République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur Et de la Recherche Scientifique

Université Ibn Khaldoun Tiaret

Faculté des Sciences et des Sciences de l'Ingénieur

Département d'Informatique

MÉMOIRE

Présenté pour l'obtention du diplôme de Magister

Filière :

Informatique

Option :

Système d'Information et de Connaissances

Réalisé par:

Mr. MOSTEFAOUI Sid Ahmed Mokhtar

Classification des documents par apprentissage

Soutenu devant le jury composé de :

Mme H.BELBACHIR	Présidente	Professeur Université USTO (ORAN)
Mme L.ZAOUI	Encadreur	M.C.A Université USTO (ORAN)
Mr B.DJEBBAR	Examinateur	Professeur Université USTO (ORAN)
Mr K.SAADOUNI	Examinateur	M.C.A Université USTO (ORAN)
Mr Y.DAHMANI	Examinateur	M.C.A Université de Tiaret
Mr A.MAATOUG	Invité	M.A Université de Tiaret

Année : 2010

Remerciements

Je souhaite tout d'abord témoigner ma profonde reconnaissance à Mme Lynda ZAOUI, pour avoir accepté de diriger ce travail de recherche ainsi que Mr Abdelkader MAATOUG, pour ses conseils et aide qui ont bien contribué à son avancement.

Je suis très reconnaissant envers le Professeur Hafida BELBACHIR de me faire l'honneur de présider le jury de cette soutenance. J'exprime toute ma gratitude au Professeur DJEBBAR et au Professeur SAADOUNI pour l'intérêt qu'ils portent à mon travail et en particulier pour avoir voyagé afin d'assister à cette présentation. J'exprime toute ma gratitude au Professeur Youcef DAHMANI pour accepter d'être un examinateur et pour son aide administratif.

Je souhaite également remercier ma famille, et plus particulièrement mes parents, mon épouse et mes enfants MERIEM et ASSIA pour leurs encouragements constants.

Résumé

L'apprentissage du support vector machine (SVM) mène à un problème d'optimisation quadratique sous contraintes linéaires bornées. Malgré ce problème est claire, Il devient impossible, en termes de stockage mémoire et temps d'apprentissage, d'être résolu pour un nombre d'exemples d'apprentissage très élevé. Pour l'objectif de réduire le temps d'apprentissage, on propose ici un algorithme qui s'inspire de la méthode de décomposition proposé par Osuna dédié à l'optimisation des SVMs : il segmente le problème d'optimisation initial en sous problèmes calculable par la machine en terme de temps CPU et stockage en mémoire, la solution obtenue s'avère en pratique plus parcimonieuse que celle trouvée par l'approche d'Osuna en qualité de temps d'apprentissage , tout en offrant des performances similaires**.**

Mots clés : Représentation vectorielle, Classification, Apprentissage, Support Vector Machines (SVM), Optimisation quadratique, Décomposition.

Abstract

Training a support vector machine (SVM) leads to a quadratic optimization problem with bound constraints and one linear equality constraint. Despite the fact that this type of problem is well understood, there are many issues to be considered in designing an SVM learner. In particular, for large learning tasks with many training examples, off-the-shelf optimization techniques for general quadratic programs quickly become intractable in their memory and time requirements. Here we propose an algorithm which aims at reducing the learning time, this algorithm is based on the decomposition method proposed by Osuna dedicated to optimizing SVMs: it divides the original optimization problem into sub problems computable by the machine in terms of CPU time and memory storage, the obtained solution is in practice more parsimonious than that found by the approach of Osuna in terms of learning time quality, while offering similar performances.

Keywords: Vector representation, Classification, Learning, Support Vector Machines (SVM), Qquadratic optimization, Decomposition.

Table des matières

Chapitre III : Les algorithmes de décomposition

Liste des figures

Liste des tableaux

Introduction générale

Introduction

La classification de textes a pour objectif de regrouper les textes similaires, c'est-à-dire thématiquement proches, au sein d'un même ensemble. L'intérêt d'une telle démarche est d'organiser les connaissances de façon à pouvoir effectuer, par la suite, une recherche ou une extraction d'information efficace.

La classification automatique de textes a débuté vers les années 1960. Mais les premiers pas dans ce domaine ont surtout été motivés par les progrès technologiques des années 1980 qui ont permis d'augmenter les capacités de stockage numérique et, par conséquent, le nombre de documents textuels à traiter. Grâce à ces progrès technologiques, le volume de documents numériques n'a cessé de croître jusqu'à en rendre impossible une classification manuelle. Des besoins en classification automatique se sont donc fait ressentir aussi bien sur Internet (moteurs de recherche) qu'au sein des entreprises (classement de documents internes, de dépêches d'agences, etc.).

On distingue dans le domaine de la classification automatique deux types d'approches : la classification supervisée et la classification non supervisée. Ces deux méthodes diffèrent sur la façon dont les classes sont générées. En effet, dans le cas de la classification non supervisée, les groupes de documents (classes) sont calculés automatiquement par la machine, tandis qu'ils sont, dans l'approche supervisée, définis par un expert. L'objectif de la catégorisation est d'affecter une étiquette à un document en fonction de son contenu textuel. Par conséquent, lorsque nous déléguons cette tâche à un ordinateur nous devons représenter chacun des textes dans un formalisme permettant d'appréhender de manière plus ou moins fine sa sémantique si l'on veut avoir une classification pertinente.

L'apprentissage automatique propose une gamme d'outils qui permettent d'avancer dans ces directions. C'est dans ce cadre que se situe notre travail, qui vise à explorer le potentiel des techniques d'apprentissage en particulier la technique Support vectors machine « SVM » d'une part, répondre aux besoins de la classification des documents et d'autre part présenter plusieurs avantages en matière des gains en complexité algorithmique et en espace mémoire.

Chapitre I

Les modèles de représentation des documents

I.1) La similarité textuelle :

La notion de similarité textuelle est très souvent utilisée dans les applications de Traitement de la langue destinées à l'exploitation de corpus textuels de grande taille. Par exemple, en Recherche Documentaire, les documents pertinents retournés par le moteur de recherche sont les plus proches de la requête selon une certaine mesure de similarité **[Salton et McGill, 1983]** ; de même, dans le cas de la structuration automatique de bases de données textuelles (classification non supervisée), les documents sont également regroupés en classes en fonction d'une mesure de similarité spécifique **[Salton et al, 1975]**.

L'objectif de nombreuses tâches des systèmes de traitement du langage naturel comme la recherche documentaire (RD) et la classification automatique, repose sur la notion de similarité textuelle qui est généralement calculée à partir d'une représentation spécifique des documents dans un espace défini à priori. Une approche standard très utilisée en RD, est l'utilisation de mesures de similarité (au sens mathématique) sur des représentations vectorielles des documents.

I.1.1) Modèle de similarité textuelle :

Le modèle de similarité est une modélisation générale permettant de formaliser le représentation de textes pour le calcul de similarités sémantiques. Formellement un modèle de similarité textuelle (MST) est défini ici comme un tuple (V, R, δR, repD) **[Besançon, 2002]**, où

V	est un vocabulaire fini $m_{1,}$……, $m_{\|V\|}$ i.e l'ensemble des mots (formes de surface)
R	est l'espace de représentation des documents
$\delta_{R:}$ R x R ⟶ IR^+	est une mesure de dissimilarité sur R
$rep_D : V^*$ ⟶ R	est la fonction de représentation, qui à chaque document fait correspondre sa représentation

La définition d'un MST nécessite donc :

- Le choix d'un espace de représentation spécifique R ainsi que d'une fonction de représentation rep_D permettant de projeter les documents dans cet espace.

- Le choix d'une mesure de dissimilarité sur R, i.e. une fonction δ_R : R x R $\longrightarrow$ IR^+ vérifiant :

$$\forall (x,y) \in R^2 \begin{cases} \delta_R(x,y) \geq 0 \\ \delta_R(x,y) = 0 \Leftrightarrow x = y \\ \delta_R(x,y) = \delta_R(y,x) \end{cases}$$

Notons que la condition de symétrie peut être relâchée dans certains cas. Il est également possible d'utiliser une mesure de similarité sur R plutôt qu'une mesure de dissimilarité. Nous utiliserons néanmoins la même notation δ_R dans les deux cas. Notons qu'une mesure de dissimilarité δ_d peut toujours être transformée en une mesure de similarité δ_s par une simple transformation $\delta_s = 1/(1+\delta_d)$, mais le contraire n'est pas toujours vrai **[Van Rijsbergen, 1979]**.
Une distance (ou métrique) est une dissimilarité qui vérifie de plus la condition de l'inégalité triangulaire :

$$\delta_R(x,y) \leq \delta_R(x,z) + \delta_R(z,y)$$

Des mesures de distance peuvent donc être également utilisées pour calculer la similarité sémantique entre documents (l'inégalité triangulaire n'est néanmoins pas requise d'office pour cette mesure).
Un MST peut donc être représenté schématiquement par le schéma suivant :

$$V^* \xrightarrow{\text{rep}_D} R$$

I.2) Déséquentialistation et Composition :

[Besançon, 2002], a fait une hypothèse supplémentaire : on s'intéresse aux modèles de similarité textuelle dans lesquels la fonction de représentation des documents à partir des mots se décompose en deux phases de traitement : une phase de déséquentialisation et une phase de composition.

I.2.1) La Déséquentialistation :

La phase de déséquentialisation repose sur l'idée que l'information du sens d'un document peut être portée par un ensemble d'unités linguistiques particulières. Plus précisément, le sens induit par la séquence de mots donnée par le texte peut être rattaché à certaines unités linguistiques spécifiques, à l'aide de prétraitements linguistiques adaptés.

L'hypothèse que le document peut être représenté par un ensemble d'unités linguistiques n'est pas simpliste puisqu'elle permet d'utiliser des unités linguistiques aussi complexes que l'on veut : en particulier, elle peut également permettre de considérer que la structure syntaxique de la phrase est également porteuse d'une information sémantique importante, et que les liens syntaxiques (sujet, objet, groupe syntaxique…) ou sémantiques (procès, actants, circonstants…) repérés par une analyse avancée sont porteurs d'une information que l'on souhaite conserver. Dans ce cas, une unité linguistique pour la représentation peut être un tuple ou une relation (par exemple, une unité linguistique pourrait être SUJ(X,Y)) **[Besançon, 2002]**.

Comme mentionné, la première étape pour la définition d'un MST est donc la définition de l'ensemble des unités linguistiques U sur lequel repose la représentation des textes, ainsi que la définition de la fonction de prétraitement linguistique $\varphi : V^* \longrightarrow 2^{UxIN}$, qui transforme une séquence de mots du vocabulaire en un multi ensemble d'unités linguistiques.

Les unités linguistiques les plus simple sont les formes de surface des mots tels qu'ils apparaissent dans le document. Ces unités linguistiques sont en général évidemment peu représentatives du sens réel du document, pour plusieurs raisons : les unités repérées ne correspondent pas forcément à des unités linguistiques pertinentes, elles sont ambiguës (des formes de surface identiques peuvent relever des mots différents d'un même mots), elles sont variables (plusieurs formes de surfaces différentes peuvent relever du même mot ou de mots différents ayant le même sens), et elles n'ont pas toutes la même importance sémantique. Un certain nombre de traitements linguistiques peuvent être mise en œuvre pour identifier des unités linguistiques plus pertinentes **[Gaussier et al, 2000]**.

Segmentation des unités linguistiques : La segmentation des unités linguistiques simples est un problème de base pour la représentation des documents. Cette opération peut être faite par l'application de certaines règles (*tokenisation*) **:** par exemple utiliser l'espace ou la ponctuation

comme des séparateurs d'unités. Cette solution est simple à mettre en œuvre, mais constitue une approche relativement grossière : en français, l'apostrophe doit –elle être ou non considéré comme un séparateur « aujourd'hui », l'espace lui-même n'est pas un séparateur très fiable « parce que ».Une solution peut être d'utiliser un lexique et d'identifier dans la phrase des élément du lexique (*lexematisation*) : les unités linguistiques considérées sont donc les lexèmes. Le problème se pose alors pour les formes inconnues ou pour les mots composés, des expressions figées, ou des collocation (les unités complexes sont regroupées sous l'appellation *multi-terme*). Si on veut les considérer comme unités linguistiques, on peut soit les ajouter dans le lexique, soit par exemple introduire des règles de reconnaissance des multi-termes : en général, ces règles prennent en compte un filtrage linguistique sur des patrons morpho-syntaxiques, et un filtrage statistique sur les fréquences d'apparition es multi-termes **[Daille, 1994].**

Ambiguïté des mots : le problème de l'ambiguïté est double. L'ambiguïté peut être lexicale : une forme de surface peut correspondre à plusieurs mots (*tour* est différent suivant qu'on fasse référence à « une tour » ou à « un tour »). Si cette ambiguïté correspond à des unités discriminables par les propriétés morpho-syntaxiques, elle peut être résolue par un étiquetage morpho-sytaxique automatique des mots du documents; les méthodes pour réaliser cet étiquetage sont efficaces et rapides, par exemple à base de règles **[Brill, 1992]**, ou de chaînes de Markov cachées . les unités linguistiques considérées seront alors associés à une étiquette morpho-syntaxique.

L'ambiguïté peut également être sémantique : un mot peut avoir plusieurs sens (avocat peut désigner un fruit ou un homme de barreau). Cette ambiguïté doit être traitée par des méthodes de désambiguïsation sémantique souvent plus délicates à mettre en œuvre car elle demande souvent les ressources importantes. Néanmoins, la désambiguïsation semble être un des problèmes clés pour la représentation de documents, par exemple pour la recherche documentaire, où il est important de connaître le sens des mots de la requête et des documents pour éviter de considérer comme pertinents des documents contenant les mots de la requête mais utilisés dans un sens différent **[Besançon, 2002].** Plusieurs travaux ont été envisagés pour résoudre le problème de la désambiguïsation sémantique pour la recherche documentaire, par exemple **[Mihalcea et Moldovan , 2000]**.

Variabilité des mots : Le Problème de la variabilité est lui aussi multiple. La variabilité peut être flexionnelle (passage au pluriel : journal → journaux, conjugaisons des verbes : chanter→ chante, chanteront) ou dérivationnelle (passage d'une catégorie morpho-syntaxique à une autre, pour une même racine, peuple → peupler → peuplement). On fait en général une hypothèse que les modifications de sens liées au variation morphologiques sont suffisamment faibles, en particulier dans le cadre de la recherche documentaire, pour pouvoir être ignorée, ainsi des techniques de désuffixation (*stemming*) sont souvent utilisées, par exemple pour l'anglais on trouve l'algorithme de stemming de **[Porter, 1980]**, et pour le français on trouve l'algorithme de **[Savoy, 1999]**.

I.2.2) La Composition :

La phase de composition repose sur l'idée que la représentation sémantique d'un document peut ensuite être construite en composant (selon une méthode de composition appropriée) les représentations des unités linguistiques extraites lors de la phase de déséquentialisation **[Besançon, 2002]**. Le problème de la comopositionnalité est un problème général des méthodes sémantiques **[Nazarenko, 1998]** : le sens du tout est-il « calculable » à partir du sens des parties ?

I.3) La représentation de corpus documentaires :

Il est souvent admis que le sens d'un document peut être porté par un ensemble d'unités linguistiques particulières, caractéristiques plus ou moins élaborées issues de l'analyse du corpus documentaire **[Besançon, 2002].** Chaque unité linguistique définit une « idée », et par conséquent, un ensemble de mêmes unités linguistiques est censé définir un sens identique.

Les premières unités linguistiques à s'être imposées comme représentatives du sens sont le radical et le lemme des mots. La reconnaissance de ces unités linguistiques nécessite d'effectuer un prétraitement linguistique des mots du texte. Certaines unités linguistiques, plus rudimentaires, ne nécessitent aucun traitement a priori, comme le « sac de mots » ou la phrase.

I.3.1) Les unités linguistiques:

a) Le mot : Dans cette approche l'unité linguistique choisie est le mot tel qu'il apparaît dans le document. Cette approche ne nécessite aucun traitement préliminaire, chaque mot est extrait du texte en considérant des séparateurs entre chaque mot, comme l'espace, la virgule, la tabulation, le point et la ponctuation en général. L'approche par sac de mots est généralement associée à une

procédure de filtrage. En effet, le nombre de mots caractérisant un corpus de documents peut rapidement atteindre une centaine de milliers. Il devient alors nécessaire, afin d'envisager un traitement analytique des textes, de ne conserver qu'un sous-ensemble de ces mots. Classiquement, le filtrage repose à la base sur les fréquences d'occurrence des mots dans le corpus.

b) La phrase : Certaines approches utilisent non pas des mots, mais des groupes de mots, voire des phrases, comme éléments représentatifs du sens. L'intérêt d'utiliser une phrase ou un groupe de mots est double. Tout d'abord ce type d'unités linguistiques possède une unité de sens plus complète qu'un simple mot de par la « Contextualisation » des mots **[Lewis, 1992a].** En effet, l'association d'un ensemble de mots, même dans le désordre, offre davantage d'information sur le champ sémantique dans lequel on se trouve. De plus, ce type d'unités linguistiques définit une relation d'ordre entre les mots : un des avantages de cette représentation est qu'elle ouvre la porte à la gestion des cooccurrences de mots. En pratique, les expérimentations avec ce type de représentation ne sont, pour l'instant, pas très concluantes. En effet, si la phrase est une unité linguistique à forte valeur sémantique ajoutée, la fréquence d'apparition de groupes de mots ne permet pas d'offrir des statistiques fiables. Le grand nombre de combinaisons entre les mots engendre des fréquences trop faibles pour être exploitables.

c) La technique des « n-grammes » :

Les n-grammes ont été introduits par Shannon en 1948 **[Shannon, 1948].** En général le n-gramme se définit comme étant une séquence de n caractères consécutifs. Cependant il existe quelques variantes dans la littérature. En effet dans **[Caropreso et al., 2001]**, l'auteur définit le n-gramme comme étant une séquence de n "mots désuffixés" et dans **[Cavnar et al., 1994]** l'auteur définit le n-gramme comme étant une séquence de n caractères non obligatoirement consécutifs. Nous nous contenterons ici de la notion de n caractères consécutifs. Ainsi, le mot « chèse » est composé des n-grammes suivants :

- bi-grammes : « _c », « ch », « hè », « ès », « se », « e_ »

- tri-grammes : « __c », « _ch », « chè », « hès », « èse », « se_ », etc.

De manière générale pour une chaîne de k caractères entourée de blancs, on génère k + 1 bigrammes, k + 1 tri-grammes ou plus généralement k + 1 n-grammes **[Cavnar et al., 1994].** Pour un document quelconque l'ensemble des n-grammes est obtenu en déplaçant une fenêtre de n caractères sur le texte. Avant chaque déplacement de 1 caractère de la fenêtre, le n-gramme contenu dans la fenêtre est défini comme étant un des n-grammes du document. Une fois extraits tous les n-grammes d'un document, on définit par profil d'un document, la liste des n-grammes triés par ordre décroissant de leur fréquence d'apparition. Les avantages liés à l'utilisation des n-grammes sont multiples. Tout d'abord, les méthodes basées sur les n-grammes sont indépendantes de la langue. Elles ne nécessitent ni la segmentation des documents en unités linguistiques, ni la mise en place de prétraitements tels que le filtrage, la désuffixation ou la lemmatisation. Les n-grammes sont aussi tolérants aux bruits (principalement les fautes de frappe quelles qu'elles soient) **[Manning et al., 1999].** Pour un mot mal orthographié dans un texte, avec l'utilisation des n-grammes, le mot n'est pas perdu dans sa totalité. Par exemple le mot « gamme » mal orthographié peut donner « game ». Les bi-grammes de ces deux mots donnent :

- « gamme » : « _g », « ga », « am », « mm », « me », « e_ »

- « game » : « _g », « ga », « am », « me », « e_ »

On remarque que malgré la faute de frappe, il n'y a qu'un bi-gramme qui diffère entre le mot correct et le mot mal orthographié.

I.3.2) Le prétraitement des documents :

I.3.2.1) Stemmatisation (radicalisation ou « stemming »)

L'utilisation de mots comme unité linguistique est possible, mais pose toutefois un certain nombre de problèmes. En effet, il existe plusieurs centaines de milliers de mots dans un corpus documentaire et associer un sens à chacun de ces mots n'est pas nécessairement des plus pertinents. En effet, beaucoup de mots ont des racines communes, des sens communs. Attribuer un sens différent à « chantaient » et « chantent » par exemple, relèverait d'une redondance sans pertinence sémantique.

La stemmatisation qui se nomme également radicalisation ou en anglais « stemming » est un prétraitement qui consiste à trouver la racine de chaque mot. C'est un traitement qui procède à

une analyse morphologique du texte. Une implémentation très connue est celle de Porter **[Porter, 1980].** Cet algorithme consiste principalement à effectuer une « désuffixation ». C'est un traitement fondé sur l'étude de la morphologie des mots. Il se base sur un dictionnaire de suffixes qui permet d'extraire le radical du mot.

La stemmatisation de : « Ces vases sont disproportionnés », grâce à l'algorithme de Porter donne : « Ce vas sont disproportion ». Une vraie radicalisation aurait donné : « Ce vase sont proportion » **[Plantié, 2006]**.

I.3.2.2) Lemmatisation

La lemmatisation nécessite une analyse plus poussée que la stemmatisation. La lemmatisation se fonde sur un lexique qui est un ensemble de lemmes, que l'on peut assimiler globalement aux entrées d'un dictionnaire.

