République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université HADJ LAKHDAR – BATNA
Faculté des sciences
Département d'Informatique

N° d'ordre :…….
Série :…….

Mémoire

Présenté en vue de l'obtention du diplôme de Magister en Informatique

Option : Informatique Industrielle

Apprentissage Incrémental

&

Machines à Vecteurs Supports

Présenté par : Marref Nadia

Soutenu le :18 /12 /2013

Devant le jury :

Pr. BILAMI Azeddine,	Université de Batna	Président
Pr. BATOUCHE Mohamed	Université de Constantine	Rapporteur
Pr. ZIDANI Abdelmadjid	Université de Batna	Examinateur
Pr. BENMOHAMED Mohamed	Université de Constantine	Examinateur

Dédicace

A la mémoire de mon père qui a laissé un grand vide après son départ

A ma chère mère

A celui qui m'a toujours soutenu, mon époux

A mes frères et sœurs

A toute ma famille

A mes amies

Remerciements

Tous mes remerciements s'adressent tous d'abord à tout puissant ALLAH, d'avoir guidé mes pas vers le chemin du savoir.

Je tiens à exprimer ma profonde gratitude à Mr Pr Batouche, Professeur à l'université de Constantine, de m'avoir fait l'honneur de diriger mes travaux durant ces années de magister en étant toujours disponible et encourageant , ses conseils , sa rigueur scientifique et ses valeurs humaines m'ont bien guidé.

Je tiens à remercier également Mr Pr Bilami Azeddinne, Maître de conférence à l'université de Batna, d'avoir accepté d'être le président de jury.

Je remercie vivement Mr Pr Zidani Abdelmadjid Maître de conférence à l'université de Batna, et Mr Pr Benmohamed Mohamed Maître de conférence à l'université de Constantine, d'avoir accepté d'être examinateurs.

Je tiens énormément à remercier mes amies et mes collègues pour leurs encouragements.

Pour terminer, je tiens à remercier mon époux et tous les membres de ma famille pour leurs soutiens et leurs encouragements.

Résumé

Les machines à vecteurs supports (Support Vector Machines) sont de nouvelles techniques d'apprentissage statistique proposées par V. Vapnik en 1995. Elles permettent d'aborder des problèmes très divers comme la classification, la régression, la fusion, etc. Depuis leur introduction dans le domaine de la Reconnaissance de Formes (RdF), plusieurs travaux ont pu montrer l'efficacité de ces techniques principalement en traitement d'image.

L'idée essentielle des SVM consiste à projeter les données de l'espace d'entrée (appartenant à deux classes différentes) non-linéairement séparables dans un espace de plus grande dimension appelé espace de caractéristiques de façon à ce que les données deviennent linéairement séparables. Dans cet espace, la technique de construction de l'hyperplan optimal est utilisée pour calculer la fonction de classement séparant les deux classes.

L'apprentissage des SVM se ramène à résoudre un programme quadratique, la mise en œuvre d'un algorithme de SVM est donc coûteuse en temps et mémoire. Pour remédier à ce problème, les travaux plus récents visent à construire des algorithmes de SVM incrémentaux.

Le principe des algorithmes incrémentaux est de ne charger qu'un petit bloc de données en mémoire à la fois, de construire un modèle partiel et de le mettre à jour en chargeant consécutivement des blocs de données. Cela permet de traiter de très grandes quantités de données sans difficulté de mémoire sur une machine standard. Un algorithme incrémental peut donner des résultats approximatifs ou identiques à ceux obtenus par un algorithme d'apprentissage chargeant une seule fois l'ensemble entier des données.

Dans ce mémoire un nouvel algorithme incrémental est proposé pour résoudre un problème d'un SVM binaire, l'algorithme est testé sur une base de données artificielle, et sur des paires de chiffres manuscrits extraites à partir de la base de données standard MNIST.

MOTS CLETS : Apprentissage incrémental, Machines à Vecteurs Supports, Apprentissage statistique, Classification, Apprentissage hors ligne.

Abstract

The Support Vector Machines (SVM) are new techniques of the statistical learning theory proposed by Vapnick in 1995, they can address very different problems as recognition, regression etc. since their introduction into the field of pattern recognition several works have shown the efficiency of these techniques mainly in image processing.

The basic idea of SVM is to project the input vectors into high-dimensional feature space using a non-linear transformation chosen a priory. In this space an optimal Separating Hyperplane is considered,

SVM learning is reduced to solving a quadratic program, implementation of an algorithm for SVM is time consuming and memory. To remedy this problem, more recent works aimed at building incremental algorithm SVM.

The principle of incremental algorithms is to only load a small block of data in memory at once, to build a partial model and updating of the loading of the data blocks sequentially. It can handle large amounts of data without memory difficulty on a standard machine. an incremental algorithm can give approximate results or identical to those obtained by a learning charging once the entire set of data.

In this thesis a new incremental algorithm is proposed, for solving a binary problem of SVM, The algorithm is tested on a database artificial and on pairs of handwritten digits extracted from the standard database MNIST.

Key words: Incremental learning, support vector machines , statistic learning , classification , batch learning.

ملخص

تعتبر تقنية ألة الأشعة الحاملة من أكفأ طرق التلقين الإحصائي، حيث عرفت خلال العشرية الأخيرة تطورا كبيرا من الناحيتين النظرية و التطبيقية. تعتمد هذه التقنية على قواعد نظرية صلبة مبنية على تعظيم الهوة ، مما يمنحها قدرة كبيرة على التعميم. و قد تم استخدام طريقة آلة الأشعة الحاملة بنجاح في العديد من المجالات، كالتعرف الالي على صور الأشخاص ،و على الأصوات و كذا الكتابة اليدوية.

قمنا في هذه الأطروحة باقتراح طريقة جديدة للتمرن خاصة بتقنية الة الاشعة الحاملة حيث تعمل بشكل تزايدي.

لقد تم اختبار الطريقة المقترحة على قواعد بيانات اصطناعية و حقيقية واسعة الاستخدام (الحروف اليدوية وقد تم تحميلهم من قاعدة المعطياتUSPS) مدعمة بتحليلات تبين محاسنها و حدودها. النتائج كانت مشجعة و تفتح أفاقا جديدة للبحث.

الكلمات المفتاحية : ألة الأشعة الحاملة , التلقين الإحصائي، التلقين التزايدي .

Table des matières

Chapitre2 : Les Machines à Vecteurs Supports

Chapitre3 : Mise en œuvre des machines à vecteurs de supports

Liste des Figures

Liste des tableaux

Introduction générale

La croissance impressionnante que le réseau Internet a connue ces deux dernières décennies a renforcé le besoin de techniques permettant l'extraction de connaissances à partir des données. La classification automatique en est une des méthodes principales.

Les méthodes de l'apprentissage automatique forment une classe de techniques attrayantes pour l'accomplissement des tâches d'extraction de connaissances évoquées. Bien choisis, ces outils peuvent être amenés à accompagner, voire à remplacer l'opérateur humain. Par exemple, aux Etats-Unis, les services postaux (USPS) ont automatisé l'aiguillage du courrier en fonction du code postal se trouvant sur les enveloppes.

Les algorithmes d'apprentissage automatique ont été appliqués à divers domaines, notamment le traitement du langage naturel et de la parole, la reconnaissance de l'écriture manuscrite, la vision robotisée, la fouille de données, les moteurs de recherche sur Internet, le diagnostic médical, la bioinformatique, etc. Les techniques d'apprentissage ont ainsi joué un rôle crucial dans des applications qui vont de la mise au point de médicaments à l'analyse de grands réseaux de télécommunication.

Le problème principal de la classification est de construire un classificateur à partir d'un ensemble de données. Un grand nombre de données est généralement nécessaire pour créer un système de classification avec un taux de reconnaissance assez élevé. Pratiquement, il est souvent difficile d'obtenir cette quantité de données dans certains types d'application. Pour cette raison, l'adaptation permanente des classificateurs devient une alternative très intéressante aux apprentissages « batch » classiques. De plus, dans plusieurs contextes applicatifs, le classificateur doit tenir compte de nouvelles classes et les intégrer dans sa base de connaissances et dans le processus de classification. Cela amène le besoin de définir des classificateurs de type « auto-évolutifs ».

Nous pouvons citer au moins trois situations où les algorithmes de classification classiques présentent des limitations. Une première situation où l'ensemble des données à traiter est de taille trop importante et une seconde situation dans laquelle la base d'apprentissage est incomplète. La troisième situation concerne l'apprentissage de concepts dynamiques.

Dans la première situation, les calculs deviendront trop importants voire irréalisables, à cause de la complexité algorithmique. Dans ce cas des solutions basées sur des méthodes sélectives existent et visent à utiliser uniquement les données les plus pertinentes pour effectuer l'apprentissage. La difficulté réside évidemment dans la mise en œuvre du critère de sélection qui risque d'écarter certaines informations pertinentes.

Dans la seconde situation, les données sont acquises en ligne, de façon séquentielle. Toutes les données ne sont donc pas connues a priori, elles deviennent disponibles au fur et à mesure pour enrichir cette base. C'est le cas dans les applications en ligne où l'objectif consiste à déterminer le modèle de classification avec les données disponibles à chaque instant, sans attendre que la base d'apprentissage soit complétée. Ceci amène à redéfinir le modèle de classification à chaque fois qu'une nouvelle donnée se présente : C'est la **mise à jour** du modèle.

La troisième situation (l'apprentissage de concepts dynamiques) concerne des tâches de classification pour lesquelles l'association entre un objet donné et sa classe peut changer au cours du temps. Ce genre de situation se rencontre par exemple dans le problème de la détermination automatique d'articles pertinents pour un lecteur dont les centres d'intérêts peuvent varier.

La stratégie permettant d'effectuer cette mise à jour, consiste à utiliser les techniques d'**apprentissage incrémental** développées avec des **règles de mise à jour récursives**. Grâce à ces techniques, les informations portées par les nouvelles données sont incorporées séquentiellement dans le modèle de classification sans réutiliser (ou très peu) les anciennes données. L'estimation se fait de façon récursive en utilisant les nouvelles informations et les connaissances déjà extraites sous forme de modèles de classes.

Les machines à vecteurs supports sont un ensemble de techniques d'apprentissage destinées à résoudre des problèmes de discrimination, c'est à dire décider à quelle classe appartient un échantillon, ou de régression, c'est-à-dire prédire la valeur numérique d'une variable. 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 du SVM.

La méthode cherche alors l'hyperplan qui sépare les exemples positifs des exemples négatifs, en garantissant que la marge entre le plus proche des positifs et des négatifs soit maximale. Intuitivement, cela garantit un bon niveau de généralisation car de nouveaux exemples pourront ne pas être trop similaires à ceux utilisés pour trouver l'hyperplan mais être tout de même situés franchement d'un coté ou l'autre de la frontière. Un autre intérêt est la sélection de Vecteurs Supports qui représentent les vecteurs discriminants grâce auxquels est déterminé l'hyperplan. Les exemples utilisés lors de la recherche de l'hyperplan ne sont alors plus utiles et seuls ces vecteurs supports sont utilisés pour classer un nouveau cas.

La méthode SVM appartient à une classe de méthodes dite méthodes des noyaux, qui consistent à projeter les données qui sont non linéairement séparables dans un autre espace de dimension plus élevée où elles peuvent le devenir. La méthode SVM développée par Vladimir Vapnik, se base sur le principe de risque structurel qui lui confère un pouvoir fort de généralisation [32]. La minimisation du risque structurel permet d'éviter le phénomène de sur-apprentissage des données à la convergence du processus d'apprentissage.

De nombreux travaux ont démontré la supériorité du SVM par rapport aux méthodes discriminantes classiques. Sa robustesse vis-à-vis de la dimensionnalité des données et son pouvoir accru de généralisation font que le SVM est nettement plus avantageux.

L'objet de ce mémoire est l'étude et l'implémentation des algorithmes incrémentaux dédiés aux SVM ainsi que la proposition et l'implémentation d'un nouvel algorithme incrémental appliqué aux données de concept statique c'est-à-dire que les données d'apprentissage ne changent pas au cours du temps.

Ce mémoire est organisé en quatre chapitres. Le premier chapitre traite les notions fondamentales de l'apprentissage automatique et passer en revue les différents types d'apprentissage. Il décrit également la théorie de l'apprentissage statistique dont dérive le principe du risque structurel. Le deuxième chapitre décrit en détail la théorie et les fondements des machines à vecteurs supports.

Pour résoudre le problème d'optimisation des machines à vecteurs supports, une description détaillée de la mise en œuvre de deux algorithmes hors ligne est présentée dans le troisième chapitre. Dans ce dernier, nous présentons également les résultats d'expérimentation de ces deux algorithmes. Ainsi, en premier lieu, nous présentons les résultats de simulation dans le cas d'une classification binaire sur des bases de données standards. Il s'ensuit une deuxième partie qui est consacrée aux expérimentations effectuées sur les deux bases de données de chiffres manuscrits USPS et MNIST. Une sélection manuelle des hyper-paramètres est utilisée pour différents noyaux.

