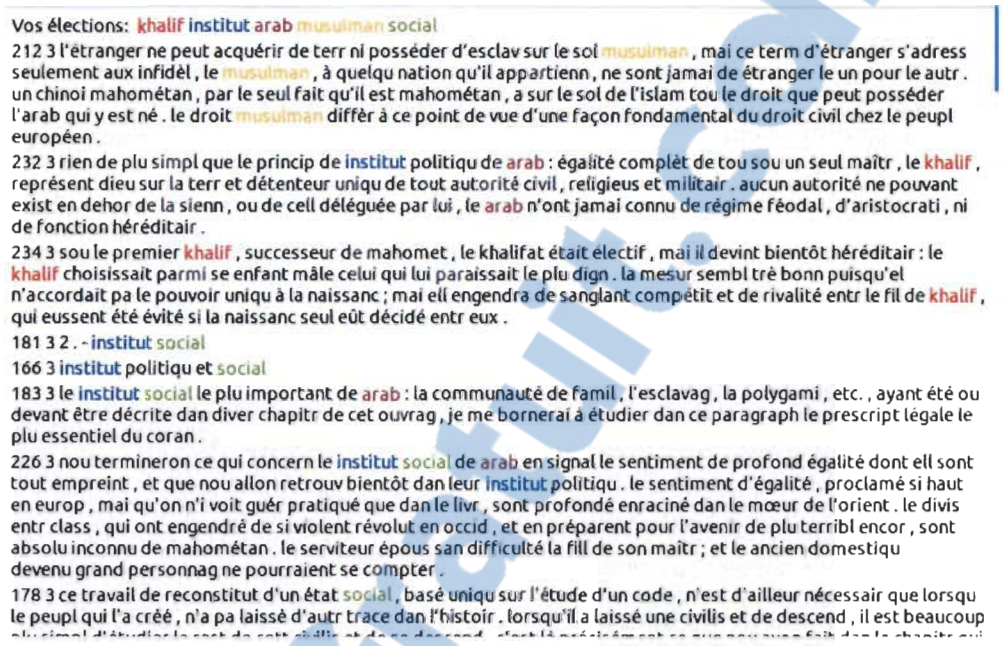


FIGURE A.7 – Affichage des segments du chapitre 3.

A.5 Compléments d'analyse du chapitre 4.

A.5.1 Courbes d'apprentissage.