L'objectif de la lemmatisation est d'associer à chaque mot une entrée dans le lexique. Or, l'analyse morphologique est insuffisante pour extraire les lemmes d'un texte car de nombreux mots de même graphie peuvent provenir de différents lemmes. Par exemple « offense » peut être considéré comme le lemme définissant la parole ou la faute qui blesse, ou comme le verbe "offenser" conjugué. Cette ambiguïté se résoud en analysant la catégorie grammaticale du mot en question dans la phrase. La lemmatisation nécessite donc de réaliser une analyse supplémentaire : l'analyse syntaxique. La lemmatisation recouvre donc deux analyses regroupées sous le terme d'analyse morphosyntaxique. Depuis la fin des années 80, les lemmatiseurs sont capables d'associer à chaque mot d'un texte, son lemme, grâce à un étiqueteur morphosyntaxique (nom, verbe, adjectif, etc.) dont les taux de réussite avoisinent les 90% **[Schmid, 1994].** La construction d'un lemmatiseur nécessite néanmoins un étiquetage de quelques milliers de mots.

I.3.2.3) La « stop-list »

Avant d'effectuer un des prétraitements précédents, il est usuel d'utiliser une «stop-list ». L'objectif de la « stop-list » est d'éliminer tous les mots ne participant pas activement au sens du document. La « stop-list » est une liste répertoriant tous les mots outils (pronoms, articles, etc.) et les mots trop fréquents pour être discriminants. D'un point de vue linguistique les mots outils sont par définition des mots "vides" de sens. Et, d'un point de vue statistique, les mots trop fréquents (et de distribution uniforme) ne sont d'aucune aide à un processus de catégorisation puisque non

discriminants. De par sa nature ce prétraitement s'effectue donc en amont des autres prétraitements linguistiques et son objectif est d'éliminer toutes les unités linguistiques non discriminantes. L'inconvénient de la « stop-list » est qu'elle reste dépendante d'une langue donnée. La « stop-list » pose un autre problème plus fondamental. En effet, qui peut dire a priori que tel type syntaxique ou mot « outil » est vide de sens ? On risque dans certains cas d'omettre certaines unités linguistiques qui auraient permis d'apporter une aide précieuse à la tâche de classification. En particulier, il est peut être imprudent d'établir cette liste avant d'avoir fait le choix de la technique de classification qui peut parfois nécessiter toutes les informations disponibles.

I.3.2.4) La loi de Zipf

Il est plus simple pour un auteur (rédacteur d'un document) de répéter les mots que d'en chercher de nouveaux. Dans les années 1930, un scientifique de l'université de Harvard, G.K. Zipf, **[Zipf 1949]** a constaté que la fréquence du second mot le plus fréquent est la moitié de celle du premier, la fréquence du troisième mot le plus fréquent, son tiers, etc... En classant les mots d'un texte par fréquence décroissante, on observe que la fréquence d'utilisation d'un mot est inversement proportionnelle à son rang. Cette loi peut s'exprimer de la manière suivante :

Fréquence d'un mot de rang N = (Fréquence du mot de rang 1) / N

À partir de cette loi, on constate qu'en traçant pour chaque mot le couple rang / effectif dans un repère logarithmique, alors le nuage de points paraît linéaire. Il est d'ailleurs possible de calculer l'équation de la droite de régression linéaire sur ce nuage. La répartition des points autour de cette droite montre que la linéarité n'est qu'approximative mais reste toutefois fortement significative.

En étudiant la liste des fréquences, on peut noter que les mots les plus fréquents sont les mots grammaticaux, et que leur ordre d'apparition dans la liste est stable d'un texte à l'autre à partir du moment où le texte est de longueur significative. Les mots lexicaux apparaissent ensuite, en commençant d'abord par les mots thématisant le document. Cette loi, pertinente dans le domaine littéraire, laisserait supposer que les termes significatifs d'un document se trouvent juste après écrêtage des articles et mots grammaticaux. Lors de la préparation des données, il est fréquent de supprimer les termes fréquents, car considérés comme peu porteurs de sens, ainsi que les termes trop rares (hapax, termes dont le nombre d'occurrences est inférieur à un seuil, généralement de 3

à 5, mais historiquement le hapax est un terme présent une seule fois dans un corpus), car considérés comme étant probablement des erreurs, par exemple typographiques.

I.4) La recherche du document le plus similaire :

Le cadre des MST permet de définir le calcul de la similarité sémantique entre deux documents. Dans un cadre applicatif, l'utilisation de la notion de similarité passe souvent par la recherche de documents particuliers dans la collection documentaire en fonction de leurs similarités (en particulier, pour un document, la recherche de document le plus similaire).

Le problème de la recherche se pose en particulier à cause de la grande taille des collections documentaires : cette recherche du document le plus similaire est un thème de recherche particulier qui a donné lieu à de nombreuses études (regroupées sous l'étiquette *nearest neighbour search* ou *similarity search* **[Weber, 1998]**).

[Van Rijsbergen, 1979] présente par exemple plusieurs méthodes de recherche dans le cadre de la recherche documentaire : recherche sérielle simple (les document sont tous regardés un par un), recherche à base de classes (les documents sont regroupés en classes, et la recherche s'effectue d'abord avec les représentants des classes avant de s'effectuer avec ses documents dans les classes les plus proches), et recherche à base de retour de pertinence (*relevance feedback*) qui utilise l'information de pertinence des premiers documents trouvés pour raffiner la recherche.

Un problème supplémentaire se pose pour une grande dimensionnalité des espaces de représentation : lorsque la dimension de l'espace augmente, la différence entre les documents les plus proches et les documents les plus lointains diminue. Ce problème est connu sous le nom de *curse of dimensionality* et a été étudié par exemple dans **[Beyer et al, 1999]**. Ce problème se pose de façon théorique pour certaines hypothèses sur la distribution des documents (par exemple, une distribution uniforme) : le succès expérimental de techniques d'indexation en haute dimensionnalité laisse penser que ces hypothèses ne sont pas toujours vérifiées dans la pratique **[Besançon, 2002]**.

I.5) Les modèles vectoriels de représentation de documents :

Le rôle de la représentation textuelle est de projeter les documents au sein d'une représentation mathématique qui en permette le traitement analytique, tout en conservant au maximum la

sémantique de ceux-ci. Pour réaliser cette projection la représentation mathématique généralement utilisée est l'utilisation d'un espace vectoriel comme espace de représentation cible. La caractéristique principale de la représentation vectorielle est que chaque « brique de base du sens » ou unité linguistique (segment textuel) est associée à une dimension propre au sein de l'espace vectoriel. Deux textes utilisant les mêmes segments textuels seront donc projetés sur des vecteurs identiques. Le formalisme le plus utilisé pour représenter les textes est le formalisme vectoriel issu de **[Salton, 1971]** et **[Salton, 1983]**.

I.5.1) Le modèle vectoriel standard :

Dans le modèle vectoriel standard, chaque document *d* est représenté par un vecteur à *n* dimensions ($w_1,\ldots, w_n$), où w_i est le poids du terme t_i dans le document *d*. Un terme peut être un mot, un lemme où un composant (plusieurs mots ou lemmes ou stems). Cette représentation requiert la définition de l'ensemble des termes d'indexation et une méthode de pondération des termes. Pour chaque paire de documents (*u, v*), (où $\vec{u}$ et $\vec{v}$ sont leurs représentation vectorielle dans l'espace à *n* dimensions), une fonction de similarité $s(\vec{u},\vec{v})$ doit être définie. Pour une requête de recherche donnée *q* (une requête est également du texte et peut être convertie en un vecteur $\vec{q}$ dans le même espace vectoriel que les autres documents), la recherche documentaire est effectuée par le calcul de la similarité entre les documents et la requête. Ainsi, les documents les plus similaires à la requête sont proposés à l'utilisateur. Plus formellement, la recherche documentaire basée sur le modèle vectoriel standard peut être définie par le 5-uplet < X, Q, T, *s*, *f* >, où X représente le corpus documentaire, Q est l'ensemble des requêtes (classes dans la classification), T représente l'ensemble des termes d'indexation, *s* est la fonction de similarité et *f* est la fonction de construction des termes d'indexation tel que T = *f* (X).

I.5.1.1) La sélection des termes d'indexation :

Les termes d'indexation sont les unités linguistiques qui sont associées aux dimensions de l'espace vectoriel de représentation, et ils sont choisis parmi l'ensemble des unités linguistiques *U* en fonction de leur pouvoir de discrimination.

La sélection des termes d'indexation correspond donc à une réduction de la dimension de l'espace à |U| dimensions à un espace à |T| dimensions effectuée en fonction de critères de discrimination. D'une manière générale, les techniques de sélection des termes d'indexation

présentées ici font partie d'un domaine de recherche propre (*feature selection*) qui trouve en particulier des applications en classification et en recherche documentaire.

Le critère de sélection des termes d'indexation le plus utilisé est *la fréquence en documents*. La fréquence en documents est, pour une unité linguistique et une collection de documents donnée, le nombre de documents la contenant. **[Salton et al, 1975]** ont montré que, pour une collection de documents *D,* la sélection des unités linguistiques qui en fréquence en documents entre $|D|/100$ et $|D|/10$ génère le plus souvent un ensemble d'unités linguistiques ayant un pouvoir de discrimination satisfaisant pour la recherche documentaire. Ce critère de sélection est souvent utilisé car il est simple et rapide à mettre en œuvre et il ne nécessite pas de connaissances particulières autres que la donnée brute du corpus.

D'autres méthodes de sélection ont également été envisagées, lorsque des données de références sont disponibles, en particulier dans le domaine de la classification automatique, parce que les algorithmes de classification sont souvent moins adaptés à des espaces de représentations de très grande taille, et qu'une sélection « agressive » des termes est souvent nécessaire. Un certain nombre de ces méthodes sont répertoriées dans **[Sebastiani, 2002]**.

Ces méthodes utilisent pour la plupart comme critère de sélection des fonctions tirées de la théorie de l'information : χ^2 (Chi-deux), gain d'information (GI), information mutuelle (IM) qui correspondent à l'intuition que les termes les plus discriminant sont ceux qui sont distribués le plus différemment dans les ensembles d'exemples positifs ou négatifs d'une classe. Les expressions mathématiques de ces fonctions sont :

$$\chi^2(t_k, c_i) = \frac{N \times (p(t_k, c_i)p(\bar{t}_k, \bar{c}_i) - p(\bar{t}_k, c_i)p(t_k, \bar{c}_i))^2}{p(t_k)p(\bar{t}_k)p(c_i)p(\bar{c}_i)}$$

$$GI(t_k, c_i) = p(t_k, c_i)\log\frac{p(t_k, c_i)}{p(c_i)p(t_k)} + p(\bar{t}_k, c_i)\log\frac{p(\bar{t}_k, c_i)}{p(c_i)p(\bar{t}_k)}$$

$$IM(t_k, c_i) = \log\frac{p(t_k, c_i)}{p(c_i)p(t_k)}$$

Où les t_k sont les termes, les c_i les classes, N est le nombre de documents dans l'ensemble d'entraînement, et les probabilités $p(t_k, c_i)$ (resp. $p(\bar{t}_k, c_i)$) sont estimées en comptant le nombre d'éléments de la classe c_i contenant (resp. ne contenant pas) le terme t_k dans l'ensemble d'entraînement. Toutes ces méthodes demandent donc d'avoir ou de ne pas avoir un terme sachant une classe. Ces fonctions sont données pour une classe et un terme : la mesure prise en compte pour la sélection d'un terme t_k dans une collection comptant m classes est, pour une fonction F, $\sum_{i=1}^{m} p(c_i)F(t_k, c_i)$.

[Yang et al, 1997] présentent une comparaison de ces différentes fonctions, et montrent que les fonctions *GI* et χ^2 ont des performances comparables à la sélection par la fréquence en documents, l'information mutuelle ne donnant, pour sa part, pas de très bons résultats. Ils montrent également qu'il existe une corrélation entre les valeurs de ces fonctions.

D'autres mesures ont également été testées ; par exemple **[Ng et al, 1997]** montre que de meilleurs résultats peuvent être obtenus en utilisant à la place du χ^2 un coefficient de corrélation égal à la racine carrée signée du χ^2 (le carré du χ^2 a en effet la propriété indésirable de considérer de la même façon les facteurs qui indiquent une corrélation positive entre un terme et une classe et ceux qui indiquent négative).

Il existe d'autres techniques de représentation, qui utilisent des techniques de réduction de dimension par d'autres méthodes que la sélection des termes d'indexation (descripteurs), comme des alternatives à cette sélection. C'est le cas par exemple de modèles qui utilisent des techniques de regroupement de termes (*clustering*), voir par exemple **[Lewis, 1992b]** pour la classification.

Les méthodes de sélection de descripteurs fournissent, en général, une liste de descripteurs ordonnés du plus important au moins important (la notion d'importance dépend de la méthode de classement considérée), il reste ensuite à déterminer combien de descripteurs sont à conserver dans cette liste. Ce nombre dépend souvent du modèle, puisque, par exemple, les machines à vecteurs supports sont capables de manipuler des vecteurs de grandes dimensions alors que pour les réseaux de neurones, il est préférable de limiter la dimension des vecteurs d'entrées.

Pour choisir le bon nombre de descripteurs, il faut déterminer si l'information apportée par les descripteurs en fin de liste est utile, ou si elle est redondante avec l'information apportée par les descripteurs en début de liste.

I.5.1.2) Les schémas de pondération :

La pondération qui est accordée à un document *d* contenant un certain terme d'indexation t_j a fait l'objet de nombreuses études **[Salton et Buckley 1988]** qui prennent en général en compte des facteurs de pondération locale, de pondération globale et de normalisation en fonction de la taille du document.

La pondération locale : Ce type de pondération prend en compte les informations locales du terme qui ne dépendent que du document. Cette pondération correspond à une fonction de la fréquence d'occurrence du terme dans le document, elle est notée *tf* (*term fréquency*). C'est le nombre de fois où le terme est présent dans le document. Les fonctions les plus utilisées sont les suivantes :

- Facteur *tf* : il indique le nombre d'occurrences d'un terme donné dans le document.
- Facteur binaire : il prend pour valeur 1 si le terme est présent dans le document, et a pour valeur 0 si le terme n'est pas présent. Ce facteur est utilisé pour les représentations de type ensembliste. Il permet aussi de donner une base de comparaison par rapport aux autres pondérations locales.

- Facteur logarithmique : Ce facteur est une fonction logarithmique de la fréquence du terme dans le document, valant : $1+\log(tf)$

Cette fonction est proposée par **[Buckley et al 1992]**, elle montre qu'un terme d'une requête, qui est présent un grand nombre de fois dans un document, n'est pas plus pertinent qu'un document contenant un petit nombre de fois plusieurs termes de la requête. D'où l'importance qu'un plus grand nombre d'occurrences d'un terme dans un document ne soit pas prédominant par rapport à un plus petit nombre d'occurrences de plusieurs termes.

- Facteur augmenté **[Salton et Buckley 1988]** : il permet de réduire les différences entre valeurs pour les poids accordés aux termes du document. Il accorde pour tous termes

présents dans le document une valeur minimale, et un poids ne dépassant pas une valeur maximale pour les termes présents plusieurs fois : $0.5 + 0.5 \times \frac{tf}{\max_{document} tf}$

La pondération globale : La pondération globale contrairement à la pondération locale s'intéresse aux informations concernant les termes et dépendant de la collection de documents. Ainsi, une pondération qui prend en compte l'importance d'un terme dans toute la collection améliore les performances dans le cadre de la recherche d'informations **[Salton et al.1975b]**. Un poids plus important doit être donné aux termes qui apparaissent le moins fréquemment dans la collection, car les termes d'indexations qui sont utilisés dans de nombreux documents ont un pouvoir discriminant moins important que ceux présents dans peu de documents. Le facteur de pondération globale dépend de l'inverse de la fréquence en documents, comme par exemple le facteur *idf* (*inverted document frequency*) **[Salton et al.1975b]**, valant pour une collection de document *D*.

$$id_f(t_i) = \log\left(\frac{|D|}{tf(t_i)}\right)$$

Où $|D|$ représente le nombre de documents de la collection et $tf(t_i)$ la fréquence du terme t_i (le terme considéré) dans l'ensemble des documents de la collection.

La normalisation : La normalisation est la prise en compte de la taille du document par rapport à la pondération locale ou globale qui sont des pondérations qui ne la prennent pas en compte. En effet, la taille d'un document joue un rôle dans le style et le vocabulaire qui est utilisé dans ce document. Les documents qui sont très longs auront tendance à utiliser les mêmes termes de façon répétée, ce qui a une importance dans le calcul des facteurs. Un document long peut comporter aussi pour des raisons stylistiques un grand nombre de synonymes d'un terme pour éviter les répétitions.

La combinaison des pondérations :

La donnée de ces trois fonctions de pondération locale, globale et de normalisation forme un schéma de pondération qui est repéré dans le système SMART de la manière suivante :

Le poids du terme t dans le document *d* = (la pondération locale de *t* dans *d*) *(la pondération globale de *t*)*(la normalisation de *d*)

Que l'on peut noter aussi : $w_1(t,d) = w_l(d,t) w_g(t) w_n(d)$

Où w_1 (t,d) représente le poids du terme *t* qui apparaît dans un document *d*, w_1 (d,t) est la pondération locale du terme *t* dans le document *d*, w_g (t) la pondération globale du terme *t* et $w_n(d)$ facteur de normalisation du document *d*. On utilise souvent la pondération suivante pour un terme t_i

$$W(t,d) = tf \times idf$$

I.5.1.3) Prise en compte des dépendances dans le modèle vectoriel :

Dans le cadre des modèles vectoriels simples, il est souvent fait l'hypothèse simpliste de l'orthogonalité des axes, ce qui signifie l'indépendance des termes, puisque dans l'espace vectoriel de représentation, les axes représentent les termes. Cette représentation pose problème lorsqu'on essaye de prendre en compte des termes synonymes, car un même sens peut être décrit par différents termes qui ne seront jamais considérés comme identiques avec cette représentation.

Certaines solutions pour remédier à ce problème ont été proposées pour tenir compte des dépendances sémantiques dans le cadre du modèle vectoriel. Elles consistent à construire un espace dans lequel les axes ne sont plus orthogonaux en se basant sur les dépendances calculées pour les différents termes.

Nous allons voir dans la suite de ce chapitre deux modèles qui, en se basant sur le modèle vectoriel simple, essaient de palier à ce problème de dépendance en prenant en compte l'aspect sens et concept des documents et des requêtes. Le premier modèle vu est le modèle LSI qui se base sur la décomposition des termes d'indexations, puis nous continuerons sur le modèle DSIR **[Besançon 2002]** qui, lui, repose sur l'hypothèse de sémantique distributionnelle et se focalise sur les co-occurrences des termes.

I.5.1.3.a) Modèle Latent Semantic Indexing (LSI) :

Latent Semantic Indexing (LSI) est un modèle algébrique de recherche de documents basé sur la décomposition des termes d'indexations au travers l'espace vectoriel. C'est une variante du

modèle vectoriel qui tente de prendre en compte, pour la représentation des documents, la structure sémantique des unités linguistiques, qui sont implicites (ici latent), représentées par leurs dépendances cachées **[Deerwester et al. 1990]** et **[Furnas et al. 1988]**.

Ainsi, le but de LSI est de transformer une représentation standard par des mots clés en une autre représentation qui permet de "meilleurs" résultats. Ce qui signifie que les documents et les requêtes sémantiquement similaires seront plus proches avec la représentation transformée qu'avec les mots-clés.

L'idée est de se placer dans un nouvel espace (généralement plus petit), espace associé au concept. Ceci peut être accomplit en plaçant les vecteurs des termes d'indexations dans l'espace dimensionnel où les dimensions sont indépendantes et réduites. Cette nouvelle dimension est une combinaison linéaire des anciennes dimensions. Le problème est que la recherche dans un espace réduit doit être supérieure à la recherche dans l'espace des termes d'indexations.

LSI utilise une matrice X (terme x documents) qui est composée des vecteurs mots clés des requêtes et de documents comme pour le modèle vectoriel standard. Ensuite, une décomposition de la matrice X est effectuée. Cette décomposition en valeur singulière (appelée aussi SVD : Singular Value Decomposition) de cette matrice X permet de créer un nouvel espace vectoriel :

$$X = T_0 S_0 D'_0$$

Où X est la matrice de document-terme original (de taille t x d), T_0 est une matrice de taille t x m, D'_0 est une matrice de taille m x d et S_0 est une matrice diagonale de taille m x m qui est trié dans l'ordre croissant. Il existe juste une seule décomposition de cette façon.

La représentation par mots-clés contient beaucoup de bruits, ces bruits se retrouvent dans les dimensions de S_0 qui ont des valeurs faibles. Le modèle LSI supprime ces dimensions de valeurs faibles (en les remplaçant par la valeur 0), ce qui diminue la dimension de S_0 à k, cette matrice modifiée est appelée maintenant S. En conséquence, les matrices T_0 et D'_0 qui ont été nettoyées deviennent T et D'. On peut remarquer que k ($\leq$m) est le nombre de dimensions choisies pour le modèle réduit.