Le quatrième chapitre présente notre contribution qui est la proposition et l'implémentation d'un nouvel algorithme incrémental. Nous présentons en premier lieu une revue de la littérature des algorithmes incrémentaux dédiés aux SVMs. Ensuite, nous décrivons en détails le principe d'un algorithme incrémental existant duquel nous nous sommes inspirés. Les différentes étapes de notre algorithme incrémental sont alors décrites, et enfin une phase de test de ces deux algorithmes incrémentaux est réalisée. Une première expérience est effectuée sur une base de données artificielle à deux classes et de dimension 2. Une seconde expérience est réalisée sur les chiffres manuscrits.

Du moment que les deux algorithmes incrémentaux ne traitent pas le cas multi-classes, nous avons choisi de les appliquer aux paires des chiffres les plus fréquemment confondus. Différents noyaux sont utilisés avec différents paramètres pour évaluer les performances de ces deux algorithmes incrémentaux. Enfin, nous comparons les résultats obtenus avec ceux des algorithmes hors ligne fournis au troisième chapitre.

Enfin, une conclusion générale et des perspectives de travail viennent clôturer ce mémoire.

Chapitre 1

L'apprentissage Automatique

1.1 Introduction

L'**apprentissage automatique (ou artificiel)** (*machine-learning* en anglais) est un des champs d'étude de l'intelligence artificielle. Commençons par la définition de l'AAAI[1] et celle fournie dans l'avant-propos de (Cornuéjols *et al.*,2002).

L'apprentissage artificiel fait référence à la capacité d'un système à acquérir et intégrer de façon autonome des connaissances[2].

Cette notion englobe toute méthode permettant de construire un modèle de la réalité à partir de données, soit en améliorant un modèle partiel ou moins général, soit en créant complètement le modèle [22].

L'apprentissage automatique fait référence au développement, l'analyse et l'implémentation de méthodes qui permettent à une machine (au sens large) d'évoluer et de remplir des tâches associées à une intelligence artificielle grâce à un processus d'apprentissage. Cet apprentissage permet d'avoir un système qui s'optimise en fonction de l'environnement, les expériences et les résultats observés [22].

Voyons quelques exemples. La capacité d'apprentissage est une caractéristique des êtres vivants. De la naissance à l'âge adulte, les êtres vivants acquièrent de nombreuses capacités qui leur permettent de survivre dans leur environnement. L'apprentissage d'un langage, de l'écriture et de la lecture sont de bons exemples des capacités humaines, et des phénomènes mis en jeu : apprentissage par cœur, apprentissage supervisé par d'autres êtres humains, apprentissage par généralisation.

Une application classique de l'apprentissage artificiel est la reconnaissance de caractères manuscrits, tels qu'ils apparaissent sur une enveloppe. La difficulté tient au fait que la variété des formes rencontrée est infinie. L'apprentissage par cœur n'est pas possible, et il faut donc être capable de généraliser à partir d'un ensemble d'exemples de caractères.

[1] American Association for Artificial Intelligence, http://www.aaai.org.

[2] Machine learning refers to a system capable of the autonomous acquisition and integration of knowledge.

- **Quelques mots d'Histoire...**

L'apprentissage artificiel est une discipline jeune, à l'instar de l'Informatique et de l'Intelligence artificielle. Il se situe au carrefour d'autres disciplines: philosophie, psychologie, biologie, logique, mathématique [22].les premières études remontent à des travaux de statistique dans les années 1920. C'est après la seconde guerre mondiale que les premières expériences deviennent possibles. Se développent ensuite dans les années 1960 les approches connexionnistes avec des perceptrons, et la reconnaissance des formes. La mise en évidence des limites du perceptron simple arrête toutes les recherches dans ce domaine jusqu'à la renaissance dans les années 1980. Les années 1970 sont dominées par des systèmes mettant l'accent sur les connaissances, les systèmes experts, Les limites de tels systèmes se font sentir dans les années 1980, pendant lesquelles a lieu le retour du connexionnisme avec un nouvel algorithme d'apprentissage [22].

Les mathématiciens commencèrent à s'éloigner du cadre cognitif de l'apprentissage pour envisager le problème sous l'angle de l'optimisation, pendant qu'apparaissaient de nouvelles méthodes comme les arbres de décision ou l'induction de programmes logiques. L'influence de la théorie statistique de l'apprentissage s'est réellement fait sentir dans les années 1990, avec l'ouvrage de Vapnik [31].

Dans le présent chapitre, nous commencerons par introduire les concepts fondamentaux du Machine Learning supervisé dans le cas de la classification. Nous validerons ensuite cette approche en effectuant une analyse statistique de l'apprentissage, principalement basée sur la théorie de Vapnik [31]. Finalement, nous présenterons les principales étapes de la mise en pratique de la méthodologie ML. Dans la dernière section de ce chapitre nous présentons les différentes méthodes de classification.

1.2 Concepts et Sources de l'apprentissage automatique

L'apprentissage de l'être humain se compose de plusieurs processus qu'il est difficile précisément à décrire. Les facultés d'apprentissage chez l'humain lui ont conféré un avantage évolutif déterminant pour son développement [22].

Par " faculté d'apprendre " on entend un ensemble d'aptitudes comme :

- ✓ L'obtention de la capacité de parler en observant les autres.
- ✓ L'obtention de la capacité de lire, d'écrire, d'effectuer des opérations arithmétiques et logiques avec l'aide d'un tuteur.
- ✓ L'obtention d'habilités motrices et sportives en s'exerçant.

1.2.1 Qu'est-ce que l'apprentissage automatique

La faculté d'apprendre de ses expériences passées et de s'adapter est une caractéristique essentielle des formes de vies supérieures. Elle est essentielle à l'être humain dans les premières étapes de la vie pour apprendre des choses aussi fondamentales que reconnaître une voix, un visage familier, apprendre à comprendre ce qui est dit, à marcher et à parler [6].

L'apprentissage automatique est une tentative de comprendre et reproduire cette faculté d'apprentissage dans des systèmes artificiels. Il s'agit, très schématiquement, de concevoir des algorithmes capables, à partir d'un nombre important d'exemples (les *données*

correspondant à "l'expérience passée"), d'en assimiler la nature afin de pouvoir appliquer ce qu'ils ont ainsi appris aux cas futurs [6].

1.2.2 Définition

Un programme d'ordinateur est capable d'apprendre à partir d'une expérience E et par rapport à un ensemble T de tâches et selon une mesure de performance P, si sa performance à effectuer une tâche de T, mesurée par P, s'améliore avec l'expérience E[22].

1.2.3 Modélisation

L'apprentissage automatique d'une machine toujours concerne un ensemble de tâches concrètes - T. Pour déterminer la performance de la machine, on utilise une mesure de la performance P. La machine peut avoir à l'avance un ensemble d'expérience E ou elle va enrichir cet ensemble plus tard.

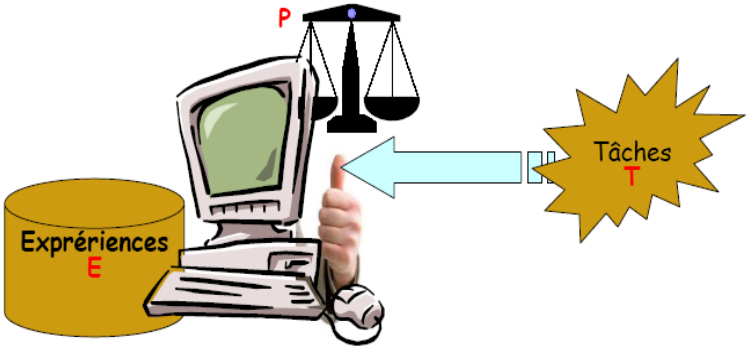


Figure 1.1 : Schéma de modélisation d'une machine d'apprentissage

Donc, l'apprentissage automatique pour la machine est qu'avec l'ensemble de tâches T que la machine doit réaliser, elle utilise l'ensemble d'expériences E telle que sa performance sur T est améliorée.

1.2.4 Domaines d'applications de l'apprentissage automatique:

L'apprentissage automatique s'applique à un grand nombre d'activités humaines et convient en particulier au problème de la prise de décision automatisée. Il s'agira, par exemple:

- D'établir un diagnostic médical à partir de la description clinique d'un patient;
- De donner une réponse à la demande de prêt bancaire de la part d'un client sur la base de sa situation personnelle;
- De déclencher un processus d'alerte en fonction de signaux reçus par des capteurs ;
- De la reconnaissance des formes;
- De la reconnaissance de la parole et du texte écrit;
- De contrôler un processus et de diagnostiquer des pannes;

1.2.5 Mémoriser n'est pas généraliser

Le terme **apprentissage** dans la langue courante est ambigu. Il désigne aussi bien l'apprentissage "par cœur" d'une poésie, que l'apprentissage d'une tâche complexe telle que la lecture. Clarifions la distinction [6] :

- Le premier type d'apprentissage correspond à une simple mémorisation. Or les ordinateurs contemporains, avec leurs mémoires de masse colossales, n'ont aucune difficulté à mémoriser une encyclopédie entière, sons et images inclus.
- Le second type d'apprentissage se distingue fondamentalement du premier en cela qu'il fait largement appel à notre faculté de généraliser. Ainsi pour apprendre à lire, on doit être capable d'identifier un mot écrit d'une manière que l'on n'a encore jamais vue auparavant.

1.3 Types d'apprentissage

Les algorithmes d'apprentissage peuvent se catégoriser selon le mode d'apprentissage qu'ils emploient :

1.3.1 L'apprentissage supervisé

Si les classes sont prédéterminées et les *exemples* connus, le système apprend à classer selon un modèle de classement ; on parle alors d'apprentissage supervisé (ou d'analyse discriminante).

Un expert (ou oracle) doit préalablement correctement étiqueter des exemples. L'apprenant peut alors trouver ou approximer la fonction qui permet d'affecter la bonne « étiquette » à ces exemples. Parfois il est préférable d'associer une donnée non pas à une classe unique, mais une probabilité d'appartenance à chacune des classes prédéterminées (on parle alors d'apprentissage supervisé probabiliste). L'analyse discriminante linéaire ou les SVM sont des exemples typiques. Autre exemple : en fonction de *points communs* détectés avec les symptômes d'autres patients connus (les « exemples »), le système peut catégoriser de nouveaux patients au vu de leurs analyses médicales en risque estimé (probabilité) de développer telle ou telle maladie.

1.3.2 L'apprentissage non-supervisé

Quand le système ou l'opérateur ne dispose que d'exemples, mais non d'étiquettes, et que le nombre de classes et leur nature n'ont pas été prédéterminés, on parle d'apprentissage non supervisé (ou clustering). Aucun expert n'est disponible ni requis. L'algorithme doit découvrir par lui-même la structure plus ou moins cachée des données.

Le système doit ici dans l'espace de description (la somme des données) cibler les données selon leurs attributs disponibles, pour les classer en groupe homogènes d'exemples. La similarité est généralement calculée selon la fonction de distance entre paires d'exemples. C'est ensuite à l'opérateur d'associer ou déduire du sens pour chaque groupe. Divers outils mathématiques et logiciels peuvent l'aider. On parle aussi d'analyse des données en régression. Si l'approche est probabiliste (c'est à dire que chaque exemple au lieu d'être classé dans une seule classe est associé aux probabilités d'appartenir à chacune des classes), on parle alors de *« soft clustering »* (par opposition au *« hard clustering »*) [22].

Exemple : Un épidémiologiste pourrait par exemple dans un ensemble assez large de victimes de cancers du foie tenter de faire émerger des hypothèses explicatives, l'ordinateur pourrait

différencier différents groupes, qu'on pourrait ensuite associer par exemple à leur provenance géographique, génétique, à l'alcoolisme ou à l'exposition à un métal lourd ou à une toxine telle que l'aflatoxine.

1.3.3 L'apprentissage semi-supervisé

Effectué de manière probabiliste ou non, il vise à faire apparaitre la distribution sous-jacente des *« exemples »* dans leur espace de description. Il est mis en œuvre quand des données (ou *« étiquettes »*) manquent... Le modèle doit utiliser des exemples *non-étiquetés* pouvant néanmoins renseigner.

Exemple : En médecine, il peut constituer une aide au diagnostic ou au choix des moyens les moins onéreux de tests de diagnostics.

1.3.4 L'apprentissage partiellement supervisé (probabiliste ou non)

Quand l'étiquetage des données est partiel. C'est le cas quand un modèle énonce qu'une donnée n'appartient pas à une classe *A*, mais peut-être à une classe *B* ou *C* (*A*, *B* et *C* étant 3 maladies par exemple évoquées dans le cadre d'un diagnostic différentiel).

1.3.5 L'apprentissage par renforcement

L'algorithme apprend un comportement étant donné une observation. L'action de l'algorithme sur l'environnement produit une valeur de retour qui guide l'algorithme.

1.4 Les algorithmes utilisés

- Les machines à vecteurs support
- Le boosting
- Les réseaux de neurones pour un apprentissage supervisé ou non-supervisé
- La méthode des k plus proches voisins pour un apprentissage supervisé
- Les arbres de décision
- Les méthodes statistiques comme par exemple le modèle de mixture gaussienne
- La régression logistique
- L'analyse discriminante linéaire
- La logique floue
- Les algorithmes génétiques et la programmation génétique

Ces méthodes sont souvent combinées pour obtenir diverses variantes d'apprentissage. L'utilisation de tel ou tel algorithme dépend fortement de la tâche à résoudre (classification, estimation de valeurs, etc.).