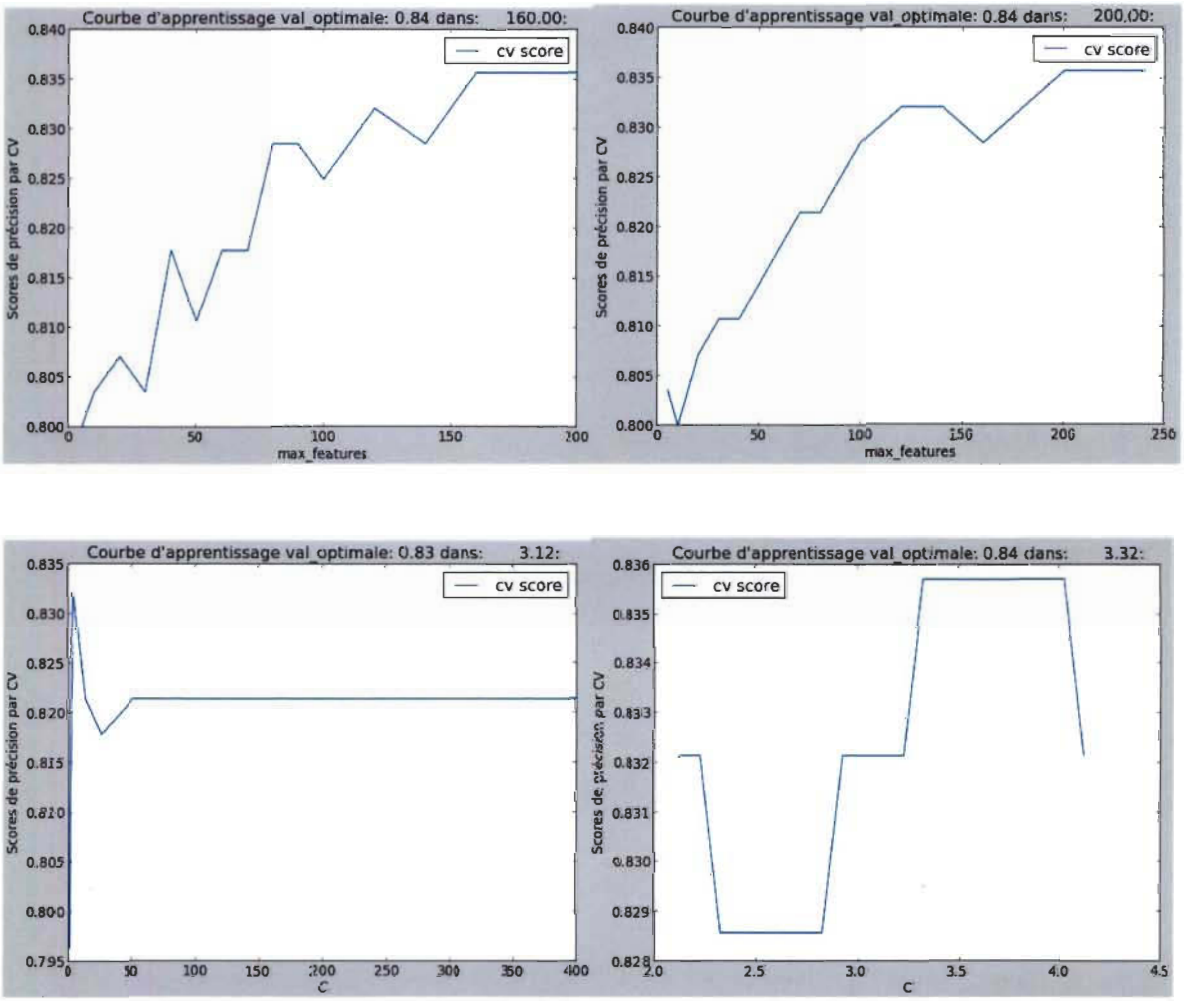


FIGURE A.8 – Courbes d'apprentissage du chapitre (classe) 4. En haut : à gauche RF gini, à droite RF entropie. En bas : à gauche SVM itération 1, à droite itération 2.

A.5.2 Exploration du contenu.

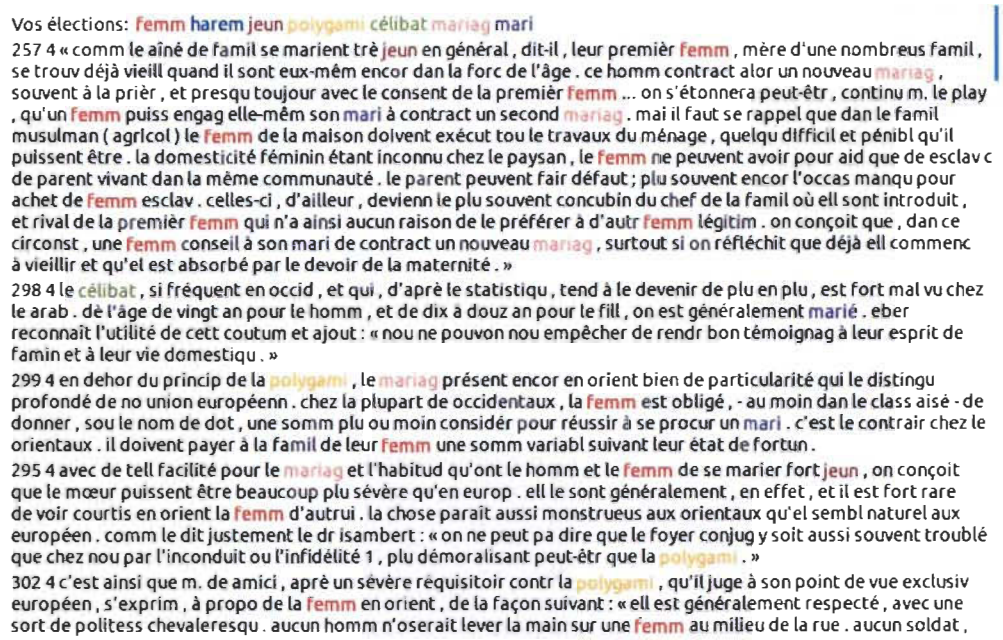


FIGURE A.9 – Affichage des segments du chapitre 4.