On obtient ainsi un nouvel espace vectoriel : $X = T_0 S_0 D_0^{'} \approx \hat{X} = TSD'$

Ce modèle permet donc de représenter les documents dans un espace de dimension k. Il permet, de façon symétrique, de représenter les termes des vecteurs qui sont une indication du profil de co-occurrence du terme dans les documents. Cette propriété peut être utilisée pour établir une notion de similarité entre termes, ou représenter des documents comme moyenne des vecteurs représentant les termes qu'ils contiennent **[Besançon, 2002]**.

Ce modèle a montré des performances très intéressantes. Pour des corpus de petit et de moyenne taille, la performance est très supérieure au modèle vectoriel classique. D'après les conclusions des systèmes qui ont expérimenté le modèle LSI **[Deerwester et al. 1990]** et **[Furnas et al. 1988]**, quand la taille du corpus augmente, la différence avec les autres modèles semble diminuer.

I.5.1.3.b) Le Modèle DSIR [Besançon 2002] :

Le modèle DSIR (Distributional Semantics based Information Retrieval) ou Recherche documentaire à base de sémantique distributionnelle, est un modèle qui intègre la représentation vectorielle des documents et de leurs connaissances sémantiques. Le modèle DSIR intègre des données de fréquences de co-occurrence[1] et d'occurrence entre les termes, extraites automatiquement à partir des corpus de textes. L'utilisation de co-occurrence de mots pour représenter les liens sémantiques entre ces mots repose sur la notion de sémantique distributionnelle.

Représentation du modèle :

L'hypothèse faite est que dans le cadre de la sémantique distributionnelle **[Besançon 2002]**, on suppose l'existence d'une forte corrélation entre les caractéristiques distributionnelles observables des mots et leurs sens : la sémantique d'un mot est reliée à l'ensemble des contextes dans lesquels il apparaît. L'hypothèse principale pour la sémantique distributionnelle est de considérer que ces contextes apportent suffisamment d'informations pour identifier le sens du mot. La démarche permettant d'aboutir à ce résultat est décomposée en trois étapes:

[1] La co-occurrence est la relation entre des termes qui peuvent partager partiellement un concept.

- Définition du contexte d'un mot dans un corpus, qui permet d'identifier les mots qui sont considérés comme co-occurrents à un lot donné et qui, selon l'hypothèse de la SD, contribuent à son sens.
- La représentation des mots selon la définition prise en compte pour les contextes.
- La définition d'une mesure de similarité entre les représentations des mots qui est alors identifiée avec la mesure de la similarité entre les contextes.

Afin d'identifier les types de relations sémantiques, plusieurs sortes de contextes peuvent être définis :

- Les contextes positionnels : fenêtres de n mots, si n est petit alors ça favorise les relations de composition.
- Les contextes syntaxiques : les contextes dépendent de la structure syntaxique de l'unité textuelle (grammaire et analyse).
- Les contextes documentaires : les contextes sont définis selon les unités textuelles à l'intérieur d'un document (paragraphe, section, chapitre).

La représentation de ces unités linguistiques se fait à l'aide d'une matrice appelée matrice de co-occurrence (i.e. C). Chaque ligne de cette matrice représente le profil de co-occurrence de l'unité linguistique :

$$C = \begin{pmatrix} c_1 \\ c_2 \\ . \\ . \\ c_{|U|} \end{pmatrix} = \begin{pmatrix} c_{11}c_{12}....c_{1|T|} \\ c_{21}c_{22}....c_{2|T|} \\ . \\ . \\ c_{|U|1}c_{|U|2}..c_{|U||T|} \end{pmatrix}$$

Fonction de représentation :

L'hypothèse de la SD peut être reformulée ainsi : deux unités linguistiques sont sémantiquement similaires si leurs contextes textuels sont similaires. Ce qui permet d'écrire la fonction de représentation du modèle DSIR ainsi :

$$rep_{DS}(d) = \sum_{i=1}^{|U|} w(d,u_i) rep_{DS}([u_i])$$

où la représentation d'une unité linguistique particulière (terme d'indexation) est son profil de co-occurrence, pondérée par un facteur associé à l'importance de cette unité : $rep_{DS}([u_i])= p_i\ c_i$. Les pondérations $w\ (d,u_i)$ et p_i données à chaque unité linguistique u_i sont définies comme dans le modèle vectoriel standard, mais ces pondérations sont calculées pour toutes les unités linguistiques de *U*, et non seulement sur *T*. On a donc :

$$rep_{DS}(d) = \sum_{i=1}^{|U|} w_1(d,u_i) c_i$$

Avec $w_1(d,u_i) = w(d,u_i)\ p_i = w_1(d,u_i)\ w_n(d)\ w_g(u_i)$

La collection de documents est alors représentée par le produit matriciel : D = FC

Où F est une matrice d'occurrence de dimension |D| x |U| comme dans le modèle vectoriel, et C la matrice de co-occurrence de dimension |U| x |T|.

Intégrations des co-occurrences :

Le modèle DSIR n'est pas un modèle d'expansion de requêtes, mais un modèle de représentation. C'est à dire que dans le modèle DSIR, les requêtes et les documents sont représentés de la même manière; l'intégration des co-occurrences change donc la fonction de représentation (rep). Pour prendre un exemple, un modèle d'expansion de requête à base de co-occurrence cherche pour la requête "chat" des documents qui contiennent également "lait", "gouttière" ou "miauler", alors que le modèle DSIR cherche, pour la même requête, les documents qui contiennent des mots qui ont aussi pour contexte les mots "lait", "gouttière" ou "miauler".

Pour **[Besançon 2002]** et **[Besançon 2001]**, la façon de calculer les fréquences de cooccurrence dépend des relations de co-occurrence qui sont prises en compte. L'approche la plus simpliste est de calculer toutes les co-occurrences entre toutes les unités linguistiques sur un corpus de référence. Mais cette position n'est pas suffisante. En effet, les co-occurrences non linguistiquement pertinentes peuvent être prises en compte.

Cette approche repose sur un filtrage négatif des co-occurrences, qui permet d'éliminer certaines co-occurrences non souhaitées. Une autre approche peut être de sélectionner les bonnes co-occurrences à conserver. Ceci en utilisant les résultats d'une analyse syntaxique pour extraire les relations entre les différents mots de la phrase. Par exemple, les relations qui pourraient être données par un analyseur syntaxique :

SUBJ (serpent avaler)
OBJ (avaler proie)
ADJ (proie entier)
NN (serpent boa)

Toutes ces informations sont synthétisées dans un graphe de co-occurrence, après un pré traitement ne gardant que les lemmes des noms, des verbes et des adjectifs.

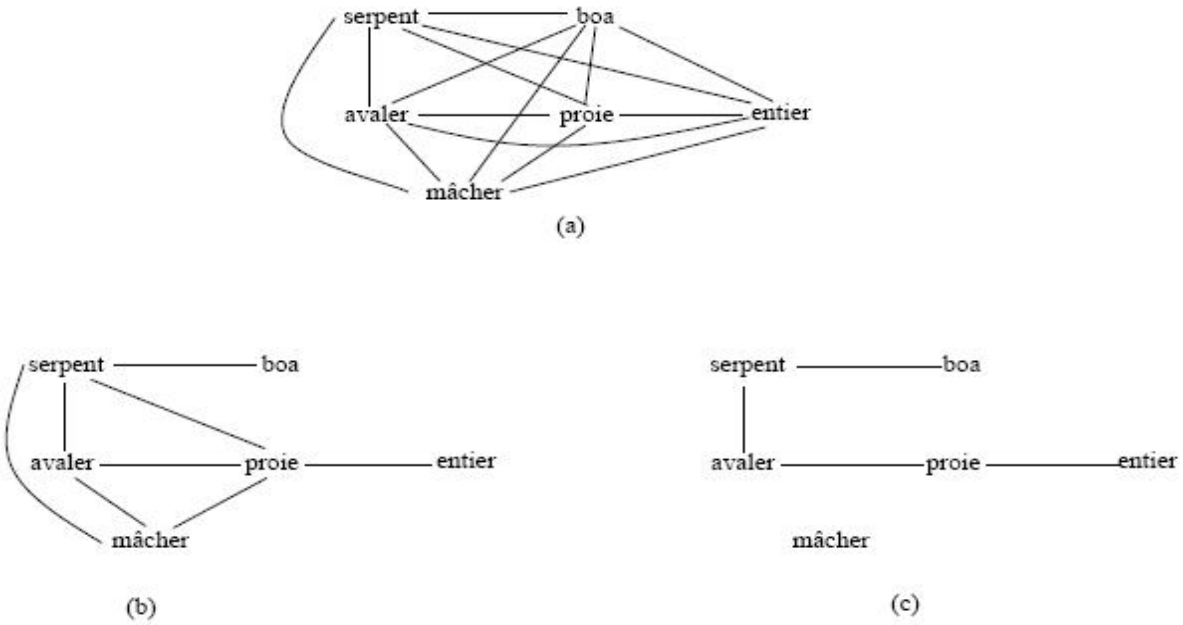

Figure I.1 : *Exemples de graphes de co-occurrences, (a) sans filtrage syntaxique, (b) avec filtrage syntaxique sur les groupes syntaxiques, (c) avec filtrage syntaxique sur les relations syntaxiques.*

Cet exemple permet de montrer que les filtrages syntaxiques des co-occurrences réduisent le nombre de co-occurrences prises en compte, et ainsi éliminent les co-occurrences entre termes non liés syntaxiquement. Néanmoins, on remarque par exemple que le terme "mâcher" n'est relié avec aucun autre terme de la phrase (alors qu'une co-occurrence du type (avaler mâcher) est

suggérée par la phrase). La pratique permet de remarquer que les résultats de l'intégration de connaissances syntaxiques, pour le calcul des co-occurrences pour une tâche de recherche documentaire, montre qu'un filtrage syntaxique sur les groupes syntaxiques par rapport aux co-occurrences permet de réduire le bruit et d'avoir de meilleures performances.

Pour l'évaluation de la pertinence, le modèle DSIR utilise le cosinus comme pour le modèle vectoriel. Dans ce modèle, on tient compte des dépendances entre les termes grâce à l'utilisation des co-occurrences de ces termes.

I.5.2) Modèle logique

On considère qu'un document est jugé pertinent à une requête de l'utilisateur si son contenu sémantique implique logiquement celle-ci. La notion de pertinence, intrinsèque à tout système de recherche d'informations, est alors vue comme une inférence logique. Ce modèle permet de formaliser les paramètres intervenant dans un processus de recherche d'informations et de définir correctement la correspondance entre un document et une requête de l'utilisateur. Il permet aussi de définir la formulation automatique d'une requête, ainsi que la mesure de pertinence associée aux réponses données par le système.

Représentation des documents et requêtes

Le modèle booléen est un exemple simple qui met en oeuvre l'implication logique. Un document est modélisé par une proposition logique formée de la conjonction de ses mots clés. On considère ces mots clés comme des propositions atomiques dans les modèles de la logique des propositions. La requête est une expression logique quelconque. L'idée de base de ce modèle est la suivante: étant donné un document D et une requête Q, D est pertinent vis-à-vis de Q si D implique Q, ce qui se note mathématiquement D$\rightarrow$Q. Prenons l'exemple de documents et d'une requête contenant des termes d'indexations :

$$D_1 = \{A \wedge B\}$$
$$D_2 = \{B \wedge c\}$$
$$D_3 = \{A \wedge B \wedge C\}$$
$$Q = \{A \wedge B \wedge \neg C\}$$

Où A, B, C sont des termes d'indexations. Le document D1 est retrouvé par le système par ce que D1 et l'implication D$\rightarrow$Q est vrai (**Tab I.1**), D2 et D3 ne sont pas retrouvés.

D	Q	D→Q
Faux Faux Vrai Vrai	Faux Vrai Faux Vrai	Vrai Vrai Faux Vrai

Tab I.1 : *Tableau logique de l'implication*

L'implication logique D→Q peut être réécrite ainsi (règle de logique) ¬DvQ

Le terme d'indexation est une partie du texte dans une forme sémantique. Le terme d'indexation est vrai s'il apparaît dans le document. Dans l'exemple ci-dessus, ceci signifie que le document D_1 parle de A et B, et que l'utilisateur recherche un document qui parle de A et B mais pas de C. Le principe du calcul de la correspondance consiste à établir l'implication logique entre D et Q, c'est à dire que le document répond à la requête si D⊃Q est évaluée à Vrai, il faut établir que D⊃Q est une tautologie, et donc que l'on a |=D⊃Q.

I.5.3) Modèle probabiliste

A la différence du modèle logique où on intègre déjà la probabilité de l'implication comme dans le modèle de **[Van Rijsbergen 1986]**, le modèle probabiliste représente la probabilité de la pertinence d'un document D par rapport à une requête Q. Le but de cette fonction de similarité dans ce modèle est d'essayer de séparer les documents pertinents des non pertinents au sein d'une collection. L'idée de base, dans ce modèle probabiliste, est de tenter de déterminer les probabilités P(R/D) et P(NR/D) pour une requête donnée. Cette probabilité signifie : si on retrouve le document D, quelle est la probabilité qu'on obtienne l'information pertinente et non pertinente (**Figure I.2**).

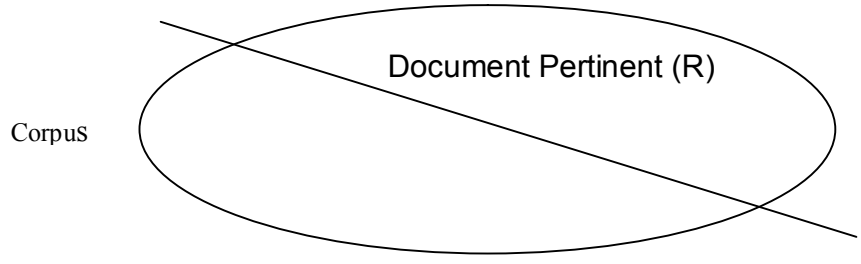

FIG.I.2 : *Corpus pour une requête Q*

Représentation des documents et requêtes

On ne prend en compte que l'absence ou la présence de termes dans les documents et dans les requêtes comme des caractéristiques observables. Ainsi, les termes considérés ne sont pas pondérés, mais prennent seulement les valeurs 0 (absent) ou 1 (présent).

Puisque nous supposons que chaque document est décrit par la présence ou l'absence de termes, n'importe quel document peut être représenté par un vecteur binaire :

$$X=(x_1,x_2,\ldots..,x_n)$$

x_i = 0 ou 1 indique l'absence ou la présence des termes considérés. Nous supposons également qu'il y a deux événements mutuellement exclusifs :

w_1= document pertinent w_2= document non pertinent

Ce que nous désirons calculer pour chaque document est $P(w_1/x)$ et peut-être $P(w_2/x)$ de sorte que nous puissions décider ce qui est pertinent et ce qui est non-pertinent.

Compte tenu de ces probabilités, différentes utilisations en sont faites. Par exemple, la règle de décision qui est employée par **[Van Rijsbergen 1979]** est connue en tant que règle de la décision de Bayes.

$$[P(w_1/x) > P(w_2/x) \rightarrow \text{x pertinent, x est non pertinent}]$$

La signification de $[E \rightarrow p,q]$ est si E est vrai alors on prend p, ou sinon on prend q

I.5.4) Modèle de réseaux de neurones

Les scientifiques en neurologie ont étudié le cerveau humain depuis de nombreuses années, ils ont pu constater qu'il était constitué d'un grand nombre (billions) de cellules de différents types appelés neurones. Chaque neurone est connecté à plusieurs autres neurones par des liaisons que l'on nomme liaisons synaptiques. Un neurone est stimulé par la réception d'un signal. Par le biais d'une réaction automatique, il émet un signal de sortie destiné aux autres neurones avec lesquels il a une liaison.

Représentation des documents et requêtes

Ce symbolisme est un graphe simplifié représentant les connexions entre les neurones d'un cerveau humain. Les noeuds de ce graphe sont des unités processus et les liens entre ces unités jouent le rôle des connections synaptiques. Pour simuler le fait que la force (l'intensité du signal) d'une connexion synaptique dans le cerveau humain change continuellement, un poids est affecté à chaque connexion de notre réseau de neurone. A chaque instant, l'état des noeuds est définit par un niveau d'activation (par l'utilisation d'une fonction ayant pour paramètre l'état initial et le signal reçu par le neurone). Dépendant du niveau d'activation, un noeud A peut envoyer un signal à un noeud voisin B. L'intensité du signal reçu par le noeud B dépend du poids qui est affecté au lien synaptique entre les noeuds A et B.

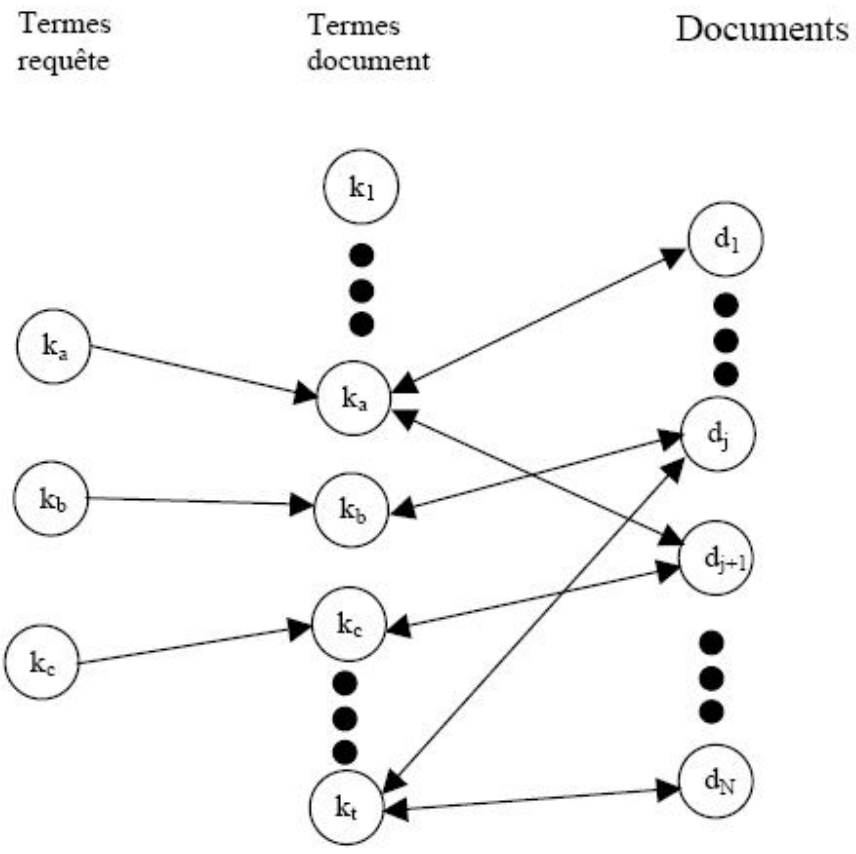

Figure I.3 : *Modèle de réseau de neurones pour la RI* ***[Baeza-Yates et Ribeiro-Neto 1999]***

Un réseau de neurones pour la recherche d'information peut être illustré comme dans la figure (**Figure I.3)**. On utilise le modèle de réseau de neurones pour symboliser la relation entre un document de la collection et les termes qu'il contient, ainsi que les termes de la requête. On remarque que le réseau est composé de trois couches qui sont elles même composées de noeuds : une couche contenant les noeuds qui représentent les termes de la requête, une couche qui contient les noeuds représentant les termes des documents et la troisième couche qui contient les noeuds qui représentent les documents eux-même. Selon **[Baeza-Yates et Ribeiro-Neto 1999]** il est possible d'observer une similarité entre les topologies du réseau de neurones, le réseau d'inférence et le réseau "de croyance".

Les noeuds documents sont activés par l'ensemble de noeuds terme-document, qui eux aussi sont activés par les noeuds correspondant au terme de la requête.

Conclusion

Dans ce chapitre, nous avons introduit la notion de la similarité textuelle ainsi les différents modèle de représentation de textes permettant de calculer cette similarité sémantique entre les documents. Nous avons présenté, au premier temps, le modèle vectoriel standard et ses variantes (LSI, DSIR), qui tentent de prendre en compte la structure sémantique des unités linguistiques, puis les modèles logiques ,réseaux de neurone et le modèle probabiliste, ce dernier est conçu spécifiquement pour la recherche documentaire, dont la représentation des documents dépond des requêtes.

Chapitre II
Les algorithmes de classification par apprentissage

II) L'apprentissage automatique

II.1) Introduction :

La classification, Intuitivement, il s'agit de déterminer la catégorie (que l'on appellera aussi classe) à laquelle un objet appartient. La croissance impressionnante que le réseau Internet a connue ces dix dernières années a renforcé le besoin de techniques facilitant l'accès à l'information. L'ensemble de ces techniques sont reprises dans ce que l'on appelle l'*Information Access*. La classification automatique en est une des méthodes principales. Un programme qui réalise de la classification automatique est appelé classificateur. Les classificateurs sont utilisés dans bien d'autres contextes que celui de l'*Information Access*.

En termes d'action, le fait de classer un objet correspond à prendre une décision sur base d'une ou plusieurs règles. Dès lors une des premières approches pour automatiser le traitement, fut d'extraire la connaissance experte de spécialistes sous formes de règles. Ainsi, pour chaque catégorie on disposait d'un ensemble de règles permettant de déterminer l'appartenance d'un objet à la catégorie. Bien que cette façon de faire ait été largement utilisée jusqu'à la fin des années 80, on en conçoit aisément les limites. En effet, quand le nombre de documents et de catégories est très important, il devient impossible de rassembler toutes les connaissances expertes nécessaires aux prises de décision. Dans les années 90, l'approche *Machine Learning* (ML) devient très populaire. En bref, il s'agit d'apprendre automatiquement les règles de décision sur base d'un ensemble d'objets pré-classés. Il s'agit donc d'un processus inductif suivant lequel un classificateur est construit à partir d'exemples

II.2) Généralités :

La résolution de problèmes par la construction de machines capables d'apprendre à partir d'expériences caractérise l'approche fondamentale du Machine Learning (ML). Parmi les techniques du ML on trouve entre autres les méthodes supervisées et non-supervisées. Dans le cas des méthodes supervisées, on cherche à estimer (à apprendre) une fonction f(x) à partir d'exemples.