1.5 Facteurs de pertinence et d'efficacité

La qualité de l'apprentissage et de l'analyse dépendent du besoin en amont et *a priori* compétence de l'opérateur pour préparer l'analyse. Elle dépend aussi de la complexité du modèle (spécifique ou généraliste) et de son adaptation au sujet à traiter. Enfin, la qualité du travail dépendra aussi du mode (de mise en évidence visuelle) des résultats pour l'utilisateur

final (un résultat pertinent pourrait être caché dans un schéma trop complexe, ou mal mis en évidence par une représentation graphique inappropriée). Avant cela, la qualité du travail dépendra de facteurs initiaux contraignants, liées à la base de données :

1. **Nombre d'exemples :** moins il y en a plus l'analyse est difficile, mais plus il y en a plus le besoin de mémoire informatique est élevé et plus longue est l'analyse.

2. **Nombre et qualité des attributs :** décrivant ces exemples (La distance entre deux "exemples" numériques (prix, taille, poids, intensité lumineuse, intensité de bruit, etc) est facile à établir, celle entre deux attributs catégoriels (couleur, utilité, est plus délicate)

3. **Pourcentage de données renseignées** et manquantes

4. **« Bruit »** ; Le nombre et la « *localisation* » des valeurs douteuses (erreurs) ou naturellement non conformes au modèle de distribution générale des « *exemples* » sur leur espace de distribution

1.6 Théorie de l'apprentissage statistique

1.6.1 Introduction

La théorie de l'apprentissage statistique se situe à la frontière de plusieurs disciplines, incluant bien évidemment

- la statistique,
- la théorie de l'information et
- l'analyse fonctionnelle.

Dans cette théorie on essaye de concevoir des machines basées sur des algorithmes capables de généralisation, c'est-à-dire ayant de bonnes performances dans des situations non apprises à l'avance.

Le sujet de l'apprentissage statistique a été considéré par Vapnik [31] comme étant un problème d'inférence statistique basé sur un nombre limité d'observations. Le principe d'induction automatique qui constitue le raisonnement fondamental de l'apprentissage statistique, a pour but de créer des systèmes automatiques pouvant passer d'observations particulières à des lois générales [31].

Cette approche est innovante par rapport aux statistiques classiques puisqu'elle fournit des bornes non-asymptotiques sur la confiance de l'estimation de l'erreur de généralisation du modèle par l'erreur empirique.

1.6.2 La classification en théorie

1.6.2.1 Modèle général

Le modèle général du problème d'apprentissage à partir d'un échantillon d'observations est composé de trois parties :

- **Un environnement** : il engendre des formes x_i tirées indépendamment et de manière identiquement distribuée selon la loi de probabilité D_x fixe mais inconnue d'un espace d'entrée X.
- **Un oracle ou superviseur** : il retourne pour chaque forme x_i une réponse désirée ou étiquette (label) u_i.
- **Un apprenant** : Il est capable de réaliser une fonction h d'un espace d'hypothèses H qui vérifie $y_i = h(x_i)$ ou y_i est sa sortie.

  La recherche de la fonction désirée dans F est basée sur un échantillon d'apprentissage S_m tel que $S_m = \{ (x_1, u_1), \dots\dots, (x_m, u_m) \}$

La figure 1.2 illustre ces trois modules.

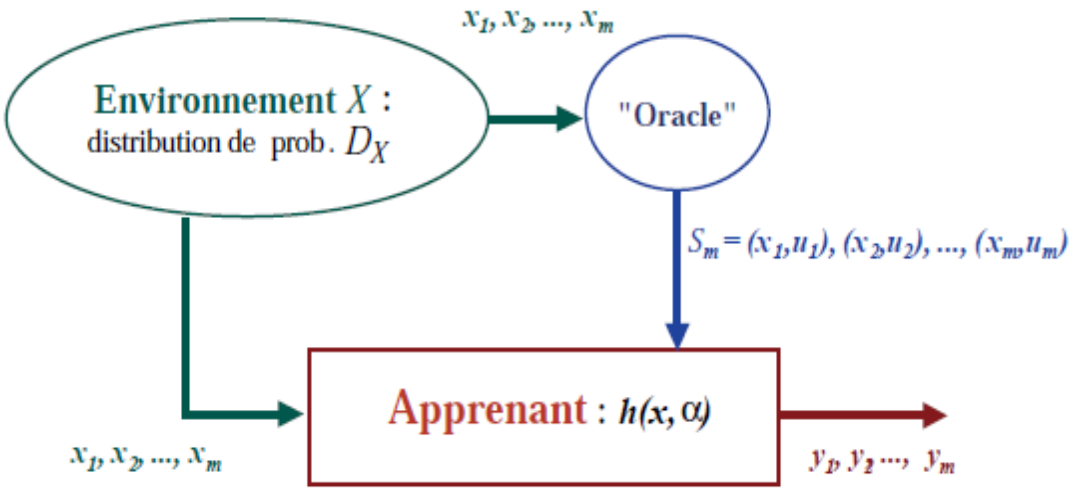


Figure 1.2 : Les modèles d'un système d'apprentissage[22].

1.6.2.2 Formulation

La tâche qu'un classificateur doit effectuer peut être exprimée par une fonction que l'on appelle fonction de décision [4][82]

$$f : x \longrightarrow u$$

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

Dans la suite de ce chapitre, nous nous limiterons à la classification binaire. Dans ce cas, l'ensemble u correspond à {+1, -1} La plupart du temps on interprétera +1 et -1 respectivement comme l'appartenance et la non-appartenance à une catégorie déterminée.

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 [4].

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 [4].

Jusqu'a présent nous n'avons fait que définir le domaine et l'image de la fonction à estimer. On ne peut bien entendu pas accepter n'importe quelle fonction de cette forme. Nous devons en quelque sorte imposer que la fonction représente bien la relation entre les objets et leurs catégories. Pour ce faire nous avons besoin de modéliser le processus selon lequel les données sont générées et d'introduire une fonction de coût indiquant à quel point notre fonction f s'écarte de ce processus.

1.6.2.3 Fonction d'erreur et risque

Si nous disposons de m exemples bien classés $(x_1, u_1), \ldots\ldots, (x_m, u_m)$, une première approche pour déterminer les performances d'un classificateur est de comparer ses prédictions avec les classes u_i attendues. A cette fin, on introduit une fonction d'erreur.

- **Définition (fonction d'erreur)**

Soit le triplet $(x, u, f(x)) \in X \times U \times U$ ou x est un objet, u sa catégorie et f(x) la prédiction du classificateur. Toute fonction C : $X \times U \times U \rightarrow [0, \infty]$ telle que C(x, u, u) = 0 est appelée fonction d'erreur [4].

Par la suite nous utiliserons une fonction d'erreur simple et bien adaptée à la classification binaire

$$E(x, u, f(x)) = \frac{1}{2} |f(x) - u| \qquad (1.1)$$

Nous introduisons à présent la notion de risque qui représente l'erreur moyenne commise sur toute la distribution P(x; u) par la fonction f(x) [4] :

$$R[f] = \int \frac{1}{2} |f(x) - u| dp(x, u) \qquad (1.2)$$

Nous sommes à présent en mesure d'introduire un critère selon lequel la fonction f devra être optimale :

$$f^{opt} = arg_f \min(R[f]) \qquad (1.3)$$

En d'autres termes f^{opt} devra être telle que l'erreur moyenne sur toute la distribution soit minimale.

- **Remarque**

Le critère que nous venons de formuler est malheureusement inutilisable tel quel. En effet pour calculer le risque nous devrions disposer d'une estimation de la distribution P(x; u), ce qui n'est pas le cas. L'étude statistique de la section 1.6.3 va nous permettre d'approcher le risque via une borne supérieure.

1.6.2.4 Machine d'apprentissage

On désigne par machine d'apprentissage, une machine dont la tâche est d'apprendre une fonction au travers d'exemples. Une machine d'apprentissage est donc définie par la classe de fonctions F qu'elle peut implémenter [4]. Dans notre cas, ces fonctions sont des fonctions de décision.

1.6.2.5 Risque empirique

Si on dispose d'une machine d'apprentissage et d'un ensemble de m exemples, on peut exprimer le risque empirique d'une fonction $f_{\bar{\gamma}}$:

$$R_{emp}\left[f_\gamma\right] = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left|f_\gamma(x_i) - u_i\right| \quad (1.4)$$

De manière similaire à la section précédente, on peut dériver un critère de sélection de la fonction optimale f^n :

$$f^n = f_{\gamma^*}\ telle\ que\ \ \gamma^* = arg_\gamma \min(R_{emp}\ \left[f_\gamma\right]) \quad (1.5)$$

Ce critère est appelé minimisation du risque empirique (MRE) [4].

1.6.3 Analyse statistique de l'apprentissage

1.6.3.1 Insuffisance de minimisation du risque empirique (MRE)

On peut légitimement se demander si le critère de minimisation du risque empirique mène toujours à un classificateur possédant un bon pouvoir de généralisation.

Si on n'impose aucune restriction sur la classe F on peut toujours trouver une fonction f qui se comporte bien vis à vis du risque empirique, sans pour autant assurer une bonne généralisation sur de nouveaux exemples.

La restriction des fonctions implémentables nous confronte à un dilemme que les statisticiens appellent biais-variance [33].

Dans la terminologie apprentissage automatique, on l'appelle surgénéralisation, et apprentissage par cœur, concrètement cela dit deux choses :

- D'une part, en limitant fortement la taille de l'ensemble F , on tend à expliquer la relation entre les objets et leurs classes de manière trop grossière (sur généralisation). Dans ce cas le risque empirique sera élevé mais le modèle ne collera pas aux exemples de trop près (voire figure 1.3 (b)) .
- D'autre part, si on admet un grand nombre de fonctions, la relation sera modélisée de manière complexe et le bruit associé aux mesures risque également d'être appris. On parle souvent d'apprentissage par cœur parce que le classificateur aura un risque empirique très faible mais ses performances sur d'autres jeux de données seront mauvaises (voire figure 1.3 (a)).

On voit que la "taille" ou "complexité" de l'ensemble de fonctions F joue un rôle fondamental. Ce que l'on nomme la *capacité* $h(F)$ est une mesure de cette complexité.

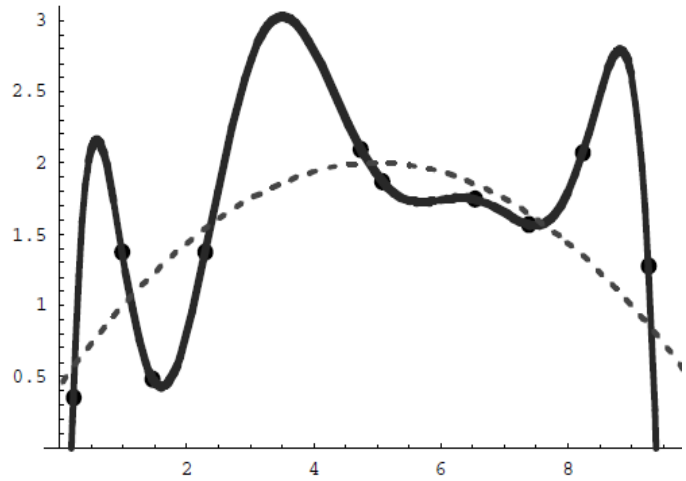


Figure 1.3 (a) : l'apprentissage par cœur (overfitting)

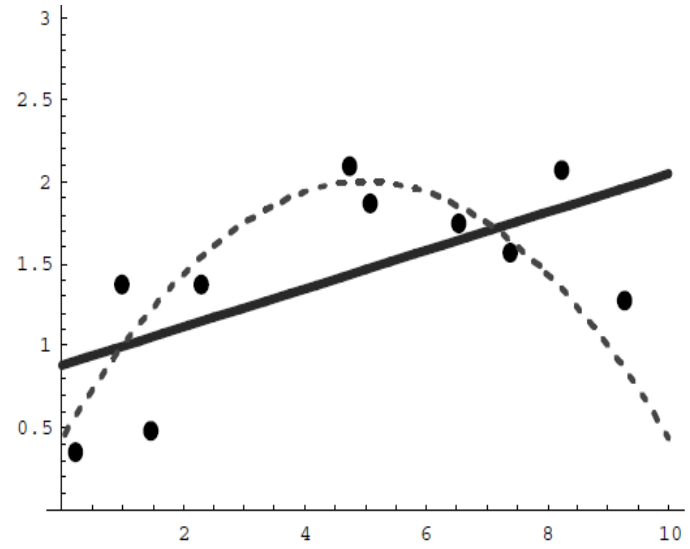


Figure 1.3 (b) : sur apprentissage (underfitting)

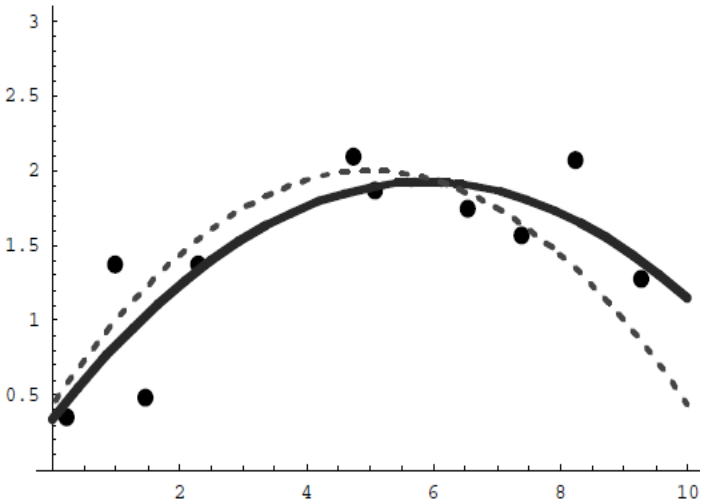


Figure 1.3 (c) : compromis entre a et b (bon modèle)

1.6.3.2 Dimension VC [4]

La dimension VC, notée h d'un ensemble de fonctions F est le nombre maximum de points pouvant être séparés de toutes les manières possibles par les fonctions de F. Cela veut dire qu'il doit exister une configuration de h (= VC(F)) points, telle que les fonctions $f \in F$ peuvent leur assigner les 2^h combinaisons de labels (classes) possibles. Cela n'est pas garanti pour tout ensemble arbitraire de h points.