A.6 Compléments d'analyse du chapitre 5.

A.6.1 Courbes d'apprentissage.

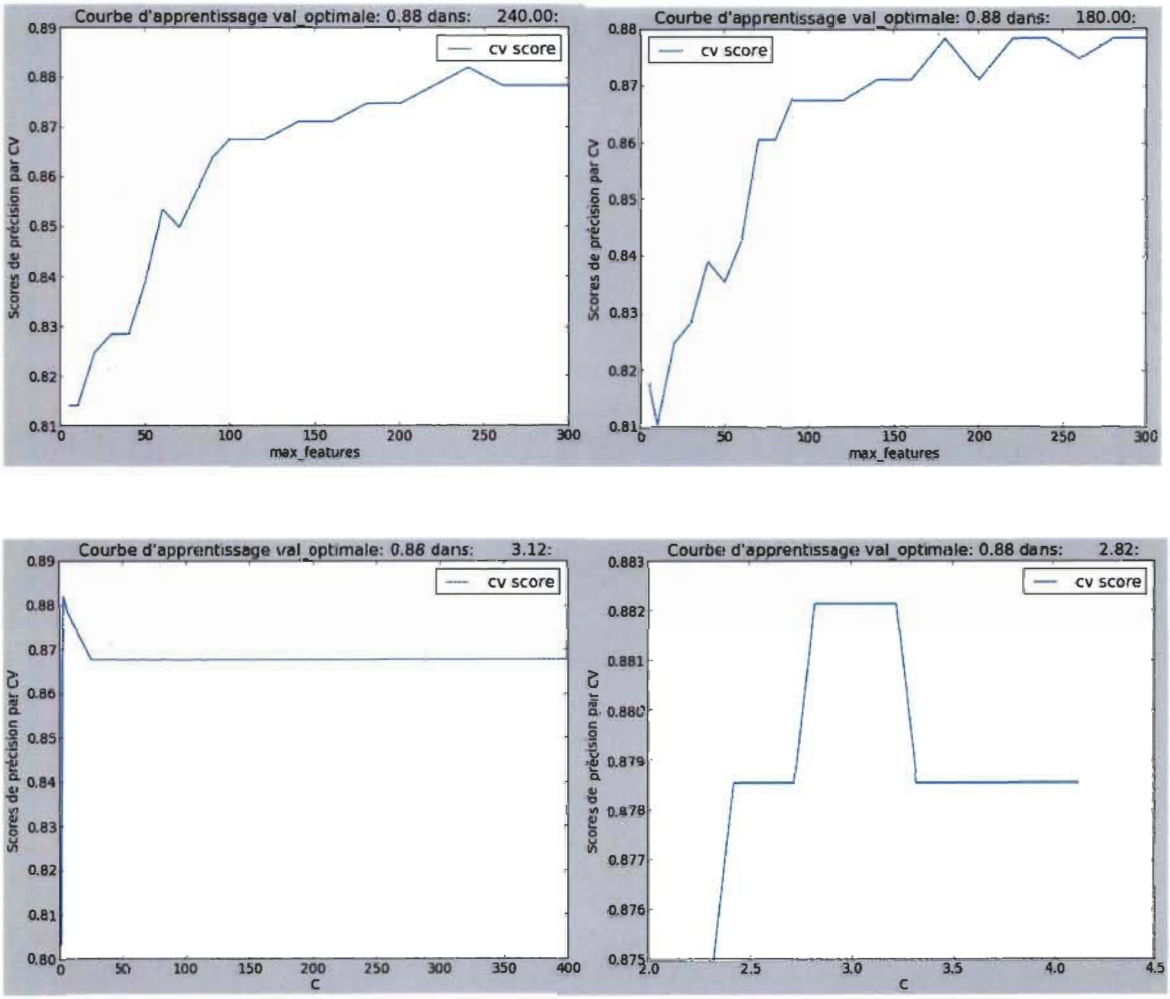


FIGURE A.10 – Courbes d'apprentissage du chapitre (classe) 5. En haut : à gauche RF gini, à droite RF entropie. En bas : à gauche SVM itération 1, à droite itération 2.

A.6.2 Exploration du contenu.

FIGURE A.11 – Affichage des segments du chapitre 5.