Les méthodes de classification font partie des techniques de ML supervisées. Dans ce cadre, un exemple (donnée) consiste en un objet ou sa représentation (un document, une image. .) accompagné de la catégorie à laquelle il appartient. La fonction f(x) que l'on va chercher à

estimer, détermine la relation entre les objets et leur catégorie. Un classificateur sera donc caractérisé par la fonction f(x) qu'il implémente. On dira qu'un classificateur a un grand pouvoir de généralisation s'il fournit de bons résultats sur des données différentes de celles qui ont été utilisées pour l'apprentissage.

Dans les techniques non-supervisées, les objets sont présentés sans leur catégorie. Un exemple type de méthodes non-supervisées est le *clustering* dont le but est de créer des groupes (*clusters*) d'objets présentant des caractéristiques semblables.

II.3) Formulation de la classification en théorie :

La tâche qu'un classificateur doit effectuer peut être exprimée par une fonction que l'on appelle fonction de décision : $f : X \rightarrow Y$

Où X est l'ensemble des objets à classer (aussi appelé espace d'entrée) Y est l'ensemble des catégories (aussi appelé espace d'arrivée)

Dans le cas de la classification binaire, l'ensemble Y correspond à $\{+1, -1\}$. Une grande partie de la littérature se focalise sur le cas binaire parce que les classifications faisant intervenir plus de 2 catégories (cas multi-classes) peuvent toujours être ré-exprimées sous forme de classifications binaires. En effet, si l'on considère une classification multi-classes, nous pouvons effectuer, pour chaque catégorie, une classification binaire indépendante et regarder ensuite pour quelle catégorie l'appartenance s'est manifestée.

II.4) Algorithmes de classification

Nous allons présenter trois algorithmes d'apprentissage et de classification. Pour concrétiser le traitement, nous nous positionnons dans le cadre de la classification de documents textuels représentés avec *bag of words*. Bien que ces algorithmes soient relativement simples, ils fournissent des résultats assez satisfaisants **[Fabrizio Sebastiani,99]**.

II.4.1) Naïve Bayes

Il s'agit d'un classificateur probabiliste qui se base sur le théorème de Bayes pour prédire la classe à laquelle un document appartient :

$$P\{Y = y_i | X = x\} = \frac{P\{X = x | Y = y_i\} * P\{Y = y_i\}}{\sum_i P\{X = x | Y = y_i\} * P\{Y = y_i\}}$$

L'idée consiste à classer un texte suivant la catégorie pour laquelle cette probabilité est la plus élevée.

$$y = \arg\max_{(y_i)} (P\{X = x \mid Y = y_i\} * P\{Y = y_i\}) \quad \textbf{(3.1)}$$

Remarquons que le dénominateur du théorème de Bayes est identique pour toutes les catégories. Par conséquent, nous devons uniquement considérer le numérateur.

Pour calculer (1), noua avons besoins des deux quantités suivantes :

1- $P(Y=y_i)$
2- $P(X=x|Y=y_i)$

On parle souvent de probabilité a priori pour la première quantité. On peut l'estimer en calculant la proportion de documents du *training set* appartenant à la classe y_i (méthode du maximum de vraissemblance).

$$P\{Y = y_i\} = \frac{\{(x, y) \in Tr | Y = y_i\}}{|Tr|}$$

Observons à présent la seconde quantité. Lorsque l'on parle de P{X = x}, cela se réfère à la probabilité d'avoir un document tel que sa représentation vectorielle soit $(w_1, w_2, \ldots, w_{|T|})$. Nous supposons que les textes nous sont fournis dans le format bag of words et que les poids sont binaires (0 : aucune occurrence, 1 : au moins une occurrence).

Réexprimons la seconde expression par la *chain rule* :

$$P\{X = x_i | Y = y_i\} = P\{w_1, w_2, \ldots, w_{|T|} | Y = y_i\}$$

$$= P\{w_1 | w_2, \ldots, w_{|T|}, Y = y_i\} \times P\{w_2 | w_3, \ldots, w_{|T|} | Y = y_i\} \times \ldots \times P\{w_{|T|} | Y = y_i\}$$

L'aspect naïf de ce genre de classificateur provient du fait que l'on considère que les événements w_{jx} sont indépendants. Par conséquent, la seconde quantité devient :

$$P\{X = x_i | Y = y_i\} = P\{w_1 | Y = y_i\} \times P\{w_2 | Y = y_i\} \times \ldots \times P\{w_{|T|} | Y = y_i\} \quad \textbf{(3.2)}$$

En pratique, lorsque l'on classe un nouveau document, on ne tient compte que des w_l non-nuls dans l'expression de P {X = x | Y = y_i}. La raison est qu'il y a généralement nettement moins de termes dans un document que dans notre lexique. Pour calculer P {w_l | Y = y_i}, il suffit de compter le nombre d'occurences du terme *l* dans l'ensemble des documents classés sous y_i et de diviser cette quantité par la somme des occurences de tous les termes dans cette même catégorie.

Pour récapituler ceci, un entraînement avec Naïve Bayes consiste à calculer les quantités suivantes à partir du *training set* (l'ensemble d'apprentissage) :

- P(Y=y_i) : Un tableau contenant |*Y*| entrées
- P(*w*|Y=y_i) : Une matrice contenant |*Tr*| x |*Y*| entrées

A partir de ces structures de données, nous pouvons classer de nouveaux documents en utilisant les formules (3.1) et (3.2). Nous proposons ci-dessous deux pseudo-algorithmes reprenant l'apprentissage (Alg. 1) et la classification (Alg. 2) de textes par Naïve Bayes.

Alg.1 LEARN NAIVE BAYES (Tr, categories)

{Création du lexique}
Lexique ← ensemble des termes distincts apparaissant dans Tr
{Calcul des P {Y = y_i} et des P {w_l | Y = y_i}}
for $y_i \in$ *catégories* **do**
  $docs_i \in$ *{(x, y) ∈ Tr | y = y_i}se trouvant*
  P {Y = yi} ← (|$docs_i$| / |Tr|)
  $Text_i$ *← la somme vectorielle des documents ∈* $docs_i$
  n ← la somme des composantes de $Text_i$
  for *l = 1..|lexique|* **do**
    n_l *← $Text_i$[l]*
    P {w_l | Y = y_i} ← (n_l+1) / (n+|lexique|)
  end for
end for

Alg. 2 CLASSIFY NAIVE BAYES (x, categories)

Max_cat ← *none*
max ← *0*
for $y_i \in$ *categories* **do**
  prod ← *1*
  for *l = 1..|lexique|* **do**
  prod ← *prod* × $P\{w_l \mid Y = y_i\}$
end for
prod ← *prod* ×$P\{Y = y_i\}$
if *prod* > *max* **then**
  max ← *prod*
  max_cat ← y_i
end if
end for
return *max_cat*

II.4.2) Arbres de décision (AD)

Les arbres de décision sont une des techniques les plus populaires du ML supervisé. Cette méthode est très facile à mettre en oeuvre, de plus, on peut facilement interpréter les règles de décision issues de l'apprentissage. Il existe plusieurs versions des AD dont les plus connues sont ID3 **[J. R. Quinlan,1993]** et C4.5 [**J. R. Quinlan,1996**]. Nous présentons ici la version la plus simple : ID3. Nous considérons à nouveau la classification de textes représentés par *bags of word* avec des poids binaires.

Un AD est constitué, comme tout arbre, de noeuds et de feuilles. Chaque nœud contient un test et possède autant de descendants qu'il y a de valeurs possibles pour ce test. Dans notre cas, les tests consisteront simplement à vérifier la présence d'un terme dans le document analysé. On aura donc affaire à des arbres binaires. Aux extrémités des branches de l'arbre, on trouve des feuilles qui indiquent le résultat de la classification, c.à.d la classe que l'on assigne. Pour classer un nouveau document, il suffit de le soumettre à la racine de l'arbre et de le laisser parcourir les branches au fil des résultats des tests jusqu'à atteindre une feuille.

Dans le cadre des AD, l'entraînement consiste à créer l'arbre à partir du *training set*. Il s'agit d'un processus récursif dans lequel les données d'entraînement seront utilisées pour déterminer

l'attribut à tester lors de la création des noeuds. Au départ, on dispose d'un ensemble de documents correspondant au training set complet. Sur base de cet ensemble, on va déterminer un attribut (dans notre cas, il s'agit de l'indice d'un terme) sur lequel le test de la racine de l'arbre va porter. Notre ensemble va ensuite être divisé en deux sous-ensembles contenant respectivement les données ayant et n'ayant pas satisfait au test. Le processus est alors répété sur ces deux sous-ensembles. Lorsque l'on ne peut plus trouver d'attribut permettant de subdiviser l'ensemble, une feuille portant le label majoritairement représenté par les exemples que l'on a reçus, est créée.

Il existe plusieurs manières de déterminer les tests à réaliser dans les noeuds. L'idée commune est de créer des tests qui discriminent le plus possible les exemples d'apprentissage. En clair, cela veut dire que l'attribut sur lequel on fait le test doit séparer les exemples en deux ensembles de taille voisine. Dans ID3, on se sert d'un critère basé sur l'entropie, notion issue de la théorie de l'information. L'entropie du *training set* se définit comme suit :

$$Entropie(Tr) = \sum_{y \in Y} P(Y = y) \log_2 P(Y = y)$$

L'entropie permet de mesurer l'homogénéité des exemples. Si l'entropie vaut 0, tous les exemples appartiennent à la même catégorie, alors que si elle vaut 1, il y a autant d'exemples positifs que négatifs. Notons par ailleurs que dans ce contexte on définit 0 log 0 = 0.

L'entropie est utilisée pour formuler une mesure appelée Information Gain (IG) :

$$IG(Tr, j) = Entropie(Tr) - \left(\frac{|Tr_{j,1}|}{|Tr|} * Entropie(Tr_{j,1}) \right) - \left(\frac{|Tr_{j,0}|}{|Tr|} * Entropie(Tr_{j,0}) \right)$$

Où $Tr_{j,k}=\{(x,y) \in Tr \mid w_j=k\}$ et j est l'index de l'attribut (le terme) pour lequel le gain d'information est calculé.

Cette mesure est d'autant plus grande que l'attribut j est discriminant. En conséquence de quoi, lors de la création des noeuds, l'index de l'attribut sur lequel portera le test sera déterminé de la façon suivante :

$$best = \arg\max_j (IG(Tr', j))$$

Où Tr′ est le sous-ensemble des exemples du *training set* ayant satisfait aux tests menant au noeud courant.

Nous proposons le pseudo-algorithme (Alg. 3) qui reprend la construction d'un arbre binaire par la méthode ID3. Lors de la première invocation, les paramètres sont (Tr,null,{1, 2, . . . $|T|$}).

Alg.3 LEARN_ID3(*Tr',Branche,Attributs*)

Créer un nouveau noeud
if *Branche* <> *null* **then**
  Attacher le noeud à la Branche
end if
if $\forall (x, y) \in Tr : y = i$ **then**
  noeud ←*FEUILLE_i*
else
  $best \leftarrow \arg\max_{j} \left(IG\left(Tr', j\right)\right)$
  noeud.test_attr ← *best*
  for $k \in \{true, false\}$ **do**
    Ajouter une nouvelle branche correspondant au test $w_{best} == k$
    $Tr_{best,k} \{(x, y) \in Tr \mid w_{best} = k\}$
    if $Tr_{best,k} = \emptyset$ **then**
      Ajouter une feuille à la branche avec le label majoritaire dans Tr'
    else
      LEARN_ID3($Tr_{best,k}$ *, Branche , Attributs \ {best})*
    end if
  end for
end if
return *noeud*

II.5) Machines à Vecteurs Supports (SVM):

SVM est une méthode de classification qui fut introduite par **[Vapnik, 1995]**. Le succès de cette méthode est justifié par les solides bases théoriques qui la soutiennent. Il existe en effet un lien direct entre la théorie de l'apprentissage statistique et l'algorithme d'apprentissage de SVM. La plupart des techniques du ML possèdent un (trop) grand nombre de paramètres d'apprentissage à fixer par l'utilisateur (structure d'un réseau de neurones, coefficient de mise à jour du gradient, . . .). De plus, avec ces méthodes, le nombre de paramètres à calculer par l'algorithme d'apprentissage est en relation linéaire, voire exponentielle, avec la dimension de l'espace d'entrée. La formulation élégante de SVM laisse très peu de place aux paramètres utilisateurs et le nombre de paramètres est linéaire en la taille du *training set*. SVM est donc une méthode de classification particulièrement bien adaptée pour traiter des données de très haute dimension telles que les textes et les images. Dans la suite nous présentons les aspects théoriques de la méthode SVM.

II.5.1) Qu'est-ce qu'un SVM?

Une SVM est un algorithme d'apprentissage, permettant d'apprendre un séparateur. Ceci ramène le problème à savoir ce qu'est un séparateur. Donnons-nous un ensemble fini de vecteurs de R^n, séparés en deux groupes, ou dit autrement en deux classes. L'appartenance à un groupe ou un autre est défini par une étiquette, associée à chacun des vecteurs, sur laquelle est inscrite « groupe 1 » ou « groupe 2 ». Trouver un séparateur revient à construire une fonction, qui prend un vecteur de notre ensemble, et peut dire de quel groupe il est. Les SVM sont une solution à ce problème, comme le serait un simple apprentissage par coeur des classes associées aux vecteurs de notre ensemble. Mais avec les SVM, on attend de bonne propriétés de généralisation, à savoir que si un nouveau vecteur de présente, qui n'était pas dans l'ensemble, la SVM saura dire à quel groupe il est vraisemblable qu'il appartient, au regard des attributions de classes des vecteurs présents au départ.

L'idée est de poser, à partir des vecteurs dont on connaît les classes, un problème d'optimisation, du style « optimiser telle grandeur en s'assurant que ... ». Il y a deux difficultés. La première, c'est poser le bon problème d'optimisation. Cette notion de « bon » problème est celle qui fait référence aux théories mathématiques de la généralisation, et fait des SVM un outil d'un abord parfois difficile. La deuxième difficulté est de résoudre ce problème d'optimisation

une fois posé, et là on tombe plus dans des subtilités informatiques, dont nous retiendrons l'algorithme SMO.

II.5.2) Classificateur linéaire

II.5.2.1) Définition

Un classificateur est dit linéaire lorsqu'il est possible d'exprimer sa fonction de décision par une fonction linéaire en x qui désignera un vecteur de R^n. où n est le nombre de composantes des vecteurs contenant les données. On peut, en toute généralité, exprimer une telle fonction comme ceci : $f(x) = \langle w, x \rangle + b = \sum_{i=1}^{n} w_i x_i + b$

Où $w \in R^n$ et $b \in R$ sont des paramètres, et $x \in R^n$ est une variable.

Ce classificateur ne fournit pas des valeurs valant exclusivement -1 *(classe 1)* ou 1 *(classe 2*), mais nous dirons que quand le résultat $f_{w,b}(x)$ est positif, le vecteur x appartient à la même classe que les exemples d'étiquette 1, et que quand ce résultat est négatif, le vecteur x appartient à la même classe que les exemples d'étiquette -1.

Notons que l'équation $f_{w,b}(x)=0$ définit la frontière de séparation entre les deux classes, et que cette frontière est un hyperplan affine dans le cas du séparateur linéaire. En orientant l'hyperplan (C.à.d, en fixant un côté pour lequel les exemples sont classés positivement), la règle de décision correspond à observer de quel coté de l'hyperplan se trouve l'exemple x. La figure 4.1 représente la situation dans R^2. On voit que le vecteur w définit la pente de l'hyperplan : w est perpendiculaire à l'hyperplan. Le terme b quant à lui permet de translater l'hyperplan parallèlement à lui-même.

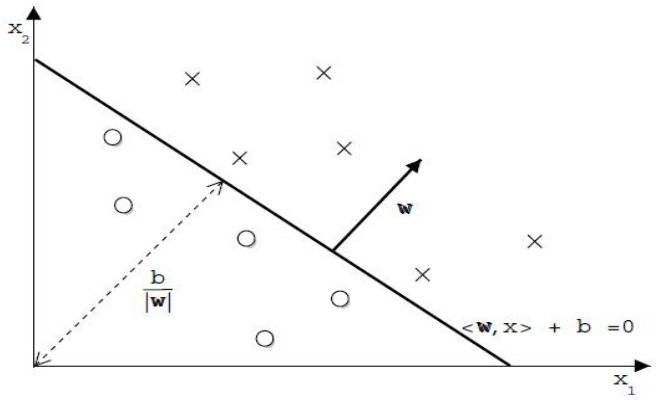

FIG.II.1 : *Représentation dans R^2 de l'hyperplan correspondant à la fonction de décision d'un classificateur linéaire*

II.5.3) Marge de l'hyperplan

II.5.3.1) La séparabilité

Soit S= $\{(x_i, y_i) \in R^n \text{ x } \{-1,1\}\}$, i=1,…, n, l'ensemble des données d'apprentissage (*training set*) avec leur étiquette $y_i \in \{-1,1\}$, $S^+ = \{x/ (x,y) \in S \text{ et } y=1\}$ et $S^- = \{x/ (x,y) \in S \text{ et } y=-1\}$. Dire que S est linéairement séparable signifie qu'il existe $w \in R^n$ et b un réel tel que :

$$f_{w,b}(x) = \langle w, x \rangle + b > 0 \qquad \forall x \in S^+$$

$$f_{w,b}(x) = \langle w, x \rangle + b < 0 \qquad \forall x \in S^-$$

Autrement dit, un *training set* S est linéairement séparable SSI :

$$\exists w \in R^n, b \in R : y_i(\langle w, x_i \rangle + b) \geq 0 \qquad \forall i = 1...n$$

La définition consiste à dire qu'il doit exister un hyperplan laissant d'un coté toutes les données positives et de l'autre, toutes les données négatives. Dans le cas de données linéairement séparables, il existe plusieurs méthodes pour trouver un tel hyperplan. La première d'entre elles et la plus connue est l'algorithme du perceptron de **[Rosenblatt, 1958]**.

II.5.3.2) Marge

Supposons que S soit linéairement séparable pour ce qui suit. C'est une hypothèse forte, que l'on réduira par la suite, mais qui va nous permettre de poser quelques notions. L'idée au coeur des SVM est que, certes, il convient de séparer les exemples de chaque classe, mais qu'il faut que l'hyperplan passe « bien au milieu ». C'est pour définir cette notion de « bien au milieu » (**FIG.II.2**) que l'on introduit la marge.

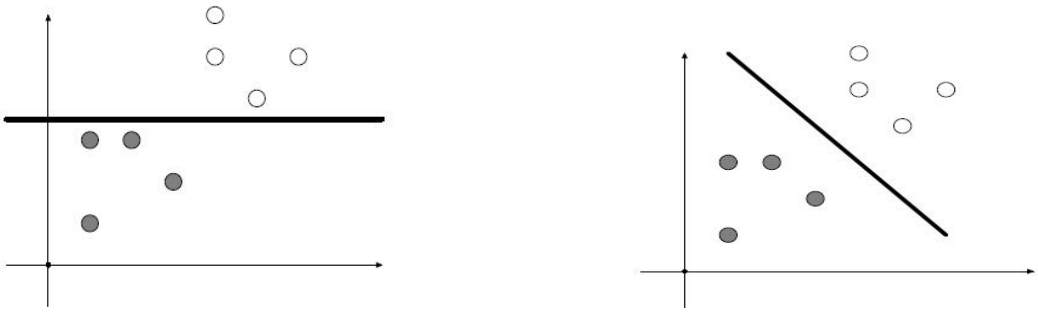

FIG.II.2 : *Les mêmes exemples sont à gauche comme à droite, séparés par une droite.*

Aidons nous de la figure II.5.3.2.b pour faire les observations suivantes. Notons déjà que les courbes d'équation $f_{w,b}(x) = C$ sont des hyperplans parallèles, et que w est normal à ces hyperplans. Le paramètre b traduit un décalage de l'hyperplan séparateur, soit une translation des valeurs de $f_{w,b}$. La norme $\|w\|$ de w influence les courbes de niveau $f_{w,b}$=C. Plus la $\|w\|$ est élevée, plus les courbes de niveau sont serrées.

Si l'on souhaite une frontière séparatrice donnée, on se trouve confronté à une indétermination pour le choix de w et de b. N'importe quel vecteur w non nul perpendiculaire à l'hyperplan convient. Une fois choisi, on détermine b tel que b/ $\|w\|$ soit la mesure orientée[2] de la distance de l'origine au plan de séparation.

La notion de marge peut être relative à un exemple particulier ou à l'ensemble du *training set*. De plus, on considère deux types de marges : fonctionnelle et géométrique.