Pour illustrer cette idée, considérons trois points représentés dans R^2 . Supposons que la famille de fonctions f corresponde aux droites de $R^2 : y = y_1 x + y_0$

La dimension VC de F est de 3 car on peut trouver une configuration de trois points séparables de toutes les façons possibles.

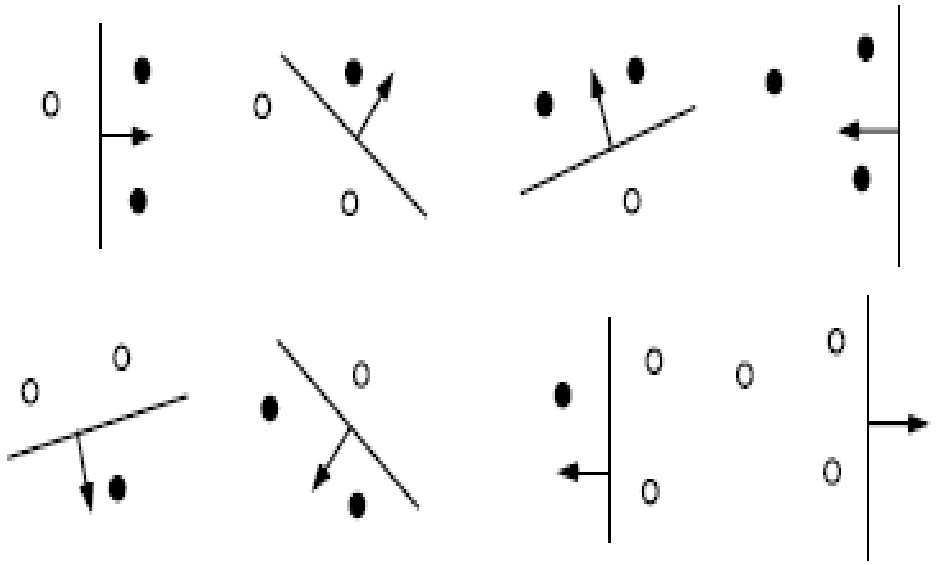


Figure 1.4 : trois points de R^2 et leurs $2^3 = 8$ configurations possibles .ces trois points sont éclatés par la famille des hyperplans orientés de R^2 [4]

Les ronds vides et pleins représentent respectivement les points assignés positivement et négativement. La flèche représente le coté de la droite où les points seront classés positivement.

Dans le chapitre 2, nous présenterons les SVM qui sont une méthode de classification basée sur la recherche d'un hyperplan séparateur. La dimension VC est un terme de capacité particulièrement bien adapté à ce genre de classificateur.

1.6.3.3 Minimisation du risque structurel

Soit F une famille de fonctions de classificateurs telle que ses éléments sont indexés par un ensemble Λ , c'est-à-dire: $F = \{f_\alpha : f_\alpha : R^d \rightarrow \{-1, +1\}, \alpha \in \Lambda\}$

D'après Vapnik [30], pour toute famille F, la majoration du risque en généralisation d'un classificateur $f_\alpha \in$ F appris sur un ensemble S de taille m vérifie :

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\frac{1}{m}\left(h(ln\frac{2m}{h}+1\right) - ln\frac{\eta}{4})} \qquad (1.6)$$

Avec la probabilité 1-η, h est la dimension VC de F.

On voit qu'il est difficile de généraliser à partir d'un échantillon de petite taille : plus m est petit, plus la différence entre le risque réel R et le risque empirique R_{emp} est susceptible d'être élevée. De même, un classificateur choisi parmi une classe très large de fonctions (h élevé) peut ``apprendre par cœur" et donner lieu à un risque plus élevé que celui mesuré [4].

L'inégalité (1.6) est particulièrement intéressante pour justifier le choix d'une machine d'apprentissage qui possède une bonne capacité de généralisation. Une bonne généralisation est obtenue quand $R(\alpha)$ est petit. Le deuxième terme de droite est appelé terme de confiance de Vapnik (la VC-confiance)[4].

1.6.3.4 Stratégies de minimisation du risque

Regardons à présent comment utiliser une borne du type (1.6) pour entraîner une machine d'apprentissage. Premièrement remarquons que dans ce type de borne, le risque empirique concerne une seule fonction alors que le terme de confiance est relatif à un ensemble de fonctions.

L'idée de base va être de fixer une série de sous ensembles de F, possédant chacun un terme de capacité propre, et ensuite d'effectuer un entraînement par MRE sur chacun d'entre eux. Concrètement, nous allons diviser l'ensemble F de la machine d'apprentissage en une structure qui consiste en l'imbrication successive de sous-ensembles dont la dimension VC va en augmentant (voir figure 1.5).

Une fois les entraînements sur chaque sous-ensemble terminés, nous allons déterminer le modèle optimal, en regardant quelle est la configuration qui minimise (1.6). On parle de sélection de modèle.

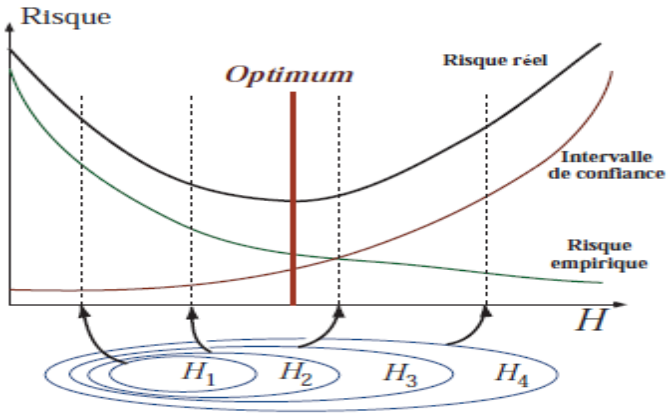


Figure 1.5 Comportement du risque empirique, l'intervalle de confiance et le risque garanti en fonction de la VC dimension [22]

1.6.4 La classification en pratique

La machine d'apprentissage, dont la tâche est d'apprendre une fonction à travers des exemples (dans notre cas c'est la fonction de décision) ne nous indique nullement comment obtenir un classificateur adapté à la tâche considérée.

Regardons à présent les différentes étapes que l'approche Machine Learning préconise pour atteindre un tel objectif.

1.6.4.1 Ensemble de données

Lorsque l'on construit un modèle afin qu'il minimise le risque empirique calculé sur les données d'apprentissage, l'erreur d'apprentissage ainsi obtenue ne peut être considérée comme une bonne mesure de l'erreur de généralisation : elle est évidemment biaisée. Pour obtenir un estimé non biaisé de l'erreur de généralisation, il est crucial de mesurer l'erreur sur des exemples qui n'ont pas servi à entraîner le modèle. Pour cela, on divise l'ensemble des données disponibles en deux parties :

- Un sous ensemble d'entraînement, dont les données serviront à l'apprentissage(ou entraînement) du modèle ;
- Un sous ensemble de test, dont les données seront utilisées uniquement pour évaluer la performance du modèle entraîné. Ce sont les données "hors-échantillon d'entraînement". On obtient ainsi l'erreur de test qui est un estimé bruité, mais non biaisé de l'erreur de généralisation.

Par ailleurs l'ensemble d'entraînement est souvent lui-même partagé entre un sous-ensemble qui sert à apprendre les paramètres d'un modèle à proprement dit, et un autre sous-ensemble dit "de validation", qui sert à la sélection de modèle.

Les différentes opérations que l'on doit effectuer avant de présenter les données à l'algorithme d'apprentissage sont :

1.6.4.1.1 Acquisition des données

Si les données proviennent d'une source analogique, il faut commencer par les transformer de manière à en avoir une représentation manipulable par un programme informatique.

1.6.4.1.2 Prétraitement

Les bons résultats qu'un classificateur automatique peut fournir reposent, en grande partie, sur la phase de prétraitement. Les données issues d'un mauvais prétraitement vont mettre en péril la qualité du classificateur.

Cette phase consiste en une succession de traitements sur les données brutes afin d'extraire de l'information et de ne garder que celle qui est utile à la classification. L'application de la phase de prétraitements sur les données brutes peut voir deux apports :

- Réduire la taille de l'information qui va être présentée au classificateur, ce qui se traduit par un gain en temps.

- Éliminer l'information non pertinente qui peut être une source de confusion pour le classificateur.

Nous allons présenter quelques méthodes utilisées lors de la phase de prétraitements.

A/ Extractions des caractéristiques

Les techniques d'extractions des caractéristiques permettent d'éliminer l'information redondante et de ne garder que l'information pertinente. On peut, par exemple, utiliser les techniques de réduction telle que la décomposition en ondelettes et l'analyse en composantes principales (ACP) pour compresser les données tout en préservant au mieux l'information utile.

B/ Sélections des caractéristiques

Il arrive souvent que même après l'application d'une technique de compression de données, le nombre de caractéristiques reste élevé pour servir d'information d'entrée d'un classificateur. En outre, la réduction ne permet pas nécessairement de sélectionner des caractéristiques permettant de bien discriminer entre éléments distincts. Certaines peuvent même être une source de confusion lors de l'affectation des éléments à des classes.

Une caractéristique est pertinente si elle contribue fortement à une bonne différenciation entre deux éléments appartenant à deux ensembles hétérogènes et un bon rapprochement entre les éléments d'un même ensemble.

Les méthodes de sélection des caractéristiques tentent de sélectionner un sous ensemble de caractéristiques adéquates parmi celles obtenues dans l'étape d'extraction. L'objectif de cette sélection est double :

- Surmonter les limitations techniques : aucun classificateur automatique ne peut fonctionner correctement avec plus d'une cinquantaine de caractéristiques.
- Accélérer le processus d'apprentissage : plus le nombre de caractéristiques est petit plus les calculs et l'apprentissage sont rapides.

1.6.4.1.3 Conversion

Il s'agit de convertir les données dans le format spécifié par l'algorithme utilisé lors de la phase de classification. La représentation vectorielle est assez populaire, notamment parce qu'elle permet de concevoir des algorithmes relativement indépendants du type d'objets à classer. Dans ce format, les données sont représentées sous forme de vecteurs dont chaque composante correspond à une caractéristique de l'objet. La plupart des algorithmes de classifications gèrent difficilement des vecteurs de grande dimension [4].

1.6.4.1.4 Post-traitement

Dans certains cas, on doit normaliser les données dans le format d'entrée. Par exemple, dans le cas d'un réseau de neurones, on conseille que la moyenne de chaque composante du vecteur sur l'ensemble des exemples d'entrainement soit proche de zéro [4].

1.6.4.2 Entraînement ou apprentissage

La phase d'entraînement consiste à sélectionner une fonction $f \in F_\alpha$ c'est-à-dire à trouver une évaluation des paramètres α_i. La sélection de ces paramètres est effectuée par un

algorithme d'apprentissage qui reçoit en entrée l'ensemble d'apprentissage ainsi qu'un ensemble de paramètres d'apprentissage : $p_1 \ldots\ldots p_k$. En ce sens, ce sont les données (de l'ensemble d'apprentissage) qui induisent l'apprentissage. L'ensemble des paramètres α_i résultant de l'apprentissage est appelé modèle.

Une machine d'apprentissage munie d'un modèle est appelée machine entraînée. La figure (1.6) schématise un tel entraînement.

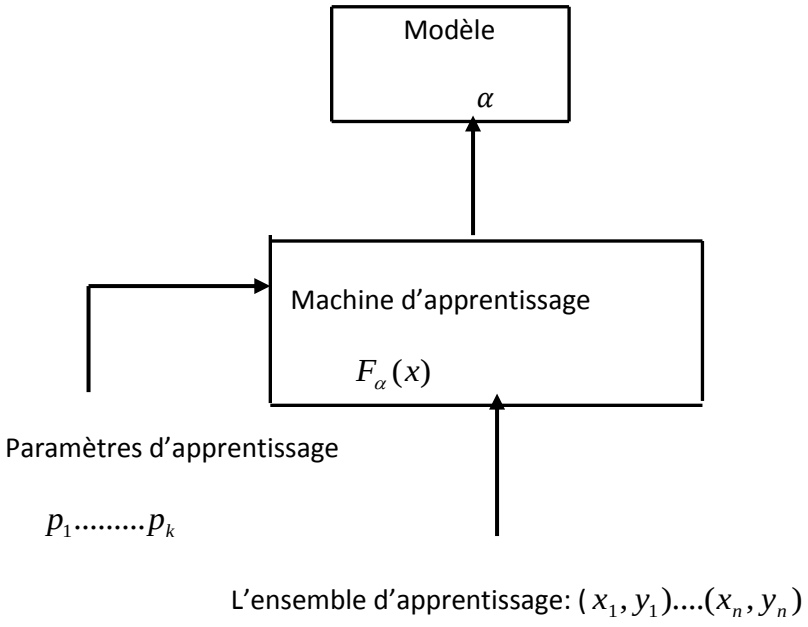


Figure 1.6 entraînement d'une machine d'apprentissage [4]

La sélection de modèle est ici comprise au sens large : il peut s'agir de choisir le meilleur entre plusieurs familles de modèles très différents, ou entre des variantes très semblables d'un même modèle, dues uniquement à des variations des valeurs d'un hyper-paramètre contrôlant la définition du modèle.