[2] La direction de *w* donne le sens positif

II.5.3.2.a) Marge fonctionnelle d'un exemple

La marge fonctionnelle d'un exemple x_i, par rapport à l'hyperplan caractérisé par w et b est la quantité : $y_i\left(\langle w, x_i\rangle + b\right)$

II.5.3.2.b) Marge géométrique d'un exemple

La marge géométrique quant à elle représente la distance euclidienne prise perpendiculairement entre l'hyperplan et l'exemple x_i, donc par rapport à l'hyperplan caractérisé par w et b la marge géométrique est définie comme étant :

$$\textbf{(3.b)} \quad \Psi_{w,b}(x_i, y_i) = y_i\left(\frac{w}{\|w\|}x_i + \frac{b}{\|w\|}\right)$$

Remarquons que la marge d'un exemple mal classé correspond à la distance négative qui le sépare de l'hyperplan.

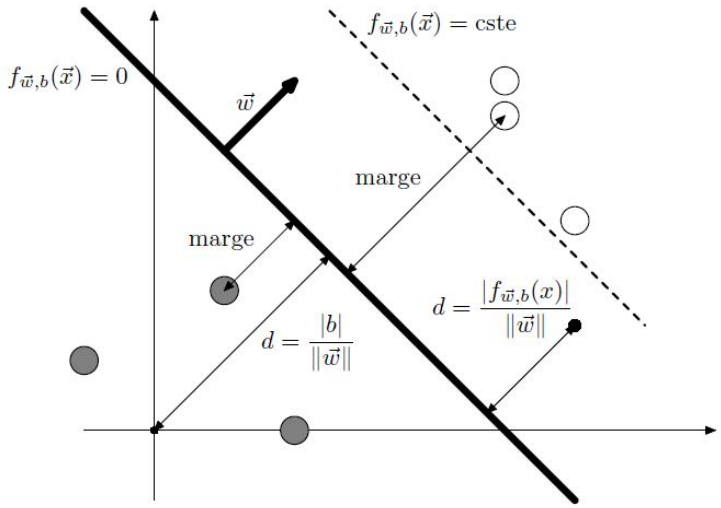

FIG.II.3 : *Définition d'un séparateur $f_{w,b}$*

II.5.3.2.c) Marge géométrique d'un taraining set

La marge du training set par rapport à l'hyperplan caractérisé par w et b est définie comme étant :

$$\Psi_{w,b} = \min_{i=1\ldots n} \Psi_{w,b}(x_i, y_i) \qquad \textbf{(3.c)}$$

Par la suite la marge utilisée est uniquement la marge géométrique notée par le terme “marge” pour désigner cette dernière. Les classificateurs ayant pour critère de maximiser la marge du *training set* sont appelés classificateurs à marge maximale. SVM, en particulier, en est un.

II.5.3.3) L’hyperplan canonique

Dans le cadre de classificateurs à marge maximale, l’hyperplan séparateur correspond à la médiatrice du plus petit segment de droite reliant les enveloppes convexes des deux catégories. Notons que l’on suppose aussi que le *training set* est linéairement séparable. Dès lors, on peut définir deux plans se trouvant de part et d’autre de l’hyperplan et parallèles à celui-ci, sur lesquels reposent les exemples le plus proches. La figure (**FIG.II.4)** illustre cette situation.

Il est donc possible de redimensionner w et b de telle sorte que les deux plans parallèles aient respectivement pour équation : $\langle w, x \rangle + b = 1$ et $\langle w, x \rangle + b = -1$

Ces deux hyperplans sont appelés hyperplans canoniques. Notons que la marge des hyperplans canoniques est $1/\|w\|$. Le vecteur w possède à présent une signification géométrique trés claire.

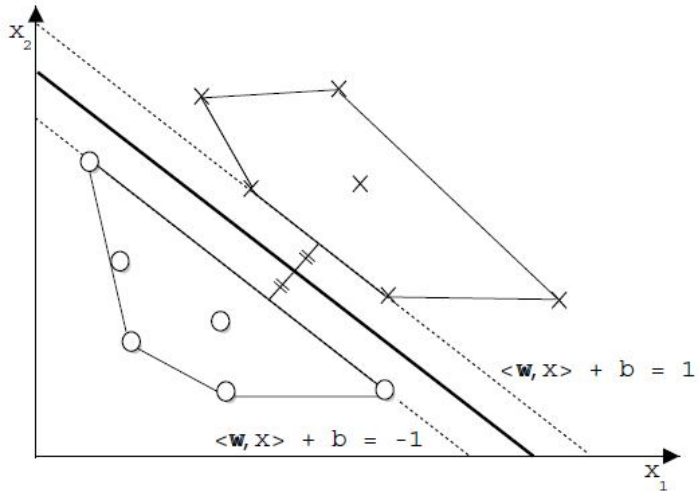

FIG.II.4 : *Hyperplans Canoniques*

II.5.4) Un problème d'optimisation

II.5.4.1) Le problème à résoudre par la SVM

II.5.4.1.1) Cas séparable

Supposant toujours que l'on dispose d'un *training set S* effectivement séparable par un séparateur linéaire. Si le séparateur sépare effectivement *S*, avec du côté positif les exemples étiquetés 1, et du côté négatif ceux étiquetés -1, alors toutes les marges des exemples sont positives (eq 3.b). Si l'une des marges est négative, c'est que le séparateur ne sépare pas correctement les deux classes, alors que c'est possible, vu qu'on suppose *S* linéairement séparable. Dans ce cas incorrect, la marge est négatif (eq 3.c). Donc maximiser la marge, veut bien dire d'abord qu'on sépare (marge positive), puis qu'on sépare bien (marge maximale).

Pour chaque séparateur, les exemples du *training set* les plus proches (eq 3.c) se trouvent sur les hyperplans canoniques de celui-ci, pour le séparateur à marge maximale ces exemples ont une marge plus grande que les exemples de plus petite marge des autres séparateurs possibles et sont appelés *vecteurs support*. En rappelle la distance qui sépare les vecteurs supports au plan de séparation, la marge donc, est $1/\|w\|$.

La largeur de la bande constituée par les hyperplans $\langle w,x\rangle + b = 1$ et $\langle w,x\rangle + b = -1$ est $2/\|w\|$. Pour trouver le séparateur de marge maximale, il suffit alors de chercher, parmi les séparateurs vérifiant pour tous les exemples $y*f(x) > 1$, le séparateur pour lequel $\|w\|$ est minimal.

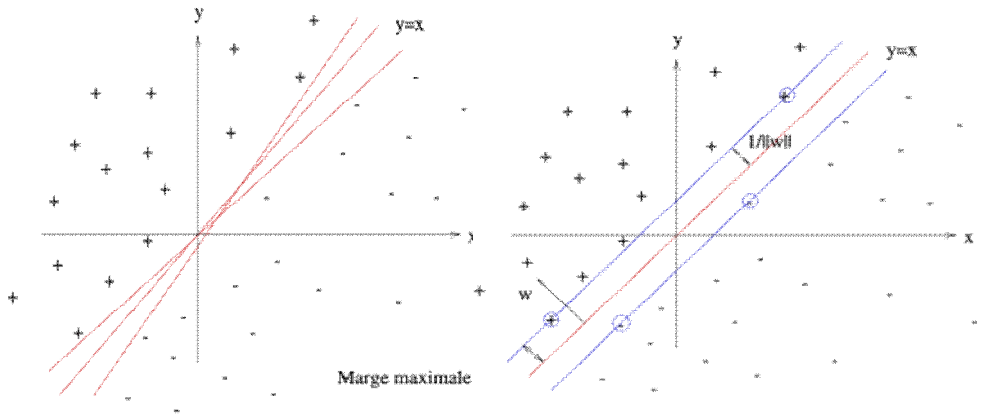

FIG.II.5 : *Infinité d'hyperplans séparateurs, et l'hyperplan optimal avec marge maximale, Les échantillons entourés sont des vecteurs supports.*

Maintenant, nous pouvons formuler un problème d'optimisation mathématique tel que sa solution nous fournisse l'hyperplan optimal (maximisant la marge) :

(QP1) Minimiser $W(w,b) = \frac{1}{2}\|x\|^2$

Tel que $y_i\left(\langle w, x_i \rangle + b\right) \geq 1$

Il s'agit d'un problème quadratique dont la fonction objective est à minimiser. Cette fonction objective est le carré de l'inverse de la double marge. L'unique contrainte stipule que les exemples doivent être bien classés et qu'ils ne dépassent pas les hyperplans canoniques.

Dans cette formulation, les variables à fixer sont les composantes *w* et *b*. D'un point de vue ML, ces variables correspondent aux α_i de la machine d'apprentissage. Le vecteur *w* possède un nombre de composantes égal à la dimension de l'espace d'entrée. En gardant cette formulation telle quelle, nous souffrons des même problèmes que les méthodes classiques du ML (*overfitting, curve of dimensionality*). Pour éviter cela, il est nécessaire d'introduire une formulation dite duale du problème. Un problème dual est un problème fournissant la même solution que le primal mais dont la formulation est différente. On appellera *variables primales*, les variables du problème primal, et *variables duales*, les variables du problème dual qui n'interviennent pas dans le primal.

Pour dualiser (QP1), nous devons former ce que l'on appelle le Lagrangien. Il s'agit de faire rentrer les contraintes dans la fonction objective et de pondérer chacune d'entre elles par une variable duale :

$$L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i\left(\langle w, x_i \rangle + b\right) - 1 \right]$$

Les variables duales α_i intervenant dans le Lagrangien sont appelées *multiplicateurs de Lagrange*. Selon le type de contraintes qu'ils représentent, les multiplicateurs doivent respecter les règles de signes suivantes :

1- contrainte du type $C_i(x) \geq 0$: $\alpha_i \geq 0$
2- contrainte du type $C_i(x) = 0$: α_i est sans restriction de signe
3- contrainte du type $C_i(x) \leq 0$: $\alpha_i \leq 0$

En pratique, pour le dernier cas on considère des multiplicateurs positifs mais ces derniers interviennent avec un signe "−" dans le Lagrangien. On peut donner une signification physique à ces multiplicateurs : la variables α_i représente la "force" avec laquelle la solution appuie sur la contrainte i. Ainsi, un hyperplan qui violerait la contrainte pour x_i (il classe cet exemple du mauvais côté) rendrait α_i très grand ce qui ferait fortement augmenter la fonction objective (L). Cette solution ne pourrait donc pas être retenue comme solution optimale. Notons que L doit être minimisé par rapport aux variables primales et maximisé par rapport aux variables duals[3].

A présent, nous introduisons les conditions *Karush Kuhn et Tucker* (KKT) statuant sur l'optimalité d'une solution.

Théorème (KKT pour les problèmes différentiables convexes [Karush,1939], [Kuhn et Tucker,1951])

Considérons un problème d'optimisation de la forme :

Minimiser $g(x)$ tel que $\begin{cases} c_i(x) \le 0 \forall i = 1..n \\ e_j(x) = 0 \forall j = 1..n' \end{cases}$

Avec $g(.)$, $c_i(x)$ et $e_j(x)$ convexes et différentiables.

Le lagrangien est formé comme suit :

$$L(x,\alpha) = g(x) - \sum_{i=1}^{n} \alpha_i c_i(x) + \sum_{j=1}^{n'} \beta_j e_j(x)$$

La solution $\bar{x}$ est optimale SSI il existe $\bar{\alpha} \in R^n$ avec $\alpha_i \ge 0 \forall i = 1..n$ et $\bar{\beta} \in R^{n'}$ avec β_j s.r.s $\forall j = 1..n'$ tels que :

$$\partial_x L(\bar{x},\bar{\alpha}) = \partial_x g(\bar{x}) - \sum_{i=1}^{n} \alpha_i \partial_x c_i(\bar{x}) + \sum_{j=1}^{n'} \beta_j \partial_x e_j(\bar{x}) = 0$$

$$\partial_{\alpha_i} L(\bar{x},\bar{\alpha}) = c_i(\bar{x}) \le 0$$

[3] En maximisant par rapport aux α_i , on assure qu'un maximum de contraintes soient satisfaites

$\partial_{\beta_j} L(\bar{x},\bar{\alpha}) = e_j(\bar{x}) = 0$

$\bar{\alpha}_i c_i(\bar{x}) = 0 \qquad \forall i = 1...n$

$\bar{\beta}_j e_j(\bar{x}) = 0 \qquad \forall j = 1...n'$

Ce théorème fondamental en optimisation mathématique, nous fournit une condition suffisante et nécessaire pour l'optimalité d'une solution dans le cadre de problèmes différentiables convexes (ce qui est notre cas). Les deux dernières conditions sont souvent appelées *conditions KKT complémentaires*. Ces conditions expriment deux choses. Pour le voir prenons la première, $\bar{\alpha}_i c_i(\bar{x}) = 0$:

1. $\bar{\alpha}_i = 0$, Dans ce cas la solution n'est pas "sur la contrainte". Il n'y a donc rien à imposer au niveau de la solution.

2. $\bar{\alpha}_i \neq 0$, La solution est "sur la contrainte". Dans ce cas, la nullité du produit impose que la solution ne dépasse pas la contrainte (elle reste faisable).

Déterminons à présent les conditions KKT de notre problème d'optimisation (QP1).

$$\partial_w L(w,b,\alpha) = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \qquad (4.1)$$

$$\partial_b L(w,b,\alpha) = \sum_{i=1}^{n} \alpha_i y_i = 0 \qquad (4.2)$$

$$\partial_{\alpha_i} L(w,b,\alpha) = -y_i\left(\langle w,x\rangle + b\right) + 1 \leq 0 \qquad (4.3)$$

$$\alpha_i\left(y_i\left(\langle w,x\rangle + b\right) - 1\right) = 0 \qquad \forall i = 1..n \qquad (4.4)$$

L'équation (4.1) permet de réexprimer *w* :

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \qquad (4.5)$$

Remarquons qu'avec cette formulation, on peut calculer *w* en fixant seulement *n* paramètres. L'idée va donc être de formuler un problème dual dans lequel *w* est remplacé par sa formulation

$$L(w,b,\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(4.5). De cette façon, le nombre de paramètres à fixer est relatif au nombre d'exemples du *training set* et non plus à la dimension de l'espace d'entrée (supposée très élevée). Pour ce faire, nous substituons (4.2) et (4.5) dans le Langrangien :

A partir de quoi nous pouvons formuler le problème dual :

$$(QP2) \qquad \textit{Maximiser} \quad W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

$$\textit{tel que} \quad \begin{cases} \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \alpha_i \geq 0 \qquad \forall i = 1 \ldots n \end{cases}$$

La résolution du dual permet donc de calculer le vecteur *w* à moindre coût, cependant cette formulation ne fait à aucun moment apparaître le terme *b*. Pour calculer ce dernier nous devons utiliser les variables primales :

$$b = -\frac{\max_{y=-1}\left(\langle w, x_i \rangle\right) + \max_{y=+1}\left(\langle w, x_i \rangle\right)}{2}$$

Nous avons à présent tous les éléments nécessaires pour exprimer la fonction de décision de notre classificateur linéaire :

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \langle x, x_i \rangle + b$$

Notons qu'un grand nombre de termes de cette somme sont nuls. En effet seuls les α_i correspondant aux exemples se trouvant sur les hyperplans canoniques ("sur la contrainte") sont non-nuls. Ces exemples sont appelés *Support Vectors* (SV). On peut les voir comme les représentants de leurs catégories car si le *training set* n'était constitué que des SV, l'hyperplan optimal que l'on trouverait serait identique.

II.5.4.1.2) Marge souple

En général, il n'est pas non plus possible de trouver une séparatrice linéaire dans l'espace de redescription. Il se peut aussi que des échantillons soient mal étiquetés, et que l'hyperplan séparateur ne soit pas la meilleure solution au problème de classement.

[Cortes et Vapnik, 1995] proposent une technique dite de marge souple, qui tolère les mauvais classements. La technique cherche un hyperplan séparateur qui minimise le nombre d'erreurs grâce à l'introduction de *variables ressort* ξ_k (*slack variables en anglais*), qui permettent de relâcher les contraintes sur les vecteurs d'apprentissage:

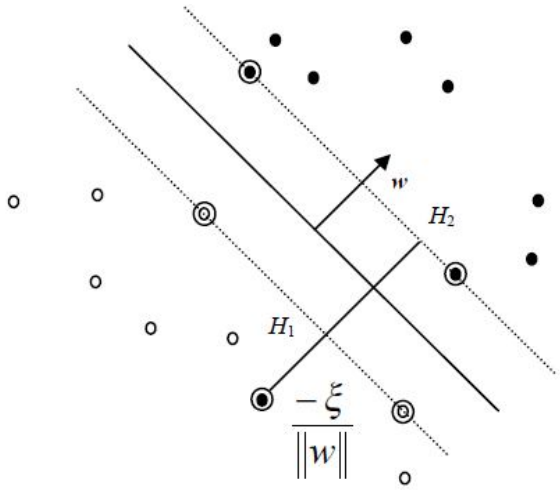

FIG.II.6 : *Les hyperplans linéaires pour un problème de classification non linéairement séparable. Lorsqu'il y a une erreur sur un exemple, cet exemple est considéré comme un vecteur support dont sa distance avec l'hyperplan de sa vraie classe est* -ξ/‖*w*‖.

$$y_i\left(\langle w, x_i\rangle + b\right) \geq 1 - \xi_i \qquad (4.6)$$

$$\forall i, \quad \xi_i \geq 0 \qquad (4.7)$$

D'après (4.6) et (4.7) lorsqu'il y a une erreur pour un point x_i donné, la variable ξ_i correspondante doit être plus grande que l'unité, puisque dans ce cas +1-ξ_i ou -1+ξ_i changent de signe. Par ce fait $\sum_i \xi_i$ détermine une borne supérieure du nombre d'erreurs en apprentissage. Ainsi, une façon

simple de traiter ce cas est de changer la fonction objective du problème primal de $\frac{\|w\|^2}{2}$ en $\frac{\|w\|^2}{2} + C$, où C est un paramètre fixé à l'avance. Plus C est grand plus on pénalise les erreurs. La figure (4.2.1) illustre ce cas.

Comme précédemment le lagrangien de ce problème s'écrit :

$$(QP3) \qquad \textit{Minimiser} \qquad L(w,b,x_i) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\textit{tel que} \quad \begin{cases} y_i\left(\langle w, x_i\rangle + b\right) \geq 1 - \xi_i & \forall i = 1..n \\ \xi_i \geq 0 & \forall i = 1..n \end{cases}$$

En formant le Lagrangien, puis en appliquant le théorème (KKT), on trouve la forme duale suivante :

$$(QP4) \qquad \textit{Maximiser} \quad W(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

$$\textit{tel que} \quad \begin{cases} \sum_{i=1}^{n}\alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C \qquad \forall i = 1...n \end{cases}$$

Nous terminons en mentionnant que l'on peut à nouveau déterminer le statut d'un exemple x_i en regardant sa variable duale α_i. Cette fois-ci il existe trois statuts différents :

1. $\alpha_i = 0$: L'exemple est bien classé et n'est pas sur un des deux hyperplans canoniques. On dira que l'exemple est un non-SV.
2. $0 < \alpha_i < C$: L'exemple est bien classé et se trouve sur un hyperplan canonique. Il s'agit donc d'un SV.
1. $\alpha_i = C$: L'exemple est mal classé. Il sera malgré tout considéré comme SV puisque $\alpha_i > 0$. Il s'agit d'un *outlier*.

II.5.3) Les fonctions Noyaux (Kernels)

Le gros intérêt des noyaux est que tout ce qu'on vient de voir sur la séparation linéaire s'applique en fait très facilement à des séparations non linéaires, sous réserve de bien faire les choses.

II.5.3.1) L'espace des caractéristiques

Imaginons un ensemble d'exemples x_i étiquetés par -1 ou +1 suivant la classe à laquelle ils appartiennent, qui ne soient pas du tout séparable linéairement. La méthode vue au section précédente fonctionne, mais la séparation est bien entendu de piètre qualité, et bon nombre de vecteurs sont des supports.

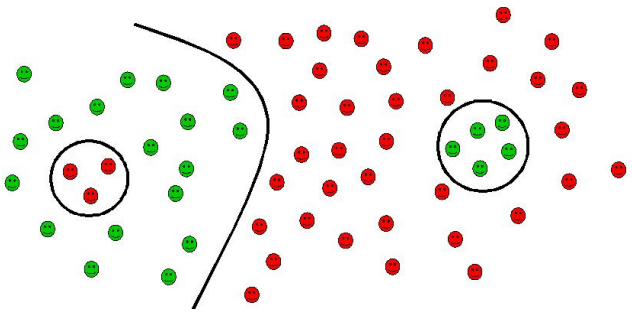

FIG.II.7 : *Exemples non séparables linéairement*

Une solution pour mieux séparer les exemples est de les projeter dans un espace différent (le *feature space*)[4], et de réaliser une séparation linéaire dans cet espace-là, où cette fois-ci elle devrait être plus adaptée.

Nous noterons le *feature space* F, et le *mapping* Φ vers cet espace, on a : $\Phi : X \to F$

$$\Phi(x) = \begin{pmatrix} \varnothing_1(x) \\ \varnothing_2(x) \\ . \\ . \\ \varnothing_n(x) \end{pmatrix}$$

[4] Souvent de dimension très élevée

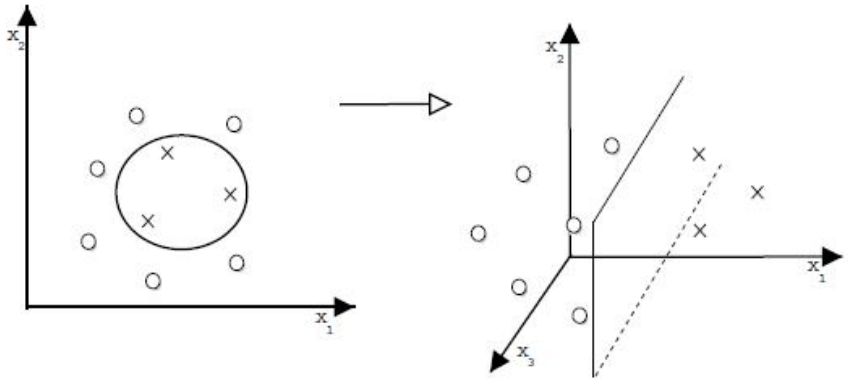

FIG.II.8 : *Un mapping Φ rendant les exemples linéairement séparables*

Les fonctions $\varnothing_i$ ne sont pas nécessairement linéaires, et on peut même avoir n = ∞! Si l'on reprend les méthodes de la section précédente, et que l'on travaille dans l'espace des caractéristiques (*feature space*), c'est-à-dire si on travaille avec le corpus

$$\overline{S} = \{(\Phi(x_i), y_i)\}_{0 \le i \le n} \quad avec \quad \forall i = 1..n \quad y_i \in \{-1,1\}$$

Au lieu de $S = \{(x_i, y_i)\}_{0 \le i \le n}$ *avec* $\forall i = 1..n$ $y_i \in \{-1,1\}$

On se retrouve à faire de la séparation linéaire sur le corpus $\overline{S}$. On obtient un séparateur, donné par la formule $w = \sum_{i=1}^{n} \alpha_i y_i x_i$ et le *b*. Pour décider de la classe d'un vecteur *x*, on pourrait calculer alors $\Phi(x)$, que l'on passerait à son tour comme argument du séparateur pour en connaître la classe +1 où −1.

En fait, on ne procédera pas comme ça, et on préférera éviter le calcul explicite de $\Phi(x)$ en remarquant que le problème d'optimisation posé à la section précédente ne fait intervenir les vecteurs que via des produits scalaires entre eux.

Notons *k(x,z)* le produit $\langle \Phi(x), \Phi(z) \rangle$. Travailler sur le corpus $\overline{S}$ revient à travailler sur le corpus S avec les algorithmes précédents, mais en remplaçant toutes les occurrences de $\langle \bullet, \bullet \rangle$ par $k(\bullet, \bullet)$.

Pour l'instant, on ne voit pas trop l'intérêt, vu que pour calculer *k(x,z),* il faut appliquer la définition, à savoir projeter *x* et *z* dans l'espace des caractéristiques et calculer le produit scalaire, dans cet espace, des deux vecteurs obtenus.

La ruse en fait est qu'on ne fera pas cette projection, car on calculera *k (x, z)* autrement. En fait, *k (x, z)* est une fonction que l'on va se donner, en s'assurant qu'il existe bien en théorie une projection Φ dans un espace qu'on ne cherchera pas à décrire. Ainsi, on calculera directement *k (x, z)*, à chaque fois que l'algorithme précédent requiert un produit scalaire, et c'est tout ! La projection dans le gros espace de caractéristiques sera implicite.

II.5.3.2) Conditions pour avoir un noyau

Il y a des conditions mathématiques, appelées théorème de Mercer, qui permettent de dire si une fonction est un noyau ou non, sans construire la projection dans l'espace des caractéristiques.

Théorème de Mercer : la fonction *k(x,z) :X* x *X* → *R* est un noyau SSi :

$$G = \left(k(x_i, x_j)\right)_{i,j=1}^{n} \quad est \quad définie \quad positive$$

Notons qu'une fonction *X* x *X* → *R* générant une matrice définie positive possède les trois propriétés fondamentales du produit scalaire : $\forall x_i, x_j \in X$

1. poitivité : $k(x_i, x_j) \geq 0$
2. symétrie : $k(x_i, x_j) = k(x_j, x_i)$
3. inégalité de Cauchy-Shwartz : $\left|k(x_i, x_j)\right| \leq \|x_i\| . \|x_j\|$

La condition de Mercer nous indique si une fonction est un *kernel* mais nous n'avons aucun renseignement sur le mapping Φ (et donc sur le *feature space*) induit par ce *kernel*. A la section

suivante, nous présentons deux *kernels* considérés comme standards. Nous verrons que la nature du *feature space* est un peu particulière.

II.5.3.3) Exemples de kernels

II.5.3.3.1) Kernel polynomial

La forme générique de ce kernel est : $k(x,z) = (a * \langle x,z \rangle + b)^d$

et correspond à une projection $\Phi(x)$ dans un espace de caractéristiques où chaque composantes $\Phi_i(x)$ est un monôme de degré inférieur à d de certaines des composantes de *x*. Le séparateur calculé avec ce noyau est un polynôme de degré *d* dont les monômes sont les composantes de *x*. La constante *c*, quand elle est élevée, donne plus d'importance aux monômes de degré élevé.

II.5.3.3.2) Le kernel RBF (Radial Basis Function)

La forme générique de ce kernel est : $k(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$

L'espace d'arrivée *F* (le *feature space*) de la fonction Φ est de dimension infinie étant donné que Φ fait correspondre une fonction continue à chaque exemple, Le *kernel RBF* permet donc de calculer des similarités dans un espace de dimension infinie.

Avec le kernel *RBF*, tous les exemples sont placés sur une sphère de rayon 1 :

$$k(x,x) = \exp\left(-\frac{\|x-x\|^2}{2\sigma^2}\right) = 1$$

Le paramètre σ permet de régler la largeur de la gaussienne. En prenant un σ grand, la similarité d'un exemple par rapport à ceux qui l'entoure sera assez élevée, alors qu'en prenant un σ tendant vers 0, l'exemple ne sera similaire à aucun autre. En resserrant fortement la gaussienne, un classificateur (faisant usage de ce kernel) peut arriver à apprendre n'importe quel *training set* sans commettre d'erreur. On sent tout de suite que le danger de l'apprentissage par cœur n'est pas loin. En fait, σ est un autre paramètre permettant de contrôler la capacité d'un classificateur.

II.5.3.3.3) Composition des kernels

Il est possible de composer des nouveaux kernels en utilisant des kernels existants. En prenant $k_1(.,.)$ et $k_2(.,.)$ des fonctions satisfaisant à la condition de Mercer, $a \in R^+$ et B une matrice définie positive, alors les fonctions suivantes sont des kernels :

1. $k(x,z) = k_1(x,z) + k_2(x,z)$
2. $k(x,z) = ak_1(x,z)$
3. $k(x,z) = k_1(x,z)k_2(x,z)$
4. $k(x,z) = x^T Bz$

II.5.4) Formulation de SVM

II.5.4.1) Cas linéairement séparable

La méthode SVM consiste en un classificateur à marge maximale dans lequel le produit scalaire a été remplacé par le kernel. Effectuons dès lors cette substitution dans le problème (QP2).

$$(QP5) \qquad Maximiser \quad W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(x_i, x_j)$$

$$tel\ que \quad \begin{cases} \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \alpha_i \geq 0 \qquad \forall i = 1...n \end{cases}$$

II.5.4.2) Formulation Soft Margin

Cela revient à relaxer la contrainte imposant que tous les exemples soient bien classés.

$$(QP6) \qquad Maximiser \quad W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(x_i, x_j)$$

$$tel\ que \quad \begin{cases} \sum_{i=1}^{n} \alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C \qquad \forall i = 1...n \end{cases}$$

Conclusion

Dans ce chapitre, nous avons introduit la classification en présentant deux algorithmes de philosophies très différentes : Naïve Bayes et les arbres de décision.

Puis, nous avons détaillé la méthode principale de ce mémoire : SVM. Tout d'abord nous avons mis en évidence la relation qui existe entre la marge (de l'hyperplan séparateur) des classificateurs linéaires et la théorie de l'apprentissage statistique. Plus spécialement, nous avons vu que le fait de maximiser la marge permettait de réduire la capacité des fonctions du classificateur. Nous avons ensuite formulé la recherche de l'hyperplan optimal sous forme d'un problème d'optimisation quadratique. Il s'est avéré que l'on pouvait en donner une représentation duale avantageuse. Principalement, cette formulation permet de limiter le nombre d'inconnus du problème au nombre d'exemples du *training set*. Cela constitue une des avantages de la méthode SVM par rapport aux autres méthodes face à la course aux dimensions. La solution apportée par SVM pour traiter des problèmes non-linéairement séparables consiste à utiliser les fonctions *kernels*. L'avantage principal de ces fonctions est de pouvoir apprendre un modèle de classification pour des exemples non linéairement séparables.

Chapitre III

Les algorithmes de décomposition

Introduction

L'apprentissage d'une machine à vecteurs de support (SVM) mène à un problème d'optimisation quadratique avec des contraintes d'inégalité et une contrainte linéaire d'égalité. Malgré le fait que ce type de problème est bien compris, il y a beaucoup d'issues à considérer pour concevoir une machine SVM, la maturité de ces approches reposent sur l'idée de décomposition. Cette décomposition dédouble (*QP*) en deux, une partie inactive et une autre active que nous applerons par la suite le *working set*. L'avantage principal de cette décomposition est qu'il suggère des algorithmes avec des conditions de mémoire linéaires dans le nombre de SV_S. Un inconvénient potentiel est que ces algorithmes peuvent avoir besoin d'un long temps d'apprentissage.

III.1) Les méthodes de décomposition

Reprenant la version matricielle du problème d'optimisation (QP6) :

$$(QP) \qquad Minimiser \quad W(\alpha) = -\alpha^T 1 + \frac{1}{2}\alpha^T Q\alpha \qquad (6.4)$$

$$tel\ que \quad \begin{cases} \alpha^T y = 0 & (6.5) \\ 0 \le \alpha \le C1 \quad \forall i = 1...n & (6.6) \end{cases}$$

Où la matrice Q est défini par $Q_{ij} = y_i y_j k(x_i, x_j)$

La taille du problème d'optimisation dépend du nombre d'exemples du training set noté *l*. La taille de la matrice Q est l^2, pour un *set training* de 10000 exemples et plus, il devient impossible de maintenir Q en mémoire. Plusieurs implémentations des solutions algorithmiques pour (QP) nécessitent un stockage explicite de Q ce qui empêche leur application.

III.1.1) Structure commune des algorithmes de résolution

Les algorithmes itératifs de résolution partagent une structure commune. Les méthodes se distinguent ensuite par leur façon de répartir les points et par le calcul des α.

Algorithme : Schéma général des algorithmes de résolution itératifs

1: initialisation

2: tant que la solution courante n'est pas optimale **faire**

3: mettre à jour à la répartition des groupes

4: calculer les coefficients α_i correspondant aux changements

5: fin tant que

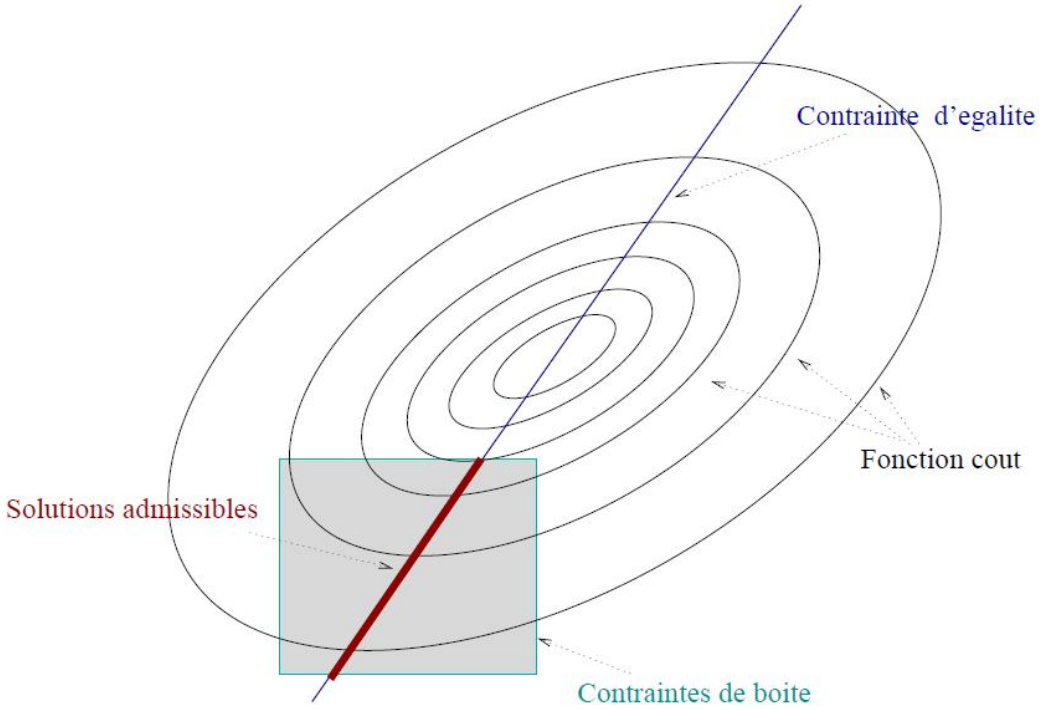

FIG.III.1: *Géométrie du problème quadratique dual. Les contraintes de boîte* $0 \le \alpha_i \le C$ *restreignent la solution à un hypercube de dimension n. La contrainte d'égalité* $\sum y_i \alpha_i = 0$ *restreint de plus la solution à un polytope de dimension n−1. La fonction de coût quadratique limitée à la ligne de recherche peut avoir son maximum à l'intérieur ou à l'extérieur de boîte de contraintes.*

III.2) Algorithmes de décomposition

Les méthodes de décomposition, comme leur nom l'indique, utilisent des sous-ensembles de la base d'apprentissage à chaque étape, de manière à résoudre des problèmes de petite dimension.

Les résultats de ces sous-problèmes sont ensuite combinés pour arriver à la solution optimale globale.

La propriété utilisée : la parcimonie

Notons tout d'abord que les formes à optimiser sont convexes et par là même n'admettent qu'une seule solution optimale. Notons par ailleurs que les contraintes sont linéaires et en nombre fini : la solution optimale α vérifie pour chacune de ses composantes α_i les conditions de Karuch Kuhn Tucker (KKT), c'est-à-dire :

$$\alpha_i = 0 \quad \rightarrow \quad y_i f(x_i) > 1$$

$$0 < \alpha_i < C \quad \rightarrow \quad y_i f(x_i) = 1$$

$$\alpha_i = C \quad \rightarrow \quad y_i f(x_i) < 1$$

Le principe général des méthodes de décomposition et/ou de contraintes actives repose sur l'observation que seuls les points non contraints dans la solution nécessitent le calcul de leur coefficient : c'est la parcimonie. En effet, les autres ont pour coefficient une valeur fixe donnée par le problème : celle de la borne qui les contraint. Ce constat a amené à différentes techniques de décompositions et différents algorithmes que nous allons détailler.

III.2.1) L'algorithme SMO (Sequential Minimal Optimization)

L'algorithme *Sequential Minimal Optimization* (SMO) posé par **[Platt, 1998]** est un algorithme qui permet de résoudre rapidement le problème quadratique (QP4) du SVM sans passer par toutes les étapes de résolution numérique d'un QP. L'idée principale des algorithmes de décomposition est de travailler avec un sous-ensemble réduit de données du problème, garder les solutions et continuer avec le reste des données où les solutions antérieures doivent être encore testées. La SMO prend cette idée à l'extrême : elle optimise seulement deux vecteurs par itération. Cette optimisation admet une solution analytique. A chaque itération, la SMO choisit deux coefficients de Lagrange α_i et α_j pour les optimiser ensemble, trouver ses valeurs optimales étant donné que toutes les autres sont fixes, et actualiser le vecteur solution α.

$$L_D = \alpha_1 + \alpha_2 - \frac{1}{2}k(x_1,x_2)\alpha_1^2 - \frac{1}{2}k(x_2,x_2)\alpha_2^2 - 2y_1y_2k(x_1,x_2)\alpha_1\alpha_2 - y_1\alpha_1v_1 - y_2\alpha_2v_2 + cte$$

Avec : $v_i = \sum_{j=3}^{n} y_i\alpha_i k(x_i,x_j)$

En respectant les contraintes : $0 \le \alpha_1,\alpha_2 \le C$ Et $\sum_{i=1}^{n}\alpha_i y_i = 0$

On dérive L_D par rapport à α_2 et on obtient des expressions de cette variable en fonction de l'erreur de classification (1)

$$\alpha_2^{new} = \alpha_2^{old} - \frac{y_2(E_1 - E_2)}{K}$$

$$\alpha_1^{new} = \alpha_1^{new} + y_1y_2(\alpha_2^{old} - \alpha_2^{new})$$

Où $K = k(x_1,x_1) + k(x_2,x_2) - 2k(x_1,x_2)$ et $E_i = f(x_i) - y_i = \left(\sum_{j=1}^{n}\alpha_j y_j k(x_j,x_i) + b\right) - y_i$ (1)

La SMO optimise deux coefficients à chaque itération. Un des deux doit violer les conditions de KKT pour être choisi dans l'itération courante.

Le critère d'arrêt de la SMO est aussi la surveillance des conditions de KKT. La comparaison entre les deux formulations du Lagrangien et de son dual peut être utilisée pour assurer la convergence.

Après apprentissage, la fonction de décision d'écrit :

$$f(x) = sign\left(\sum_{i=1}^{m}\alpha_i y_i k(x,x_i) + b\right)$$

où *m* est le nombre de vecteurs support. *f(x)* et l'étiquette de classification.

L'avantage d'une telle approche est de pouvoir solutionner les sous problèmes élémentaires de manière analytique. De plus, la complexité spatiale est très faible : la place en mémoire requise est à peu près celle qu'il faut pour stocker le *training set*.

III.2.2) L'algorithme de décomposition D'Osuna

L'idée générale de la technique de décomposition de **[Osuna et al. 1997a]** est de garder une taille de sous ensemble constante en laissant la possibilité de retirer du sous ensemble courant des points vecteurs supports. Ainsi on peut résoudre les problèmes quadratiques quelque soit la taille réelle du problème.

Les conditions d'optimalité

Le problème quadratique que nous devons résoudre est le suivant :

$$(QP4) \qquad Minimiser \quad W(\alpha) = -\alpha^T 1 + \frac{1}{2}\alpha^T Q\alpha$$

$$tel\ que \quad \begin{cases} \alpha^T y = 0 \\ -\alpha \le 0 \quad \forall i = 1...n \\ \alpha - C \le 0 \end{cases}$$

Donc les conditions de Kuhn-Tuker sont nécessaires et suffisantes pour l'optimalité. Les conditions de KT sont comme suit :

$$\nabla W(\alpha) + \left(\lambda^{eq} y - \lambda^{lo} + \lambda^{up}\right) = 0$$

$$\forall i = 1..n \qquad \lambda_i^{lo}\left(-\alpha_i\right) = 0$$

$$\forall i = 1..n \qquad \lambda_i^{up}\left(\alpha_i - C\right) = 0$$

$$\alpha^T y = 0$$

$$\lambda^{up} \ge 0 \qquad (1)$$

$$\lambda^{lo} \ge 0 \qquad (2)$$

$$0 \le \alpha \le C1$$

Afin de dériver encore d'autres expressions algébriques des conditions d'optimalité (1) et (2) nous assumons l'existence de certains α_i tels que $0 < \alpha_i < C$, et considérer les trois valeurs possibles que chaque composant de α_i peut avoir :

Cas 1 : $0 < \alpha_i < C$ pour les trois premières équations des conditions de KT nous avons :

$$(Q\alpha)_i - 1 + \lambda^{eq} y_i = 0 \qquad (3)$$

En utilisant les résultats de **[Cortes et al, 1995]** et **[Vapnik, 1995],** on peut facilement prouver que quand α est strictement entre 0 et C l'égalité suivante:

$$y_i \left(\sum_{j=1}^{l} \alpha_j y_j k(x_i, x_j) + b \right) = 1 \qquad (4)$$

Et on a $(Q\alpha)_i = y_i \left(\sum_{j=1}^{l} \alpha_j y_j k(x_i, x_j) \right)$

En combinant cette expression avec (3) et (4) nous obtenons immédiatement $\lambda^{eq} = b$.

Cas 2 : $\alpha_i = C$

Par la définition $f(x_i) = \left(\sum_{j=1}^{l} \alpha_j y_j k(x_i, x_j) + b \right)$

et notant cela $(Q\alpha)_i = y_i \left(\sum_{j=1}^{l} \alpha_j y_j k(x_i, x_j) \right) = y_i (f(x_i) - b)$

nous concluons que $y_i f(x_i) \le 1$

Cas 3 : $\alpha_i = 0$

En appliquant une manipulation algébrique semblable en tant que celle décrite pour le cas 2, nous obtenons $y_i f(x_i) \ge 1 \qquad (5)$

Stratégie de décomposition

Les conditions d'optimalité dérivés dans la section précédente sont essentielles afin de concevoir une stratégie de décomposition qui prend l'avantage que la majorité des α_i sont nuls et ce garantit qu'à chaque itération la fonction objective prend pas vers le minimum. Afin d'accomplir ce but on divise l'ensemble des variables en deux ensembles *B* et *N* de telle manière que les conditions d'optimalité se tiennent dans le sous-problème défini seulement pour les variables dans l'ensemble *B*, qu'on appele *working set,* puis on décompose le vecteur α en deux vecteurs α_B et α_N et mettre $\alpha_N = 0$. En utilisant cette décomposition les rapports suivants sont clairement vrais :

- On peut remplacer $\alpha_i = 0 \quad i \in B$, avec $\alpha_j = 0 \quad j \in N$, sans changer le coût de la fonction ou la praticabilité du sous-problème et du problème original.
- Après un tel remplacement, le nouveau sous-problème est optimal si et seulement si $y_i f(x_i) \geq 1$ Ceci résulte de l'équation (5) et de la prétention que le sous-problème était optimal avant que le remplacement ait été fait.