1.6.4.2 Evaluation du modèle (test)

Une fois le modèle obtenu, il est intéressant d'évaluer ses performances sur un ensemble indépendant de données : le test set. En effet, on ne peut se fier aux résultats obtenus sur l'ensemble d'apprentissage car la machine d'apprentissage a perdu son indépendance face à ces données. Cette phase permet de se rendre compte du pouvoir de généralisation du classificateur, c'est à dire sa capacité à obtenir de bons résultats sur n'importe quel ensemble de données provenant de la même distribution.

Dans les cas où l'on dispose de trop peu de données, on peut utiliser une technique de *validation croisée*, pour générer plusieurs paires de sous-ensembles entraînement / test. Cette technique est coûteuse en temps de calcul, mais permet d'obtenir un bon estimé de l'erreur de test, tout en conservant suffisamment d'exemples pour l'entraînement.

- **Validation croisée**

La validation croisée est une technique qui permet de tester un modèle d'apprentissage. La validation croisée se décline en plusieurs sous-méthodes. La plus répandue est la méthode « k-Fold » avec typiquement k ∈ [4,10]. Si l'on a une base d'apprentissage A_p contenant p éléments: $A_p = \{x_1, \dots, x_p\}$ la validation croisée consiste à appliquer les cinq étapes suivantes [26]:

1. Découper l'ensemble des exemples en k sous-ensembles disjoints de taille p/k.

2. Apprendre sur les k-1 sous-ensembles.

3. Calculer l'erreur sur la $k^{iéme}$ partie.

4. Réitèrer le processus p fois.

5. Obtenir l'erreur finale en calculant la moyenne des k erreurs précédentes.

La validation croisée est simple à mettre en œuvre et utilise toutes les données. Elle permet d'obtenir une estimation de l'erreur de généralisation. Cela permet d'éviter le sur apprentissage

1.6.4.4 Exploitation

Lorsque l'on dispose d'un modèle efficace pour une tâche considérée, on peut utiliser la machine d'apprentissage pour faire des prédictions sur de nouveaux exemples.

Un classificateur correspond donc à une machine entraînée. Les sociétés qui vendent ce type de classificateurs entraînent souvent la machine d'apprentissage sur un corpus relatif aux catégories spécifiées par le client. De cette manière, le client reçoit uniquement une machine entraînée sans avoir à se soucier de questions d'entraînement et de paramétrage.

Certaines applications grand public telle que la reconnaissance vocale (Babel Speech) ou l'OCR (Read Iris, Omnipage) proposent également des classificateurs entraînés. Il y a souvent une phase d'adaptation (à la voix ou à l'écriture) dont le but est de sélectionner le modèle correspondant le mieux à l'utilisateur.

1.7 Quelques méthodes de classification

1.7.1 K plus proches voisins

Plus connus en anglais sous le nom K-nearest neighbor (K-NN). Cette méthode diffère des traditionnelles méthodes d'apprentissage car aucun modèle n'est induit à partir des exemples. Les données restent telles quelles : elles sont simplement stockées en mémoire[79].

Pour prédire la classe d'un nouveau cas, l'algorithme cherche les K plus proches voisins de ce nouveau cas et prédit (s'il faut choisir) la réponse la plus fréquente de ces K plus proches voisins[79].

La méthode utilise donc deux paramètres : le nombre K et la fonction de similarité pour comparer le nouveau cas aux cas déjà classés.

$d(x_i, y_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$

Ces valeurs sont arbitraires mais importantes car des résultats très différents résultent de leurs choix. Notez aussi que, si le temps d'apprentissage est inexistant puisque les données sont stockées telles quelles, la classification d'un nouveau cas est par contre coûteuse puisqu'il faut comparer ce cas à tous les exemples déjà classés.

Voici un exemple. On doit classer le nouveau cas x_q. Si on choisit K = 1, x_q. sera classé +. Si K = 5, le même x_q. sera classé -. On voit donc que le choix de K est très important dans le résultat final.

Sur la figure 1.7 de droite, les exemples sont représentés par des points. Chaque surface autour d'un exemple montre les positions possibles de nouveaux cas à classer où le résultat de la classification sera la classe de l'exemple si K=1. Cette figure est aussi connue sous le nom de **diagramme de Voronoi**.

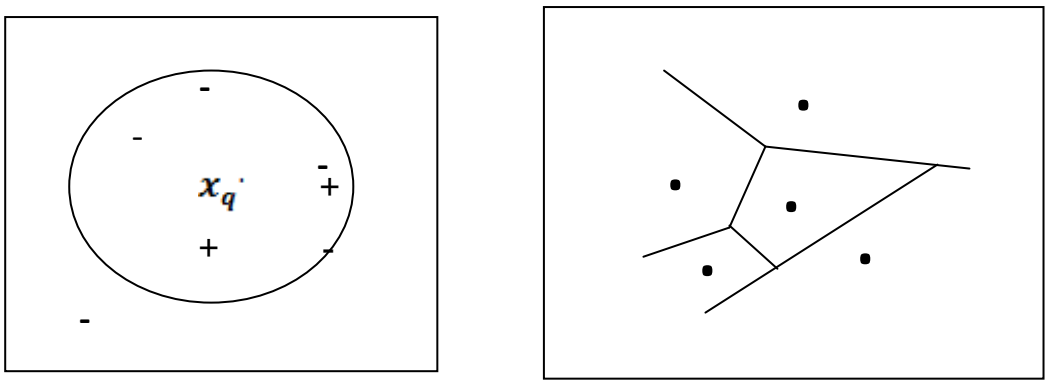


Figure 1.7 **diagramme de Voronoi**.[79]

Dans KNN de base, on choisit la classe majoritairement représentée par les K plus proches voisins. Une autre solution est de pondérer la contribution de chaque K plus proche voisin en fonction de sa distance avec le nouveau cas à classer. Notez qu'avec cette méthode de pondération, on pourrait utiliser les N exemples au lieu des K plus proches voisins : en effet, plus un exemple sera éloigné du nouveau cas à classer et moins sa classe contribuera au résultat final. Le seul désavantage est la perte de temps.

Les expériences menées avec les KNN montrent qu'ils résistent bien aux données bruitées. Par contre, ils requièrent de nombreux exemples. Un problème rencontré avec le KNN de base est qu'il utilise tous les attributs d'un cas pour calculer la similarité avec un nouveau cas à classer. en fonction de leur distance avec le nouveau cas à classer.

1.7.2 Arbres de décision

Les arbres de décision sont les plus populaires des méthodes d'apprentissage. Les algorithmes connus sont ID3 (Quinlan 1986) et C4.5 (Quinlan 1993). Comme toute méthode d'apprentissage supervisé, les arbres de décision utilisent des exemples[79].

Si l'on doit classer des exemples dans des catégories, il faut construire un arbre de décision par catégorie. Pour déterminer à quelle catégorie appartient un nouvel exemple, on utilise l'arbre de décision de chaque catégorie auquel on soumet le nouvel exemple à classer.

Chaque arbre répond Oui ou Non (il prend une décision).Concrètement, chaque nœud d'un arbre de décision contient un test (un IF...THEN) et les feuilles ont les valeurs Oui ou Non. Chaque test regarde la valeur d'un attribut de chaque exemple. En effet, on suppose qu'un exemple est un ensemble d'attributs/valeurs.

Pour construire l'arbre de décision, il faut trouver quel attribut tester à chaque nœud. C'est un processus récursif. Pour déterminer quel attribut tester à chaque étape, on utilise un calcul statistique qui détermine dans quelle mesure cet attribut sépare bien les exemples Oui/Non. On crée alors un nœud contenant ce test, et on crée autant de descendants que de valeurs possibles pour ce test[79].

L'arbre de décision classe trop bien les exemples, mais est mauvais pour généraliser, c'est-à-dire qu'il prédit mal la classification (Oui /Non) de nouvelles instances.

L'apprentissage par cœur peut survenir lorsque les exemples sont bruités ou qu'il y a peu d'exemples. Dans ce dernier cas par exemple, il se peut qu'un arbre de décision soit construit avec des tests sur des attributs qui séparent bien les exemples, mais que ces mêmes tests ne soient pas bons pour classer de nouvelles instances.

De façon générale, plus l'arbre de décision est profond, et plus le risque d'apprentissage par cœur augmente. Le problème est donc de trouver la profondeur idéale.

Pour estimer cette profondeur idéale, on divise l'ensemble des exemples en 2/3 pour les construire l'arbre de décision, et 1/3 pour valider l'arbre.

La procédure de validation est la suivante : après avoir construit un arbre de décision complet, qui risque donc l'apprentissage par cœur, on enlève successivement des nœuds de cet arbre. Ce nœud devient une feuille dont la valeur est la valeur prépondérante dans les feuilles situées jusqu'alors sous ce nœud.

On valide alors ce nouvel arbre amputé d'un nœud grâce aux exemples de validation (1/3) : si le nouvel arbre ne classe pas plus mal les exemples de validation que l'arbre précédent l'amputation, alors on le garde. On continue à amputer des nœuds de l'arbre de décision tant que le nouvel arbre continue à classer de mieux en mieux les exemples de validation.

Mais cette méthode a un inconvénient lorsque les exemples sont limités : on n'utilise en effet que les 2/3 des exemples de départ pour construire l'arbre, puisqu'on réserve 1/3 pour le valider ensuite. De nombreuses autres méthodes ont été proposées, notamment celle utilisée dans C4.5.

On écrit l'arbre de décision sous forme de règles IF. . . THEN et on élimine progressivement des conditions dans les IF. On n'élimine une condition que si elle ne décroît pas la capacité de l'arbre à classer les exemples.

1.7.3 Naïve ou décisions de Bayes

Nommés d'après le théorème de Bayes, ces méthodes sont qualifiées de "Naïve" ou "Simple" car elles supposent l'indépendance des variables. L'idée est d'utiliser des conditions de probabilité observées dans les données. On calcule la probabilité de chaque classe parmi les exemples.

Une variante des Naïve Bayes sont les réseaux Bayésiens : dans ce modèle, on ne suppose plus que les variables sont toutes indépendantes, et on autorise certaines à être liées.

Cela alourdit considérablement les calculs et les résultats n'augmentent pas de façon significative.

1.7.4 Réseaux de neurones

Un réseau neuronal est l'association, en un graphe plus ou moins complexe, d'objets élémentaires, les neurones formels. Les principaux réseaux se distinguent par l'organisation du graphe (en couches, complets.. .), c'est-à dire leur architecture, son niveau de complexité (le nombre de neurones) et par le type des neurones (leurs fonctions de transition)[73].

- Neurone formel

le neurone formel est un modèle qui se caractérise par un état interne s ∈ S, des signaux d'entrée $x_1, x_2, \dots \dots x_p$ et une fonction de transition d'état [73]

$$s = h(x_1, \dots\dots, x_p) = f(\beta_0 + \sum_{j=1}^{p} \beta_j x_j)$$

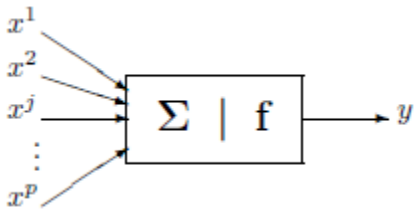


Figure 1.8 : représentation d'un neurone formel

La fonction de transition opère une transformation d'une combinaison affine des signaux d'entrée, β_0 étant appelé le biais du neurone. Cette combinaison affine est déterminée par un vecteur de poids $[\beta_0, \beta_1, \dots\dots \beta_p]$ associée à chaque neurone et dont les valeurs sont estimées dans la phase d'apprentissage. Ils constituent "la mémoire" ou "connaissance répartie" du réseau.

Les différents types de neurones se distinguent par la nature f de leur fonction de transition. Les principaux types sont :

• linéaire f est la fonction identité,
• sigmoïde de f(x) = 1/(1 + ex),
• seuil f(x) = 1[0,+1[(x),
• . . .

Les modèles linéaires et sigmoïdaux sont bien adaptés aux algorithmes d'apprentissage comme celui de rétro propagation du gradient car leur fonction de transition est différentiable. Ce sont les plus utilisés.

- Perceptron multicouche
    - architecture

Le perceptron multicouche (PMC) est un réseau composé de couches successives. Une couche est un ensemble de neurones n'ayant pas de connexion entre eux. Une couche

d'entrée lit les signaux entrant, un neurone par entrée xj , une couche en sortie fournit la réponse du système. Selon les auteurs, la couche d'entrée qui n'introduit aucune modification n'est pas comptabilisée. Une ou plusieurs couches cachées participent au transfert. Un neurone d'une couche cachée est connecté en entrée à chacun des neurones de la couche précédente et en sortie à chaque neurone de la couche suivante [73].