Les rapports précédents mènent à la proposition suivante :

Proposition :

Etant donné une solution optimale d'un sous-problème défini sur *B*, l'opération du remplacement $\alpha_i = 0 \quad i \in B$ avec $\alpha_j = 0 \quad j \in N$ vérifiant $y_i f(x_i) < 1$ génère un nouveau sous-problème qui une fois optimisé, rapporte une amélioration stricte de la fonction objective. Une preuve détaillée de cette proposition peut être trouvée dans **[Osuna et al. 1997b]**.

L'algorithme de décomposition

Supposons que nous pouvons définir un *working set B* à taille fixe, tel que $|B| \leq l$, et il est assez grand pour contenir tous les vecteurs de support, mais assez petit tels que la machine peut le manipuler et l'optimiser en utilisant un certain QP solveur. Alors l'algorithme de décomposition peut être énoncé comme suit :

1. Choisir arbitrairement $|B| \le l$ points de l'ensemble d'apprentissage.
2. Résoudre le sous-problème défini par les variables dans le B.
3. tant qu'il existe $j \in N$, tel que $y_i f(x_i) < 1$, remplacer $\alpha_i = 0 \quad i \in B$ avec $\alpha_j = 0 \quad j \in N$ et résoudre le nouveau sous-problème.

Notons que, selon les conditions d'optimalité décrites ci-dessus, cet algorithme amélioreront strictement la fonction objective à chaque itération et donc ne feront pas un cycle. Puisque la fonction objective est limitée l'algorithme doit converger à la solution optimale globale dans un nombre fini d'itérations. **FIG.II.5.5.1.1** donne une interprétation géométrique de la manière que l'algorithme de décomposition permet la redéfinition de la surface de séparation en ajoutant les points qui violent les conditions d'optimalité.

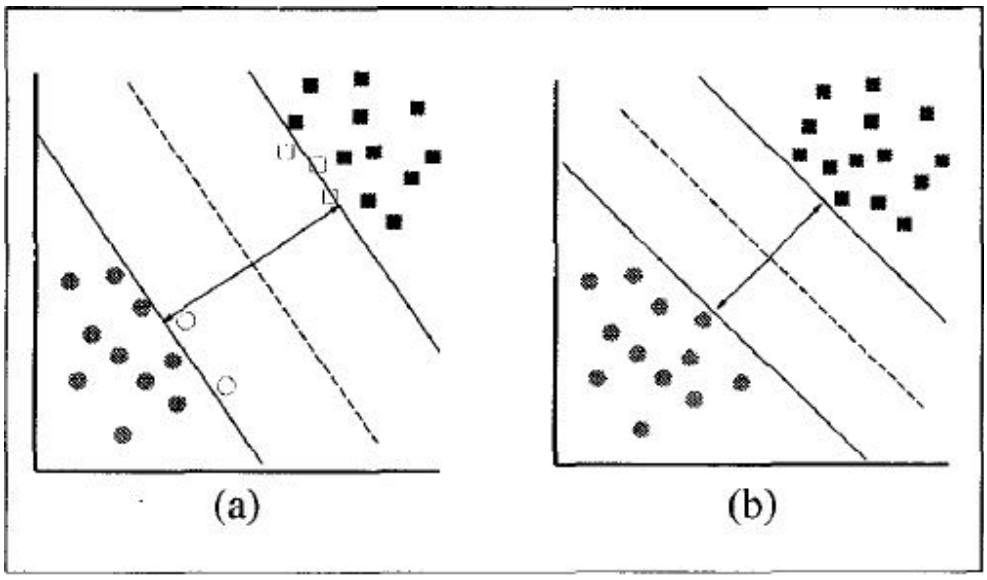

FIG.III.2 : *(a) Une solution sous-optimale où les points non-remplis ont $\alpha = 0$ mais violent des conditions d'optimalité en étant à l'intérieur de l'intervalle [-1,+1]. (b) La surface de décision est redéfinie. Depuis aucuns points avec $\alpha = 0$ sont à l'intérieur de l'intervalle [-1,+1], la solution est optimale. Noter que la taille de la marge a diminué, et la forme de la surface de décision a changé.*

III.2.3) L'algorithme SVMlight

Le SVM^{light} est un algorithme proposé par **[Joachims, 1998]**, qui utilise l'idée de décomposition de **[Osuna et al. 1997a]**. L'avantage principal de cette décomposition est qu'il

suggère des algorithmes avec des conditions de mémoire linéaires dans le nombre d'exemples d'apprentissage et linéaires dans le nombre de vecteurs supports. Un inconvénient potentiel est que ces algorithmes peuvent avoir besoin d'un long temps d'apprentissage. Pour aborder ce problème, **[Joachims, 1999]** propose un algorithme qui incorpore les idées suivantes :

- Une méthode efficace pour choisir le *working set*.
- « Shrinking » successif du problème d'optimisation. Ceci exploite la propriété que
  - beaucoup moins de vecteurs supports (SVs) que des exemples d'apprentissage.
  - Beaucoup de SVs qui ont un α_i à la limite supérieure C.
- L'amélioration de techniques de calcul tel que cacher et mettre à jour d'une façon incrémentale le gradient et les critères d'arrêt.

Shrinking Le shrinking **[Joachims, 1999]** est une heuristique visant à déterminer au cours de l'avancement de l'algorithme quels points seront certainement exclus de la solution ou bornés. On connaîtra leur valeur sans avoir à calculer leur coefficient α_i. De cette façon, on peut ne plus tenir compte de ces points et réduire mécaniquement la taille du problème à résoudre. Comme cette heuristique est faillible, il faut vérifier à l'arrêt de l'algorithme que les points exclus sont dans le bon groupe I_0 ou I_C et éventuellement de refaire une étape d'optimisation.

L'algorithme de décomposition :

La stratégie de décomposition est que pour chaque itération les variables α_i se divisent en deux catégories :

1. l'ensemble *B* des variables libres (*free variables*)
2. l'ensemble *N* des variables fixes (*fixed variables*)

Les variables libres sont ceux qui peuvent être mises à jour dans l'itération courante, considérant que les variables fixes sont temporairement fixés à une valeur particulière. L'ensemble de variables libres est désigné sous le nom de *Werking set* ayant une taille notée par q beaucoup plus petit que *l*. L'algorithme fonctionne comme suit :

Si les conditions d'optimalité ne se tiennent pas, l'algorithme décompose (QP4) et résout les petit QP-problème résultant de ceci. La décomposition se fait en séparant les variables dans

Wrking set B de celles qui sont fixes (N). Assumons que α , y, et Q sont correctement arrangés en ce qui concerne B et N, de sorte que :

$$\alpha = \begin{vmatrix} \alpha_B \\ \alpha_N \end{vmatrix} \quad y = \begin{vmatrix} y_B \\ y_N \end{vmatrix} \quad Q = \begin{vmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{vmatrix}$$

On a $Q_{BN} = Q_{NB}^T$ on peut écrire

$$(QP'5) \quad Minimiser: \quad W(\alpha) = -\alpha_B^T(1 - Q_{BN}\alpha_N) + \frac{1}{2}\alpha_B^T Q_{BB}\alpha_B + \frac{1}{2}\alpha_N^T Q_{NN}\alpha_N - \alpha_N^T 1$$

$$tel\,que \quad \alpha_B^T y_B + \alpha_N^T y_N = 0$$

$$0 \le \alpha \le C1$$

Les variables de N sont fixées, le terme $\frac{1}{2}\alpha_N^T Q_{NN}\alpha_N - \alpha_N^T 1$ et constant donc il peut être omis sans changer la solution de (QP'5) qui est un problème semi-défini positif de programmation quadratique qui est assez petit soit résolu par la plupart des méthodes disponibles. Où la vitesse de la convergence vers la solution optimale dépend du choix du *Wrking set*.

La stratégie de sélection du *working set :* En choisissant le *working set*, il est souhaitable de choisir un ensemble de variables tels que l'itération courante accomplira beaucoup de progrès vers le minimum de $W(\alpha)$. Ce qui suit propose une stratégie basée sur la méthode de **[Zoutendijk, 1970]** qui utilise l'approximation du premier ordre à la fonction objective. L'idée est de trouver la plus raide direction admissible de descente qui a seulement q éléments différents de zéro. Les variables correspondant à ces éléments composeront le *working set* courant. Cette approche mène au problème d'optimisation suivant:

$$(QP'6) \quad Minimiser \quad V(d) = \nabla W(\alpha^t)^T d$$

$$telque: \quad y^T\alpha = 0 \qquad (1)$$

$$d_i \ge 0 \quad pour \quad i : \alpha_i = 0 \qquad (2)$$

$$d_i \le 0 \quad pour \quad i : \alpha_i = C \qquad (3)$$

$$-1 \le d \le 1 \qquad (4)$$

$$\left|\{d_i : d_i \ne 0\}\right| = q \qquad (5)$$

L'objectif (1) montre qu'une direction de descente est cherchée. La contrainte (2) assure la vérification de celle (6.5) du problème (QP), tandis que (2) et (3) sont des contraintes excluant les directions menant uniquement à des points non admissibles. La contrainte (4) normalise le vecteur de descente pour faire le problème d'optimisation bien posé. Finalement, la contrainte (5) déclare que la direction de la descente impliquera seulement q variables. Les variables avec des d_i différents de zéro sont incluses dans le *working set B*. De cette façon nous choisissons le *working set* avec la direction admissible de descente la plus raide.

La convergence

La stratégie de sélection, les conditions d'optimalité, et la décomposition spécifient ensemble l'algorithme d'optimisation. Une condition minimum que cet algorithme doit accomplir est que :

- se termine seulement quand la solution optimale est atteinte.
- sinon, prend un pas vers l'optimum

La première condition peut facilement être remplie par la vérification des conditions d'optimalité citées dans la section (II.5.5.1.1) dans chaque itération. Pour le deuxième cas nous avons un $\alpha^{(t)}$ non optimal. Alors la stratégie de sélection de *working set* renvoie un problème d'optimisation du type (QP'5). Pour ce problème d'optimisation, il existe un vecteur ***d*** qui est une direction admissible de descente, pour laquelle en appliquant les résultats de **[Zoutendijk, 1970],** la solution d' (QP'5) est également faisable pour le problème général (QP)**.** Ceci signifie que nous obtenons une descente stricte dans la fonction objective d' (QP) dans chaque itération.

Comment résoudre *(QP'6)*

Il est facile calculer la solution de (QP'6) en utilisant une stratégie simple**.** Mettant $w_i = y_i g_i\left(\alpha^{(t)}\right)$, et assortir tous les α_i selon w_i par ordre décroissant. En exigeant que q soit un chiffre pair. On sélectionne successivement les éléments *q/2* à partir du dessus de la liste pour lesquels $0 < \alpha_i^{(t)} < C$, ou $d_i = -y_i$ obéit (2) et (3). De même, on sélectionne les éléments *q/2* du fond de la liste pour lesquels $0 < \alpha_i^{(t)} < C$, ou $d_i = y_i$ obéit (2) et (3). Ces q variables composent le *working set*.

III.3) Proposition et implémentation

Si les SVM (*Support Vector Machines*, ou Séparateurs à Vaste Marge) sont aujourd'hui reconnus comme l'une des meilleures méthodes d'apprentissage, ils restent considérés comme lents, pour cela et dont l'objectif d'obtenir un temps d'apprentissage attractif par rapport aux algorithmes existants nous proposerons l'application d'une stratégie, que nous avons personnellement mise au point, et qui permet d'accélérer le temps d'exécution de l'algorithme proposé par **[Osuna et al. 1997b]**. Nous avons choisi de coder cet algorithme dans l'environnement Matlab afin de profiter de sa convivialité ainsi son optimiseur *quadprog()* qui peut aller bien pour un nombre de 1000 exemples d'apprentissage. La comparaison de notre solution avec l'algorithme proposé par Osuna montre qu'il s'agit là d'une solution rapide en terme de temps d'apprentissage et d'une complexité moindre. Pour illustrer la simplicité et la rapidité de notre méthode, nous montrons enfin que sur la base de données *Reuters*, il a été possible d'obtenir des résultats satisfaisants en un temps relativement court.

III.3.1) La stratégie de l'algorithme

Notre stratégie se base sur le même principe de la proposition de **[Osuna et al. 1997b],** on divise l'ensemble des variables en deux ensembles *B* et *N* de telle manière que les conditions d'optimalité se tiennent dans le sous-problème défini seulement pour les variables dans l'ensemble *B*, qu'on appelle *working set,* puis on décompose le vecteur α en deux vecteurs α_B et α_N et mettre $\alpha_N = 0$. La différence réside dans la politique de remplacement des éléments de *B* vérifiant $\alpha_i = 0$ par les éléments de *N* qui vérifient $y_j f(x_j) < 1$, où notre méthode va sélectionner l'élément le plus proche de l'hyperplan courant en terme de marge fonctionnelle. L'application de l'algorithme suivant garantit qu'à chaque itération la fonction objective prend un pas plus rapide vers le minimum.

4.Choisir arbitrairement $|B| \le l$ points de l'ensemble d'apprentissage.

5.Résoudre le sous-problème défini par les variables dans le *B*.

6.tant qu'il existe $j \in N$, tel que $y_j f(x_j) < 1$, remplacer $\alpha_i = 0 \quad i \in B$, avec $\alpha_j = 0$

$pour \quad j \in N \quad y_j f(x_j) < y_k f(x_k) \quad \forall k \in N$, et résoudre le nouveau sous-problème.

Preuve :

Nous supposons l'existence de α_p tel que $0 < \alpha_p < c$. Supposons aussi sans perte de généralité que $y_p = y_j$ (la preuve est analogue pour $y_p = -y_j$). Alors il existe un $\varepsilon > 0$ tel que $\alpha_p - \delta > 0$ pour $\delta \in (0, \varepsilon)$. Remarquons également que $f(x_p) = y_p$, on considère $\overline{\alpha} = \alpha + \delta e_j - \delta e_p$, où e_j et e_p sont le j ième et p ième vecteur d'unité. Et notez que le opération pivote peut être traitée implicitement en laissant $\delta > 0$ et en tenant $\alpha_i = 0$. Le nouveau coût de la fonction objective $W(\overline{\alpha})$ peut être écrit comme suit :

$$
\begin{aligned}
W(\overline{\alpha}) &= -\overline{\alpha} \cdot 1 + \frac{1}{2}\overline{\alpha} \cdot Q\overline{\alpha} \\
&= -\alpha \cdot 1 + \frac{1}{2}\overline{\alpha}\left[\alpha \cdot Q\alpha + 2\alpha \cdot Q(\delta e_j - \delta e_p) + (\delta e_j - \delta e_p) \cdot Q(\delta e_j - \delta e_p)\right] \\
&= W(\alpha) + \delta\left[\frac{f(x_j) - b}{y_j} - 1 + \frac{b}{y_p}\right] + \frac{\delta^2}{2}\left[K(x_j, x_j) + K(x_p, x_p) - 2y_j y_p K(x_j, x_p)\right] \\
&= W(\alpha) + \delta\left[y_j f(x_j) - 1\right] + \frac{\delta^2}{2}\left[K(x_j, x_j) + K(x_p, x_p) - 2y_j y_p K(x_j, x_p)\right]
\end{aligned}
$$

A cet effet, puisque $y_j g(x_j) < 1$ en choisissant δ suffisamment petit, nous avons $W(\overline{\alpha}) < W(\alpha)$.

Supposons maintenant l'existence d'un élément α_k qui vérifie $y_k g(x_k) < y_j g(x_j) < 1$, et $\overline{\overline{\alpha}} = \alpha + \delta e_k - \delta e_p$

$$
W(\overline{\overline{\alpha}}) - W(\overline{\alpha}) = \delta\left[(y_k f(x_{kj})) - (y_j f(x_j))\right] + \frac{\delta^2}{2}\begin{bmatrix}(K(x_k, x_k) - 2y_k y_p K(x_k, x_p)) - (K(x_j, x_j) + \\ 2y_j y_p K(x_j, x_p))\end{bmatrix}
$$

A cet effet, puisque $y_k g(x_k) < y_j g(x_j) < 1$ en choisissant δ suffisamment petit, nous avons $W(\overline{\overline{\alpha}}) < W(\overline{\alpha}) < W(\alpha)$

III.4) Conception algorithmique

III.4.1) Stockage des documents et calcul des kernels

Notre algorithme est destiné à effectuer de la classification de documents textuels. Notre choix de représentation s'est naturellement porté sur le modèle vectoriel. Dans le cas de la représentation *bag of words*, les vecteurs présentent un très grand nombre de composantes nulles. Cette observation nous incite à considérer une manière de stocker les exemples de façon économique. (FIG III.3)

doc_i

ind	4	7										
val	0.5	0.2										

Labels

+1	-1											

FIG III.3 : *Structure de donnée utilisée pour représenter un document*

Il nous faut à présent définir des algorithmes permettant de calculer les *kernels* sur ces structures creuses. On a conçu une fonction qui calcule le produit scalaire creux. En plus de présenter une économie de stockage, cette représentation limite le nombre de multiplications au nombre d'indices communs des deux vecteurs considérés.

III.4.2) Optimisation quadratique

Dans les phases de prédiction, il est à chaque fois question de résoudre un système quadratique : le système réduit. Dans Matlab, l'optimiseur est la fonction *quadprog(),*qui cherche le minimum pour un problème spécifié par

$$\min_x \frac{1}{2} x^T Q x + f^T x \quad t.que \quad \begin{cases} A \cdot x \le b \\ Aeq \cdot x = beq \\ lb \le x \le ub \end{cases}$$

Q, A, et *Aeq* sont des matrices, et *f, b, beq*, *lb, ub,* et *x* sont des vecteurs.

x = quadprog (H,f,A,b,Aeq,beq)

Pour notre problème nous avons :

$Q_{ij} = y_i y_j K(x_i, x_j)$, $f(i) = -1$, $Aeq(1,i) = y_i$, $beq(i) = 0$, $lb(i) = 0$ et $ub(i) = c$

III.5) Implémentation de l'algorithme

Au début nous avons choisi arbitrairement $\frac{|B|}{2}$ éléments de la classe $\{+1\}$ et $\frac{|B|}{2}$ éléments de la classe $\{-1\}$. Il est facile de calculer les données du système quadratique sauf la hessienne *Q* qui est une matrice symétrique. La méthode proposée requiert que *Q* soit stockée sous forme d'une matrice triangulaire supérieure droite. Le nombre d'éléments occupés est donné par $\frac{n(n+1)}{2}$ où *n* est le nombre d'éléments qui ont visité l'ensemble *B* ($n \geq |B|$). Pour le kernel nous avons pris le *RBF* avec $\delta = 10$.

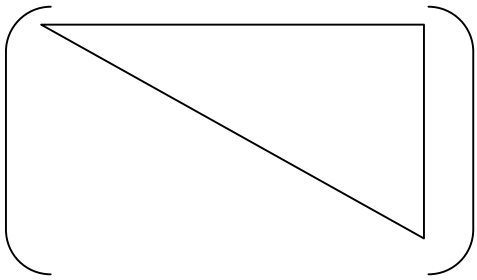

FIG III.4 : *Représentation de la hessienne Q triangulaire*

Pour faire les tests des éléments à remplacer et le test d'arrêt, nous avons utilisé les vecteurs *X* contenant les nouvelles valeurs des variables α_i comme solution de l'optimiseur *quadprog()*, *FX* pour calculer les valeurs $y_i f(x_i)$, et enfin le vecteur *FXN* contenant les valeurs $y_j f(x_j)$ $j \in N$ trié d'un ordre croissant pour chercher α_j $j \in N$ $y_j f(x_j) < y_k f(x_k)$ $\forall k \in N$.

III.5.1) Résultat d'apprentissage et de test

Pour pouvoir comparer notre algorithme avec l'autre proposé par Osuna, ils doivent se dérouler dans les mêmes circonstances, notamment le matériel (CPU, mémoire), l'optimiseur quadratique, techniques de calcul et de stockage de la matrice *Q*,…. On a voulu les comparer avec l'algorithme SVMlight sur le même corpus mais l'optimiseur de ce dernier est basé sur la méthode à points intérieurs (*Interior Points Method : IPM*), qui semble donner de très bons résultats en termes de temps de calcul et de précision de la solution par rapport à *quadprog()* adopté par notre implémentation, ainsi SVMlight utilise le caching pour gérer la matrice de Gram.