Un perceptron multicouche réalise donc une transformation

$$y = \phi(x_1, \ldots, x_p; \beta)$$

où β est le vecteur contenant chacun des paramètres β_{jkl} de la $j^{ième}$ entrée du $k^{ième}$ neurone de la $l^{ième}$ couche ; la couche d'entrée (l = 0) n'est pas paramétrée, elle ne fait que distribuer les entrées sur tous les neurones de la couche suivante.

Les entrées d'un réseau sont encore notées $x^1, x^2, \ldots, x^p$ comme les variables explicatives d'un modèle tandis que les poids des entrées sont des paramètres à estimer lors de la procédure d'apprentissage et que la sortie est la variable `a expliquer ou cible du modèle.

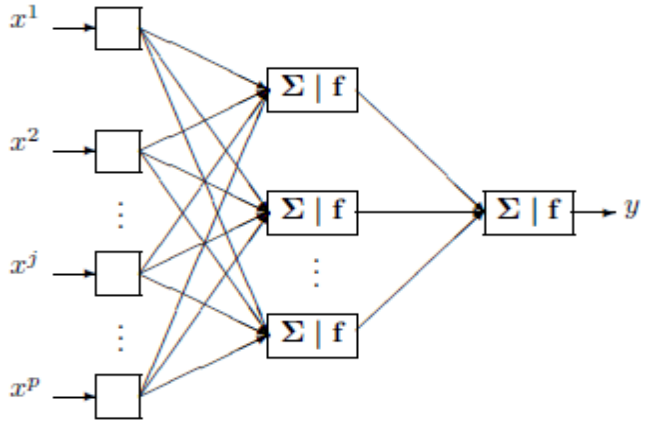


Figure 1.9 : Exemple de perceptron multicouche avec une couche cachée et une couche de sortie [73]

- L'apprentissage

Supposons que l'on dispose d'une base d'apprentissage de taille n d'observations $(x_i^1, x_i^2, \ldots, x_i^p; y_i)$ des variables explicatives $X^1, \ldots, X^p$ et de la variable à prévoir Y . L'apprentissage est l'estimation β des paramètres du modèle solutions du problème des moindres carrés :

$$\hat{\beta} = \arg\min_b Q(b) \quad avec \quad Q(b) = \frac{1}{n}\sum_{i=1}^{n}[y_i - \phi(x_i^1, \ldots, x_i^p; (b)]^2$$

L'algorithme d'optimisation le plus utilisé est celui de rétro propagation du gradient basé sur l'idée suivante : en tout point b, le vecteur gradient de Q pointe dans la direction de l'erreur croissante. Pour faire décroître il suffit donc de se déplacer en sens contraire. Il s'agit d'un algorithme itératif modifiant les poids de chaque neurone selon :

$$b_{jkl}(i) = b_{jkl}(i-1) + \Delta b_{jkl}(i)$$

Ou la correction $\Delta b_{jkl}(i)$ est proportionnelle au gradient et à l'erreur attribuée à l'entrée $\varepsilon_{jkl}(i)$ concernée et incorpore un terme d'inertie $\alpha b_{jkl}(i-1)$ permettant d'amortir les oscillations du système :

$$\Delta b_{jkl}(i) = -\tau \varepsilon_{jkl}(i) \frac{\partial Q}{\partial b_{jkl}} + \alpha b_{jkl}(i-1)$$

Le coefficient de proportionnalité τ est appelé le taux d'apprentissage. Il peut être fixé ou déterminé par l'utilisateur ou encore varier en cours d'exécution selon certaines règles paramétrées par l'utilisateur [73]. Il paraît en effet intuitivement raisonnable que, grand au début pour aller plus vite, ce taux décroisse pour aboutir à un réglage plus fin au fur et à mesure que le système s'approche d'une solution. La formule de rétro propagation de l'erreur fournit, à partir des erreurs observées sur les sorties, l'expression de l'erreur attribuée à chaque entrée de la couche de sortie à la couche d'entrée.

Une amélioration importante consiste à introduire un terme de pénalisation ou régularisation dans le critère à optimiser. Celui-ci devient alors :

$$\hat{\beta} = \arg\min_{b} Q(b) + \delta \|b\|^2$$

Le paramètre δ doit être fixé par l'utilisateur ; plus il est important et moins les paramètres ou poids peuvent prendre des valeurs "cahotiques" contribuant ainsi à limiter les risques de sur apprentissage.

Algorithme Rétro propagation du gradient

Initialisation

Les poids b_{jkl} par tirage aléatoire selon une loi uniforme sur [0, 1].

Normaliser dans [0, 1] les données d'apprentissage.

Tant que (Q > errmax) ou (niter < itermax) **Faire**

Ranger la base d'apprentissage dans un nouvel ordre aléatoire.

Pour chaque élément i = 1, . . . , n de la base **Faire**

Calculer $\varepsilon(i) = y_i - \phi(x_i^1, \ldots\ldots, x_i^p; (b)(i-1))$ en propageant les entrées vers l'avant.

L'erreur est "rétropropagée" dans les différentes couches afin d'affecter à chaque entrée une responsabilité dans l'erreur globale.

Mise à jour de chaque poids $b_{jkl}(i) = b_{jkl}(i-1) + \Delta b_{jkl}(i)$

Fin Pour

Fin Tant que

1.7.5 Programmation génétique

C'est une méthode générale qui peut être utilisée après n'importe qu'elle méthode précédente, par exemple avec les arbres de décisions. En entrée, un algorithme génétique

reçoit une population de classificateurs non optimaux. Le but du programme génétique est de produire un classificateur plus optimal que chacun de ceux de la population d'origine.

D'une façon simple, cela consiste à extraire les meilleures parties de chaque classificateur d'origine et de les mettre ensemble pour produire un nouveau classificateur. Cela suppose de pouvoir comparer l'efficacité d'un classificateur.

Un résultat important de la méthode est qu'après chaque itération on obtient un classificateur meilleur qu'avant. On peut donc arrêter les itérations à tout moment, même si le résultat n'est pas l'optimum.

1.7.6 Machines à Vecteurs Support (ou SVM)

Cette technique initiée par Vapnik tente de séparer linéairement les exemples positifs des exemples négatifs dans l'ensemble des exemples. Chaque exemple doit être représenté par un vecteur de dimension n.

La méthode cherche alors l'hyperplan qui sépare les exemples positifs des exemples négatifs, en garantissant que la marge entre le plus proche des positifs et des négatifs soit maximale.

Intuitivement, cela garantit un bon niveau de généralisation car de nouveaux exemples pourront ne pas être trop similaires à ceux utilisés pour trouver l'hyperplan mais être tout de même situés franchement d'un côté ou l'autre de la frontière.

L'intérêt des SVM est la sélection de Vecteurs Supports qui représentent les vecteurs discriminant grâce auxquels est déterminé l'hyperplan. Les exemples utilisés lors de la recherche de l'hyperplan ne sont alors plus utiles et seuls ces vecteurs supports sont utilisés pour classer un nouveau cas. Cela en fait une méthode très rapide, cette méthode sera détaillée dans le chapitre suivant.

1.8 Conclusion

Dans le présent chapitre, nous avons présenté les catégories de la classification automatique et les différents domaines où elle peut intervenir. Nous avons pu voir l'intérêt apporté par cette dernière pour accélérer le processus d'identification.

Cependant, elle peut être une source d'erreurs si le classificateur que l'on utilise est de mauvaise qualité. Le choix d'un classificateur efficace est primordial pour garantir une classification avec un minimum d'erreurs.

Dans le chapitre suivant, une technique de classification automatique supervisée est présentée. Cette technique, basée sur la minimisation du risque structurel, a prouvée ses performances dans plusieurs domaines.

Chapitre 2

Les Machines à Vecteurs Supports

2.1 Introduction

Les machines à vecteurs supports (ou séparateurs à vaste marge) sont un ensemble de techniques d'apprentissage destinées à résoudre des problèmes de discrimination , c'est-à-dire décider à quelle classe appartient un échantillon, ou de régression, c'est-à-dire prédire la valeur numérique d'une variable.

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.

Les SVM fournissent une approche très intéressante de l'approximation statistique. Souvent, le nombre des exemples pour l'apprentissage est insuffisant pour que les estimateurs fournissent un modèle avec une bonne précision. D'un autre côté, l'acquisition d'un grand nombre d'exemples s'avère être souvent très coûteuse et peut même mener à des problèmes de sur-apprentissage dans le cas ou la capacité du modèle est très complexe. Pour ces deux raisons, il faut arriver à un compromis entre la taille des échantillons et la précision recherchée.

Ces nouvelles techniques unifient deux théories :

- Minimisation du risque empirique
- Capacité d'apprentissage d'une famille de fonctions.

La plupart des techniques de l'apprentissage machine possèdent un 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 [4].

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.

L'idée principale des SVM consiste à projeter les données dans un espace de plus grande dimension appelé, espace de caractéristiques, afin que les données non linéairement séparables dans l'espace d'entrée deviennent linéairement séparables dans l'espace de caractéristiques. En appliquant dans cet espace la technique de construction d'un hyperplan optimal séparant les deux classes, on obtient une fonction de classification qui dépend d'un produit scalaire des images des données de l'espace d'entrée dans l'espace des caractéristiques.

Ce produit scalaire peut être exprimé, sous certaines conditions, par des fonctions définies dans l'espace d'entrée, qu'on appelle les noyaux.

Ce multiple choix de noyaux rend les SVM plus intéressantes et surtout plus riches puisqu'on peut toujours chercher de nouveaux noyaux qui peuvent être mieux adaptés à la tâche qu'on veut accomplir. Les trois noyaux les plus utilisés sont : le noyau linéaire, le noyau polynomial et le noyau gaussien noté aussi RBF (Radial Basis Function).

2.2 Historique

Les séparateurs à vastes marges reposent sur deux idées clés :

- la notion de marge maximale et
- la notion de fonction noyau.

Ces deux notions existaient depuis plusieurs années avant qu'elles ne soient mises en commun pour construire les SVM.

- ✓ L'idée des hyperplans à marge maximale a été explorée dès 1963 par Vladimir Vapnik et A. Lerner, et en 1973 par Richard Duda et Peter Hart dans leur livre Pattern Classification. Les fondations théoriques des SVM ont été explorées par Vapnik et ses collègues dans les années 70 avec le développement de la Théorie de Vapnik-Chervonenkis..

- ✓ L'idée des fonctions noyaux n'est pas non plus nouvelle: le théorème de Mercer date de 1909, et l'utilité des fonctions noyaux dans le contexte de l'apprentissage artificiel a été montrée dès 1964 par Aizermann, Bravermann et Rozoener.

Ce n'est toutefois qu'en 1992 que ces idées seront bien comprises et rassemblées par Boser, Guyon et Vapnik dans un article, qui est l'article fondateur des séparateurs à vaste marge.

L'idée des variables ressorts, qui permet de résoudre certaines limitations pratiques importantes, ne sera introduite qu'en 1995. À partir de cette date, qui correspond à la publication du livre de Vapnik, les SVM gagnent en popularité et sont utilisées dans de nombreuses applications.

2.3 Principe de fonctionnement général

Pour deux classes d'exemples donnés, le but de SVM est de trouver un classificateur qui va séparer les données et maximiser la distance entre ces deux classes. Avec SVM, ce classificateur est un classificateur linéaire appelé hyperplan.

2.3.1 Notions de base

2.3.1.1 Hyperplan

Plaçons-nous dans le cas d'une classification binaire (i.e. les exemples à classifier réparties en 2 classes). On appelle *hyperplan séparateur* un hyperplan qui sépare les deux classes **figure 2.1**, en particulier il sépare leurs points d'apprentissage. Comme il n'est en général pas possible d'en trouver un, on se contentera donc de chercher un hyperplan discriminant qui est une approximation au sens d'un critère à fixer (maximiser la distance entre ces deux classes) [56].

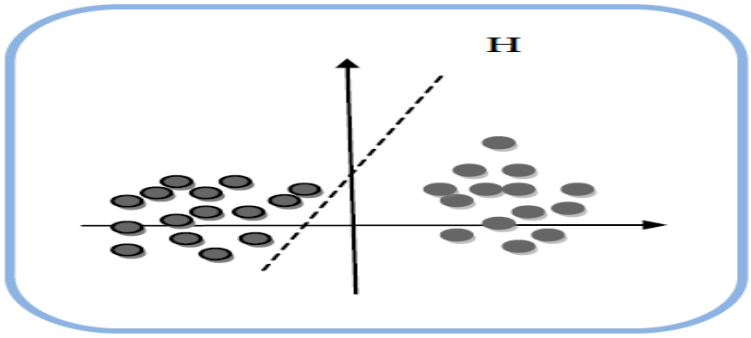


Figure 2.1 L'hyperplan H qui sépare les deux ensembles de points [1]

2.3.1.2. Vecteurs supports

Pour une tâche de détermination de l'hyperplan séparable des SVM est d'utiliser seulement les points les plus proches (i.e. les points de la frontière entre les deux classes des données) parmi l'ensemble total d'apprentissage, ces points sont appelés *vecteurs supports* **figure 2.2** [56] .