	Osuna	**Osuna modifié**	**SVLlight**
Représentation des données	Structures creuses	Structures creuses	Structures creuses
Optimisation quadratique	Quadprog()	Quadprog()	IPM ou Hildreth
Elimination des variables	globale	globale	globale
Chunking	présent	présent	présent
Sélection du Working set	remplacer $\alpha_i = 0 i \in B$ par $j \in N$ $y_j f(x_j) < 1$	remplacer $\alpha_i = 0 i \in B$ par $j \in N$ *t.que* $y_j f(x_j) < y_k f(x_k) < 1 \forall k$	Meilleure Direction admissible
Cache	Matrice triangulaire	Matrice triangulaire	Par ligne
Architecture	Procédurale	Procédurale	Orientée objet
Langage	Matlab	Matlab	C++
Ajout de nouveaux kernels	Par Substitution de fichier source	Par Substitution de fichier source	Par Substitution de fichier source

Tab III.1 : *Comparaison des caractéristiques des algorithmes*

Puisque notre objectif vise l'amélioration de l'algorithme d'Osuna, nous n'avons pas opté pour l'utilisation des techniques qui permettent aux deux algorithmes, en même temps, l'évolution en termes de temps d'apprentissage.

Corpus d'apprentissage

L'ensemble d'apprentissage utilisé est un échantillon de la base de donnée "Reuters-21578" disponible sur http://download.joachims.org/svm_light/examples/example1.tar.gz, cet ensemble contient 1000 exemples positifs (classe +1) et 1000 exemples négatifs (classe -1) dans le fichier « train.dat ». Le fichier « test.dat » contient la base de données de test dont nous avons 300 exemples positifs (classe +1) et 300 exemples négatifs (classe -1).

Résultats

Les tableaux suivants montrent les résultats obtenus par les deux algorithmes d'apprentissage selon les valeurs des paramètres C=1000, δ =10 du kernel RBF et différents échantillons pris de la base d'apprentissage "Reuters-21578". Nous allons toujours initialiser le même *working set* pour les deux algorithmes.

	Time CPU (min)	**Nbr SV**	**Test**
Osuna	5.01	398	96.99 %
Osuna modifié	4.43	387	97.16 %

Tab.III.2 : *Résultats des algorithmes pour 600 exemples d'apprentissage*

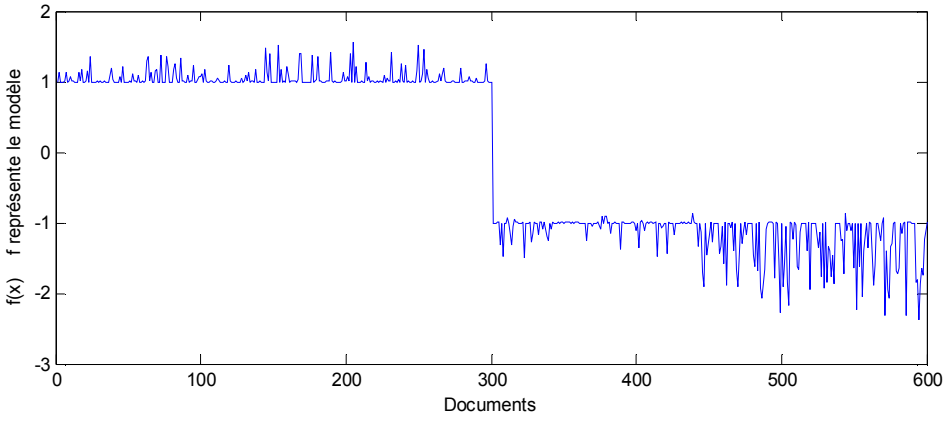

FIG.III.5 : *Application du modèle déduit par l'algorithme Osuna modifié sur l'ensemble d'apprentissage.*

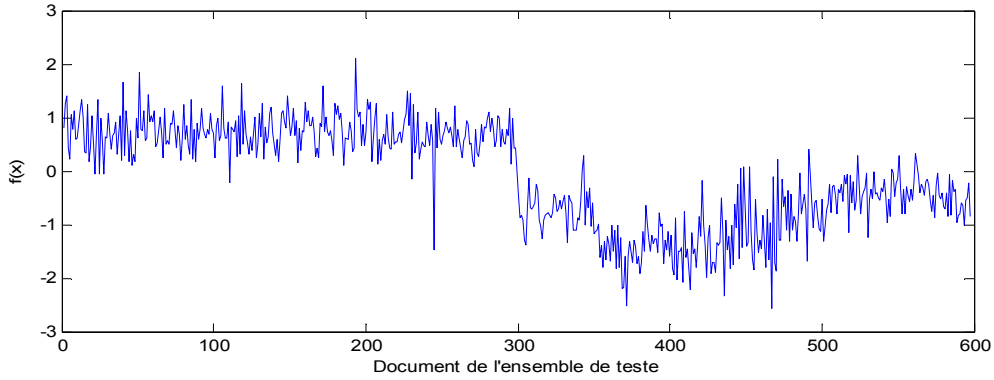

FIG.III.6 : *Application du modèle déduit par l'algorithme Osuna modifié sur l'ensemble de test.*

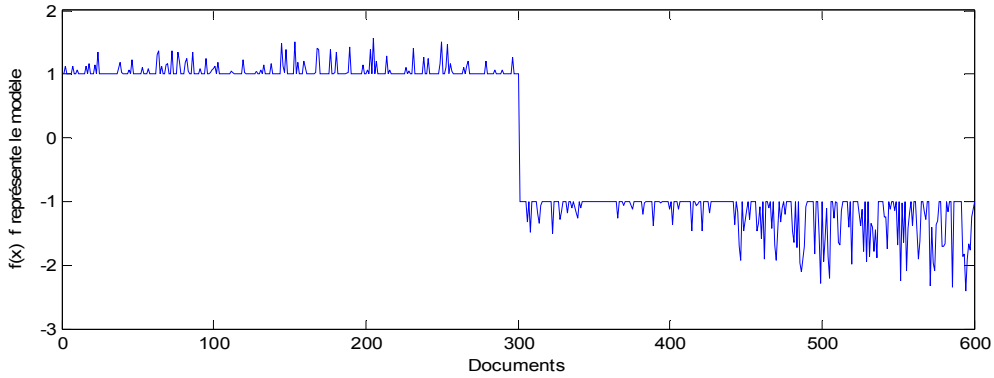

FIG.III.7 : *Application du modèle déduit par l'algorithme Osuna sur l'ensemble d'apprentissage.*

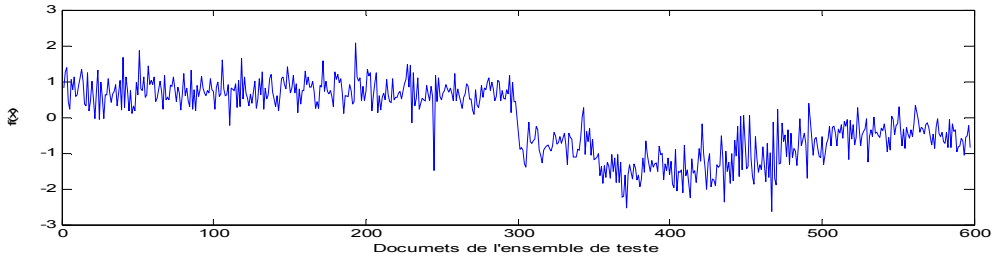

FIG.III.8 : *Application du modèle déduit par l'algorithme Osuna sur l'ensemble de test*

.

	Time CPU (min)	Nbr SV	Test
Osuna	26.27	495	97.16 %
Osuna modifié	15.88	484	97.16 %

Tab.III.3 : *Résultats des algorithmes pour 800 exemples d'apprentissage*

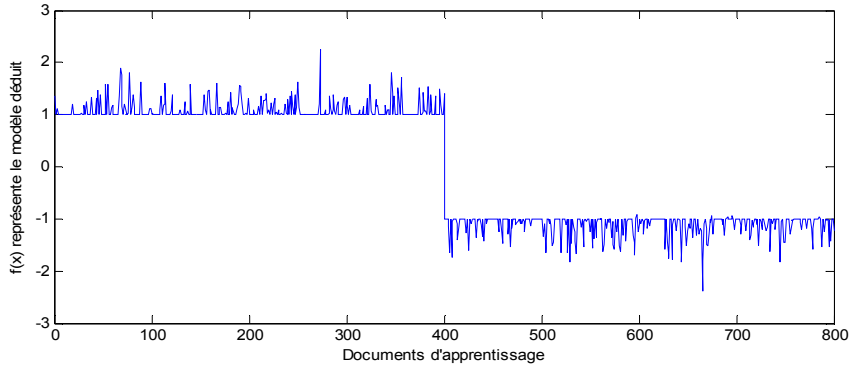

FIG.III.9 : *Application du modèle déduit par l'algorithme Osuna modifié sur l'ensemble d'apprentissage.*

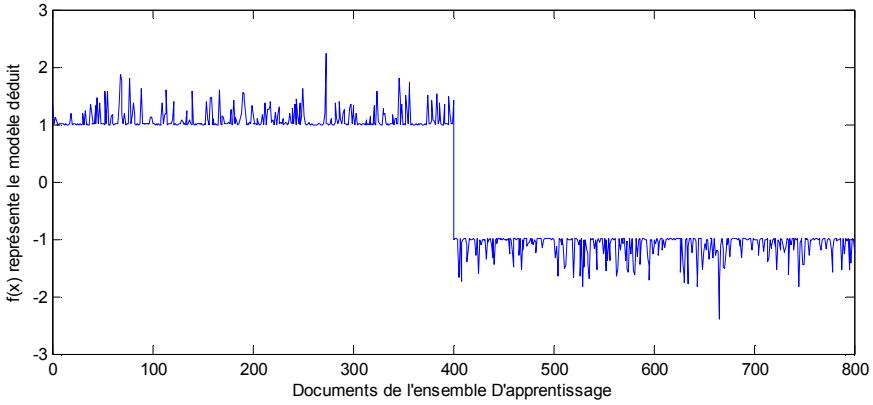

FIG.III.10 : *Application du modèle déduit par l'algorithme Osuna sur l'ensemble d'apprentissage.*

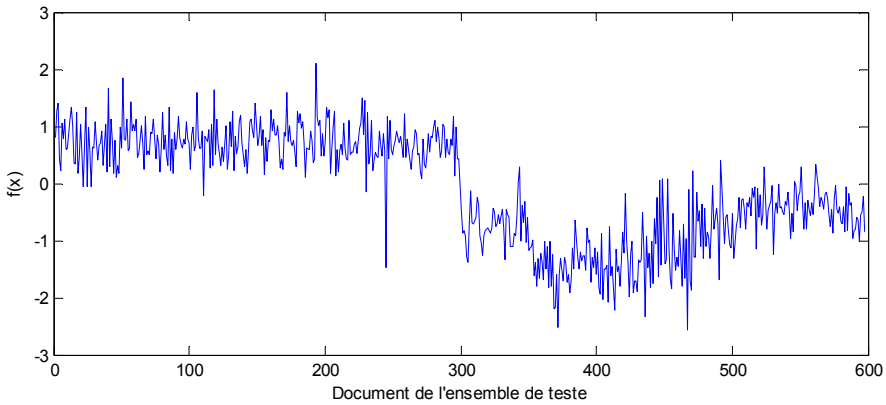

FIG.III.11 : *Application du modèle déduit par les deux algorithmes sur l'ensemble de test.*

III.5.2) Discussion

Nous allons commenter ici les différents résultats obtenus des expériences faites successivement et qui vous ont été présentées précédemment. Ces commentaires reposeront sur les observations des chiffres appartenant aux tableaux de résultats.

Nous avons commencé l'apprentissage avec un ensemble de 600 exemples pour les deux algorithmes dont 300 positifs et 300 négatifs, et qui sera augmenté pour chaque expérience, sous les mêmes valeurs des paramètres. Les tableaux ci-dessus, montrent que le temps d'apprentissage écoulé par l'algorithme Osuna modifié est toujours meilleur que l'algorithme d'origine dont la différence est proportionnelle avec la taille du corpus. Nous remarquons aussi que le modèle déduit donne toujours les mêmes résultats sur l'ensemble de test environ de 97% pour toutes les expériences.

Conclusion et perspectives

L'objectif de notre travail est l'élaboration d'un classifieur de documents par apprentissage en utilisant la méthode « Supports Vectors Machine » (SVM). Nous avons divisé le travail en deux parties dont la première est consacrée à mettre en évidence les différents modèles de représentation des documents, précisément le modèle vectoriel. La deuxième partie porte sur l'apprentissage et la formulation de la classification en théorie, puis la formulation des principes de la méthode SVM qui devient un problème d'optimisation quadratique incalculable par la machine. Dans ce contexte plusieurs approches sont proposées et qui se reposent généralement sur le principe de la décomposition. En particulier, nous avons traité les algorithmes d'Osuna, SMO et SVM^{light} ce dernier est un algorithme itératif qui à chaque itération cherche une direction admissible de descente pour définir le nouveau sous problème quadratique à optimiser en définissant des conditions d'arrêt propre à l'auteur. En se basant sur le même principe de l'algorithme d'Osuna nous avons proposé une nouvelle façon de définir les sous problèmes à optimiser (*working set*). Enfin et dans la phase de l'implémentation nous avons choisi un échantillon du corpus "Reuters-21578" contenant 2000 exemples d'apprentissage et 600 éléments de test, puis nous avons appliqué les deux algorithmes à plusieurs échantillons du corpus en augmentant la taille de l'échantillon pour chaque expérience. Les résultats obtenus montrent que notre stratégie a réussi de réduire le temps d'apprentissage par rapport à l'algorithme d'Osuna, et que le modèle déduit par chaque algorithme a donné les mêmes résultats sur la classification du corpus de test.

Une première perspective à ce travail est l'implémentation de l'algorithme en utilisant la technique de la cache sur la matrice hessienne Q, pour exploiter son influence sur la durée de l'entraînement.

Une deuxième perspective consiste à utiliser des optimiseurs quadratiques qui se basent sur d'autres techniques telle que (*Interior Points Method : IPM*), qui semble donner de très bons résultats en terme de temps de calcul et de précision de la solution.

Bibliographie

[Baeza et al, 1999] Y.Baeza et B.Ribeiro, « *Modern information retrieval* », edition ACM press, 1999

[Besançon, 2002] R.Besançon, « *Intégration de connaissances syntaxiques et sémantiques dans les représentations vectorielles de textes - Application au calcul de similarités sémantiques dans le cadre du modèle DSIR* », Thèse de doctorat - EPFL - Lausanne, 2001.

[Beyer et al, 1999] T.Beyer**,** When is « *nearest neighbor* » meaningful? In proceeding of the International *Conference* on Database Theory (ICDT), pages 217-237.

[Brill, 1992] J.Brill, « *A simple rule based part-of-speech tagger* ». In proceedings of 3rd conference on applied Natural Language Processing, (ANLP)1992.

[Buckley et al, 1992] C. Buckley, G. Salton, J. Allan (Cornell University), « *Automatic retrieval with locality information using SMART*», The first Text Retrieval Conference, page 59-72, 1992.

[Caropreso et al., 2001] M.Caropreso , S.Matwin et F.Sebastiani, « *Independent evaluation of the usefulness of statistical phrases for automated text categorization* ».In A. G. Chin (Ed.), Text Databases and Document Management: Theory ans Practice (ed. pp. 78-102): Idea Group Publishing, Hershey, US. 2001

[Cavnar et al., 1994] Cavnar W., & Trenkle J. N-gram-based text categorization. in Proceedings of SDAIR-94, the 3rd Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, US. pp. 161-175.1994

[Cortes et al, 1995] C. Cortes et V. Vapnik. « *Support vector networks*». Machine Learning, 20:1-25, 1995.

[Daille, 1994] F.Daille, « *Approche mixte pour l'extraction automatique de terminologie : statistiques lexicales et filtres linguistiques* », Thèse de doctorat, Université Paris 7. 1994.

[Derwester et al, 1990] S.Deerwester, T.Dumais, G.Furnas, T.Landauer et R.Hrashman, « *Indexing by latent semantic analysis*», Journal of th american society for information science, 41(6), pages 391-407, 1990.

[Fletcher, 1987] R. Fletcher, « *Practical Methods of Optimization* ». John Wiley et Sons, Inc., 2nd edition, 1987.

[Furnas et al. 1988] Furnas G.W., Deerwester S, Dumais S.T., Landauer T.K., Hrashman R., Streeter L.A. Lochbaum K.E., «*Information retrieval using singular decomposition model of matent semantic structure*». In Proc. Of the 11th Annual ACM SIGIR Conference on research and development in information retrieval, pages 465-480, 1988.

[Gaussier, 2000] Gaussier,« *Traitement automatique des langues* ». C (2000), Recherche d'information en français et traitement automatique des langues, P473.

[Joachims, 1998] T. Joachims, « Making large-scale support vector machine learning practical ». In A. Smola B. Scholkopf, C. Burges, editor, Advances in

Kernel Methods :Support Vector Machines. MIT Press, Cambridge, MA, 1998.

[Karush,1939] W.Karush,«*Minima of functions of several variables with inequalities as side constraints*».Master's thesis, Dept. of Mathematics, Univ. of Chicago, 1939.

[Kuhn et Tucker,1951] W. Kuhn et A. W. Tucker. « *Nonlinear programming* ». In Proc. 2nd Berkeley Symposium on Mathematical Statistics and Probabilistics, pages 481–492, Berkeley, 1951.University of California Press.

[Lewis, 1992a] D.Lewis, «*Representation and learning in information retrieval* ». PHD Thesis, Amherst, MA, USA. 1992.

[Lewis, 1992b] D.Lewis, « *An evaluation of phrasal and clustered representations on text categorisation task* », In proceedings of fifteenth annual International ACMSIGIR conference on Research and development in information retrieval.1992.

[Manning et al, 1999] C.Manning et H.Schütze, «*Foundations of Statistical Natural Language*». Cambridge, Massachusetts: The MIT Press. 1999.

[Mihalcea et al, 2000] Mihalcea et Moldovan, «*Semantic indexing using wordnet senses*». In Proceedings of ACL Workshop on IR& NLP, Hong Kong, 2000.

[Ng et al, 1997] H.Ng, Goh, et K. Low, « *Feature selection, perceptron learning, and a usability case study for text categorisation*». In Proceedings of the Twentieth Annual International, ACM SIGIR on Research and development in information retrieval, 1997.

[Osuna et al. 1997a] E.Osuna, R. Freund, and F. Girosi, « *Training Support Vector Machines: an Application to Face Detection*». Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), New York, 1997 IEEE.

[Osuna et al, 1997b] E.Osuna, R.Freund, and F. Girosi. «*Support vector machines: Training and applications*». A.I. Memo 1602, MIT A. I. Lab., 1997.

[Plantié, 2006] M.Plantié, « *Extraction automatique de connaissances pour la décision multicritère* », Thèse de doctorat - ENSM- SAINT ETIENNE, 2006.

[Platt, 1998] C. Platt, « *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*». Technical Report MSR-TR-98-14, April (1998).

[Porter, 1980] M.F.Porter, « *An algorithm for suffix stripping* ». Program 14, pp 130-137. 1980.

[Quinlan, 1993] J. R. Quinlan, «*C4.5: Programs for Machine Learning*». Morgan Kaufmann, 1993.

[Quinlan, 1996] J. R. Quinlan, « *Bagging, boosting, and C4.5* ». In Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, pages 725–730, Menlo Park, 1996.AAAI Press/MIT Press.

[Rosenblatt, 1958] F.Rosenblatt, «*The perceptron : A probabilistic model for information storage and organization in the brain*». Psychological Review, 65(6) :386–408, 1958.

[Salton , 1975a] G.Salton, « *A vector space model for automatic indexing*». Communicationsof the ACM, 18(11):613–620.

[Salton et al, 1975b] G.Salton , Yang, and Yu, « *A theory of term importance in automatic text analysis*». Journal of the American Society for Information Science and Technology, Volume 26(1) pages 33-44, 1975.

[Salton, 1981] G.Salton, « *A Blueprint for Automatic Indexing* ». SIGIR Forum, 16,(2). 1981.

[Salton, 1983] G.Salton, « *Introduction to modern information retrieval* ». MCGILL M. J., 1983.

[Salton et al, 1988] G.Salton and C.Buckley, Term-weighting «*Approaches in automatic text retrieval* ». Information Processing & Management, Volume 24, Issue 5, Pages 513-523 (1988).

[Savoy, 1999] Savoy, «*A stemming procedure and stopword list for general French corpora* », Journal of the American Society for Information Science, 1999.

[Sebastiani ,1999] F.Sebastiani, «*Machine learning in automated text categorisation*». Technical Report IEI-B4-31-1999, Consiglio Nazionale delle Ricerche, Italy, Pisa, IT, 1999.

[Sebastiani, 2002] F.Sebastiani, « *Machine learning in automated text categorisation*». ACM Computing surveys. 2002.

[Shannon, 1948] C.Shannon, « *A mathematical theory of communication*». Bell Systems Technical Journal(27), pp 379-423 & 623-656. 1948.

[Van Rijsbergen, 1979] Van Rijsbergen, «*Information retrieval*». C. (1979). p25. Butttherewords. London.1979

[Van Rijsbergen 1986] Van Rijsbergen, « *A non-classical logic for information retrieval* », The computer Journal, Vol. 29, No. 6 1986

[Vapnik, 1995] V. Vapnik, «*The Nature of Statistical Learning Theory*». Springer Verlag, New York, 1995.

[Weber, 1998] Weber , «*A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces*». In proceedings of the 24 th international on very large databases (VLDB), pages 194-205, New York, USA.1998.

[Yang et al, 1997] Y.Yang and Pedersen, « *A comparative study on feature selection in text categorisation* ». International Conference on Machine learning. Morgan Cofmann, 1997.

[Zoutendijk, 1970] Zoutendijk, « *Methods of Feasible Directions: a Study in Linear and Non-linearProgramming* ». Elsevier, 1970.