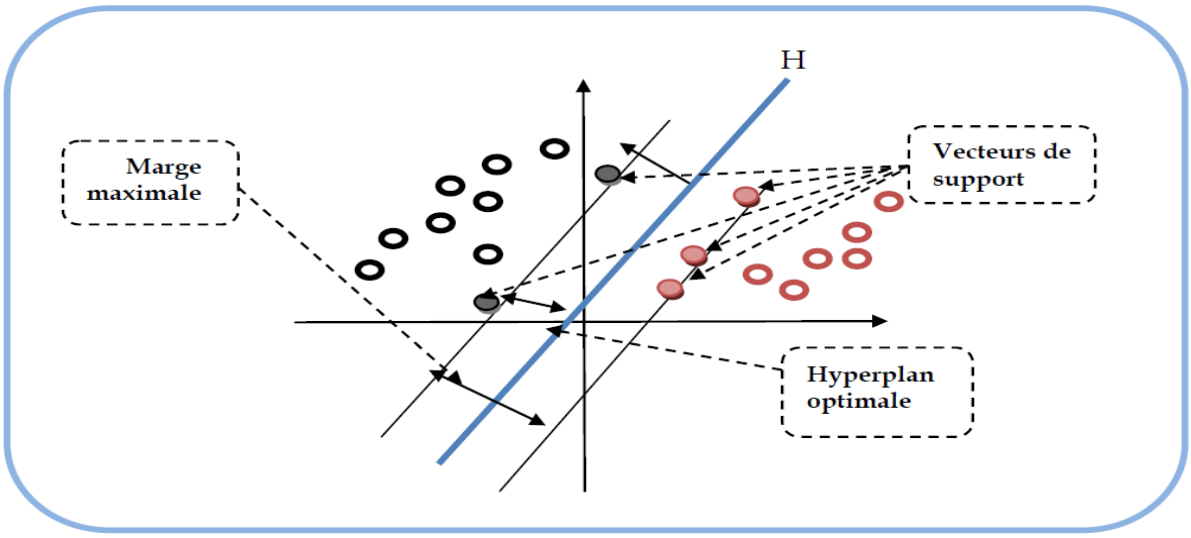


Figure 2.2 L'hyperplan H optimal, vecteurs supports et marge maximale [1]

2.3.1.2 Marge

Il existe une infinité d'hyperplans capable de séparer parfaitement les deux classes d'exemples. Le principe des SVM est de choisir celui qui va maximiser la distance minimale entre l'hyperplan et les exemples d'apprentissage (i.e. la distance entre l'hyperplan et les vecteurs supports), cette distance est appelée la marge (**figure 2.2)**.

2.3.2 Propriétés fondamentales :

- ✓ **Pourquoi maximiser la marge ? :**

Intuitivement, le fait d'avoir une marge plus large procure plus de sécurité lorsqu'on classe un nouvel exemple. De plus, si l'on trouve le classificateur qui se comporte le mieux vis-à-vis des

données d'apprentissage, il est clair qu'il sera aussi celui qui permettra au mieux de classer les nouveaux exemples.

Dans le schéma **figure 2.3**, la partie droite nous montre qu'avec un hyperplan optimal, un nouvel exemple reste bien classé alors qu'il tombe dans la marge. On constate sur la partie gauche qu'avec une plus petite marge, l'exemple se voit mal classé.

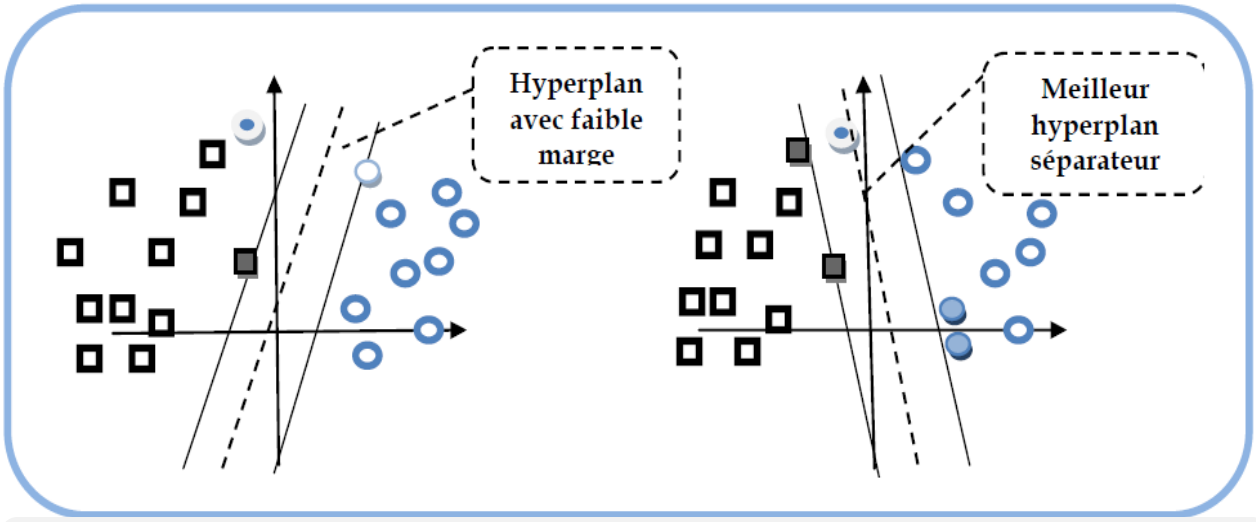


Figure 2.3 meilleur hyperplan séparateur [1]

✓ **Linéarité et non-linéarité :**

Parmi les modèles des SVM, on constate

- les cas linéairement séparables et
- les cas non linéairement séparables.

Les premiers sont les plus simples des SVM car ils permettent de trouver facilement le classificateur linéaire. Dans la plupart des problèmes réels il n'y a pas de séparation linéaire possible entre les données, le classificateur de marge maximale ne peut pas être utilisé car il fonctionne seulement si les classes de données d'apprentissage sont linéairement séparables [56].

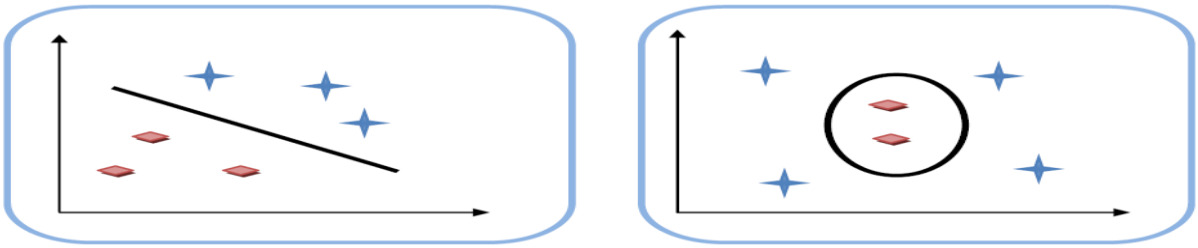


Figure 2.4 : à gauche cas linéairement séparable, à droite non linéairement séparable [1]

- **Cas non linéaire :**

Pour surmonter les inconvénients des cas non linéairement séparable, l'idée des SVM est de changer l'espace des données. La transformation non linéaire des données peut permettre une séparation linéaire des exemples dans un nouvel espace.

On va donc avoir un changement de dimension. Ce nouvel espace est appelé « espace de redescription ». En effet, intuitivement, plus la dimension de l'espace de redescription est

grande, plus la probabilité de pouvoir trouver un hyperplan séparateur entre les exemples est élevée.

On a donc une transformation d'un problème de séparation non linéaire dans l'espace de représentation en un problème de séparation linéaire dans un espace de redescription de plus grande dimension. Cette transformation non linéaire est réalisée *via* une fonction noyau.

En pratique, quelques familles de fonctions noyau paramétrables sont connues et il revient à l'utilisateur de SVM d'effectuer des tests pour déterminer celle qui convient le mieux pour son application. On peut citer les exemples de noyaux suivants : polynomial, gaussien, sigmoïde et laplacien (voir figure 2.5).

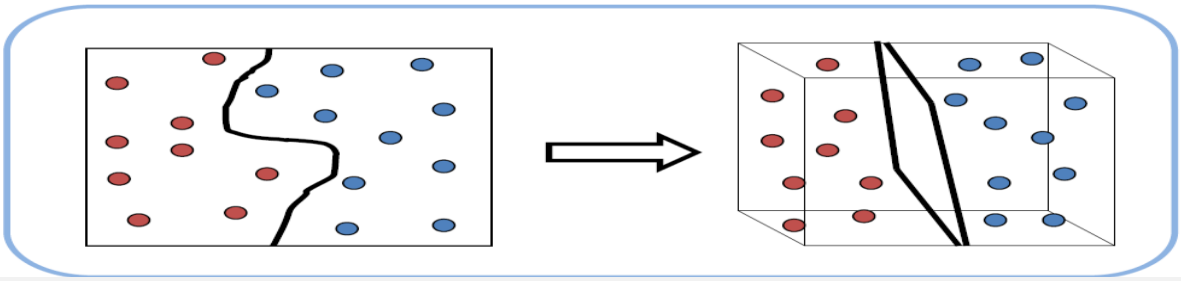


Figure 2.5 : Transformation des données dans un espace de grande dimension [1]

2.3.3 Fondements mathématiques [4]

Nous allons détailler dans les paragraphes ci-dessous les principaux fondements mathématiques sur lesquels reposent les SVM.

2.3.3.1 SVM linéaire

2.3.3.1.1 Cas linéairement séparable

- **Définition**

Un classificateur est dit linéaire lorsqu'il est possible d'exprimer sa fonction de décision par une fonction linéaire en x. Dans la suite, nous supposons que les exemples nous sont fournis dans le format vectoriel. Notre espace d'entrée X correspond donc à $\Re^n$ ou n est le nombre de composantes des vecteurs contenant les données.

$$f(x) = < w.x > + b = \sum_{j=1}^{n} w_j x_j + b$$

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

Si les données sont linéairement séparables, alors il existe un hyperplan d'équation

$$< w.x > + b = 0 \; et \; tel \; que$$

$$< w.x > + b \geq 1 \quad si \quad y_i = +1$$

$$< w.x > + b \leq 1 \; si \; y_i = -1$$

On peut combiner ces deux inéquations en une seule : $y_i(w.x + b) \geq 1$

- **Marge de l'hyperplan**

Avant de définir la marge, il est utile d'introduire la notion de l'ensemble d'apprentissage linéairement séparable.

Un ensemble d'apprentissage $((x_1, y_1), (x_2, y_2), \ldots . (x_n, y_n))$ est linéairement séparable si et seulement si :

$$\exists\, w \in R^n, b \in R : y_i (< w.x_i > + b) \geq 0 \quad \forall\, i = 1..n$$

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.

La notion de marge peut être relative à un exemple particulier où à l'ensemble d'apprentissage De plus, on considère deux types de marges : fonctionnelle et géométrique.

La marge fonctionnelle d'un exemple xi, par rapport à l'hyperplan caractérisé par w et b est la quantité :

$$y_i (< w.x_i > + b) \geq 1$$

La marge géométrique quant à elle représente la distance euclidienne prise perpendiculairement entre l'hyperplan et l'exemple xi. En prenant un point quelconque x_p se trouvant sur l'hyperplan, la marge géométrique peut s'exprimer par

$$\frac{w}{\|w\|}(x_i - x_p)$$

La figure 2.6 illustre la situation. En utilisant la marge fonctionnelle et le fait que x_p est sur l'hyperplan, on peut calculer la valeur de la marge géométrique :

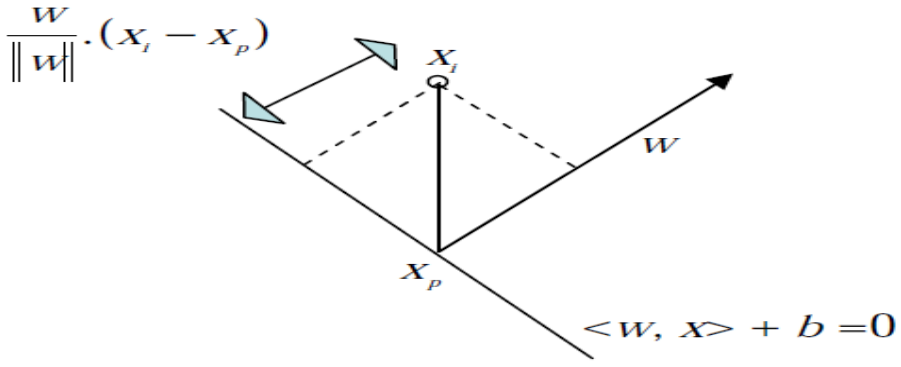


Figure 2.6 Expression de la marge pour l'exemple xi [4]

$$< w, x_i > + b \quad = \quad f(x_i)$$

$$\frac{-<w,x_p>+b \quad = \ 0}{<w(x_i-x_p)>= \ f(x_i)}$$

$$\Rightarrow \ \frac{w}{\|w\|}(x_i-x_p)=\frac{f(x_i)}{\|w\|}$$

La marge géométrique d'un exemple xi, par rapport à l'hyperplan caractérisé par w et b est la quantité :$\Psi_{w,b}\,(x_i,y_i)=y_i(\frac{w}{\|w\|}.x_i+\frac{b}{\|w\|})$

La marge de l'ensemble d'apprentissage par rapport à l'hyperplan caractérisé par w et b est définie comme étant

$$\Psi_{w,b} \ = \ min_{i=1..n}\,\Psi_{w,b}(x_i,y_i)$$

- **Hyperplans canoniques**

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 l'ensemble d'apprentissage est linéairement séparable. Dés 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 les plus proches. La figure 2.7 illustre cette situation. il est possible que différentes équations correspondent au même plan géométrique : $a\langle w,x\rangle+\frac{b}{a}=0 \ \ \forall \ a \ \in R$

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

Les deux hyperplans sont appelés hyperplans canoniques. Notons que la marge des hyperplans canoniques est $\frac{1}{\|w\|}$

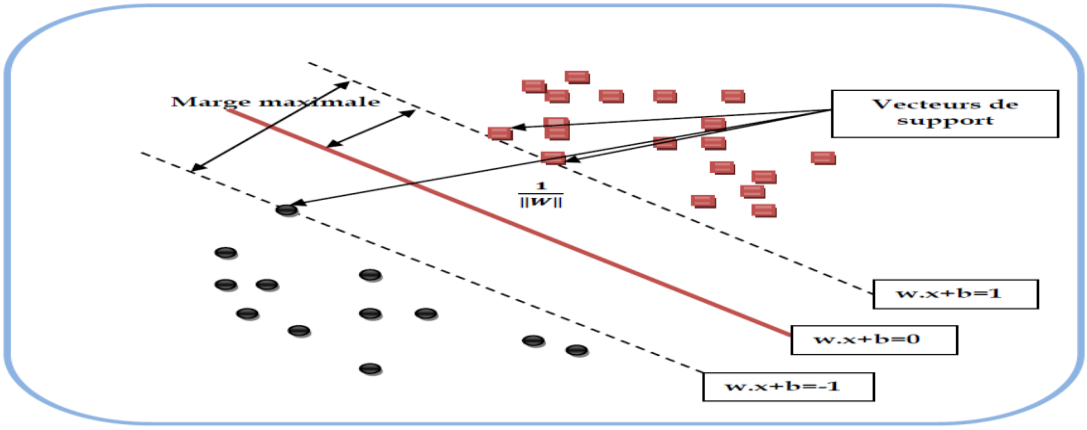


Figure 2.7 Exemple graphique des données linéairement séparables [1]

- **VC dimension**

La VC-dimension dVC des hyperplans canoniques de $\Re^n$ est égale à n+1 [2]. Autrement dit, tout ensemble de n+1 points est séparable par un hyperplan dans $\Re^n$. On remarque que dVC correspond au nombre total de paramètres du classificateur ($w \in \Re^n$; $b \in R$).

Pour appliquer le principe de la Minimisation du Risque Structurel (MRS), on doit construire des ensembles Fi d'hyperplans qui dépendent de la VC-dimension et minimisent simultanément le risque empirique et la VC-dimension [2]. L'ensemble des hyperplans canoniques est caractérisé par une contrainte sur la norme du vecteur w. En effet, Vapnik (1995) montre que l'ensemble des fonctions : $\{f_n(x) = sign(w^t x + w_0)\ ,\ \|w\| \leq A\}$

a une VC-dimension qui satisfait la borne suivante : $d_{vc} \leq \min\{A^2, p\} + 1$ (2.1)

Si r est le rayon de la plus petite hyper sphère contenant les points $\{x_i \in \Re^n\}$

L'inégalité (2.1) devient $d_{vc} \leq \min\{r^2 A^2, p\} + 1$

Ainsi en minimisant la borne A , c'est-à-dire en maximisant la marge, la vc-dimension du classificateur sera minimale. on parle alors d'un classificateur à marge maximale.

- **Classificateur à marge maximale**

Maintenant que nous avons défini les notions de marges et d'hyperplans canoniques, nous pouvons formuler un problème d'optimisation mathématique tel que sa solution nous fournisse l'hyperplan optimal (maximisant la marge) :

QP1	Minimiser	$W(w,b) = \frac{1}{2}\|w\|^2$
	Tel que	$y_i(\langle w, x_i\rangle + b) \geq 1$

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_i et b. 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 mêmes problèmes que les méthodes classiques.

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.

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

Les variables duales α_i intervenant dans le Lagrangien sont appelées multiplicateurs de Lagrange. Elles représentent 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 pourra donc pas être retenue comme solution optimale. Notons que L doit être minimisé par rapport aux variables primales et maximisé par rapport aux variables duales.

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

- **Théorème (KKT)** [4]

Pour les problèmes différentiables convexes : Considérons un problème d'optimisation de la forme :

$$\text{Minimiser} \quad g(x)$$

$$\text{Tel que} \quad \begin{cases} c_i(x) \leq 0 \ \forall \, i \in 1..n \\ e_j(x) = 0 \ \forall \, j \in 1..n \end{cases}$$

avec $g(.)$, $c_i(.), e_j(.)$ 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 $\bar{\alpha} \in \mathbb{R}^n$ avec $\alpha_i \geq 0 \ \forall \, i = 1..n$ et $\bar{\beta} \in \mathbb{R}^{n'}$

Avec β_j s.r.s $\forall \, j = 1..n'$ tels que :

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

$$\begin{aligned} \partial_{\alpha_i} L(\bar{x}, \bar{\alpha}) &= c_i(\bar{x}) \leq 0 \\ \partial_{\beta_j} L(\bar{x}, \bar{\alpha}) &= e_j(\bar{x}) = 0 \\ \bar{\alpha}_i \, c_i(\bar{x}) &= 0 \ \forall \ i = 1..n \\ \bar{\beta}_j \, e_j(\bar{x}) &= 0 \ \forall \ j = 1..n' \end{aligned}$$

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:

1. $\alpha_i = 0$: 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. $\alpha_i \neq 0$: 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.

$$\partial_w L(w,b,\alpha) = w - \sum \alpha_i y_i x_i = 0 \quad (1)$$

$$\partial_b L(w,b,\alpha) = \sum \alpha_i y_i = 0 \quad (2)$$

$$\partial_{\alpha_i} L(w,b,\alpha) = -y_i(\langle w, x_i \rangle + b) + 1 \leq 0 \quad (3)$$

$$\alpha_i(y_i(\langle w, x_i \rangle + b) - 1) = 0 \ \forall i = 1..n \quad (4)$$

L'équation (1), permet de ré exprimer w : $w = \sum_{i=1}^{n} \alpha_i y_i x_i$ (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 (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 (supposé très élevée). Pour ce faire, nous substituons (2) et (5) dans le Lagrangien :

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

$$= \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle - \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle + \sum_{i=0}^{n} \alpha_i$$

$$= \sum_{i=0}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

A partir de quoi nous pouvons formuler le problème dual :

$$\textbf{QP2} \ \ Maximiser \ W(\alpha) = \sum_{i=0}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

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

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 :

$$b = -\frac{max_{y_i=-1}(< w, x_i >) + max_{y_i=+1}(< w, x_i >)}{2}$$

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 son nuls. En effet seuls les α_i correspondants aux exemples se trouvant sur les hyperplans canoniques (sur la contrainte) sont non nuls.

Ces exemples sont appelés vecteurs support (SV). On peut les voir comme les représentants de leurs catégories car si l'ensemble d'apprentissage n'était constitué que des SV, l'hyperplan optimal que l'on trouverait serait identique. Les vecteurs qui ne sont pas de support (α_i =0) n'ont aucune influence dans la solution.

La fonction de décision pour la classification de vecteurs inconnus x est donnée par

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

Où m est le nombre de vecteurs supports

2.3.3.1.2 Cas linéairement non séparable

Le classificateur de marge maximale ne peut pas être utilisé dans la plupart des problèmes réels : si les données ont été affectées par le bruit, il n'y a pas de séparation linéaire entre elles. Dans ce cas, le problème d'optimisation ne peut pas être résolu.

Pour surmonter cette nouvelle contrainte, nous allons introduire une notion de « tolérance » faisant appel à une technique dite des **variables ressort** (*slack variables*)

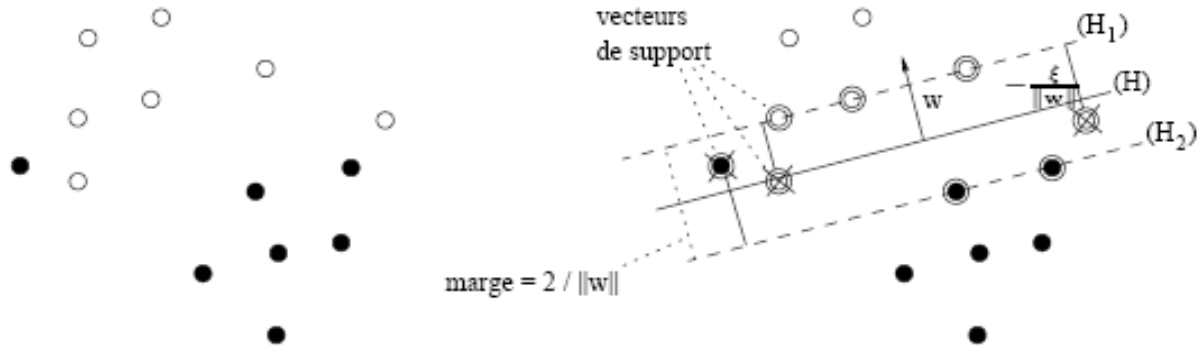


Figure 2.8 à gauche un ensemble de données non linéairement séparables, à droite la solution SVM linéaire à marge douce . Les vecteurs de support sont encerclés et les points d'apprentissage mal classés sont marqués d'une croix [53].

Le problème d'optimisation initial était : Minimiser $\frac{1}{2}\|w\|^2$

$$\text{tel que} \qquad y_i(w.x_i + b) \geq 1$$

La technique des variables ressort permet de construire un hyperplan en admettant des erreurs,

mais en les minimisant, ce qui amène à assouplir les contraintes en introduisant les variables ressort $\xi_i \geq 0$ dans la définition des contraintes :

$$w.x_i + b \geq +1 - \xi_i \quad si\ y_i = +1$$

$$w.x_i + b \leq -1 + \xi_i \quad si\ y_i = -1$$

Ce qui s'écrit $$y_i(w.x_i + b) \geq 1 - \xi_i \quad \forall i = 1..n$$

Quand une erreur de classification intervient, la variable ξ_i a une valeur plus grande que 1, donc $\sum_i \xi_i$ est une borne supérieure du nombre d'erreurs à l'apprentissage.

De là, un moyen naturel pour pénaliser les erreurs est de remplacer la fonction précédente à minimiser par : $\frac{1}{2}\|w\|^2 + C\ \sum_{i=1}^{n} \xi_i$

Autrement dit, on cherche à maximiser la marge en s'autorisant pour chaque contrainte une erreur positive ξ_i la plus petite possible [31]. Le coefficient C est défini comme un paramètre de régularisation, il donne un compromis entre la marge et le nombre d'erreurs admissibles. L'ajout du terme $C\sum_{i=1}^{n} \xi_i$ peut être considéré comme une mesure d'une certaine quantité de mauvaise classification. Ainsi, une faible valeur de C entraîne une faible tolérance. D'autres formulations existent, comme $C\sum_{i=1}^{n} \xi_i^2$ [31].

En suivant la même démarche du Lagrangien que précédemment, nous aboutissons à la forme duale suivante :

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[y_i(\langle w, x_i\rangle + b - 1 + \xi_i] - \sum_{i=1}^{n}\mu_i\,\xi_i$$

$$\partial_w L(w,b,\alpha) = w - \sum \alpha_i y_i x_i = 0$$

$$\partial_b L(w,b,\alpha) = \sum \alpha_i y_i = 0$$

$$\partial_{\xi_i} L = C - \alpha_i - \mu_i = 0$$

$$L_D = \sum_{i=0}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

La seule différence avec le cas linéairement séparable est que

$$C - \alpha_i - \mu_i = 0$$

$$\mu_i \geq 0$$

$$\alpha_i \leq C \quad \forall\, i \in 1..n$$

Le problème d'optimisation est équivalent au cas de la marge maximale, avec une contrainte additionnelle ($\alpha_i \leq C$).

Cette formulation est connue comme «contrainte de boîte », car chaque valeur α_i est limitée par 0 d'un côté et par C de l'autre. C s'est révélé être un compromis entre la précision et la régularisation (le contrôle de l'erreur).

Les conditions KKT sont désormais :

$$\alpha_i | y_i(\langle w, x_i \rangle + b) - 1 + \xi_i |$$

$$\xi_i(\alpha_i - C) = 0 \qquad \forall\, i \in 1..n$$

Ces conditions impliquent que les variables d'écart soient différentes de zéro quand $\alpha_i = C$ c'est à dire, quand leur marge est moins de $1/\|w\|$ Les vecteurs pour lesquels

$0 \leq \alpha_i \leq C$: sont considérés comme vecteurs supports.

Il existe trois statuts différentiels :

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.
3. $\alpha_i = C$: L'exemple est mal classé. Il sera malgré tout considéré comme SV puisque $\alpha_i > 0$. Il s'agit d'un outlier.

La fonction de décision pour la classification de vecteurs inconnus **x** reste :

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

2.3.3.2 SVM non linéaire

Avant d'introduire la formulation d'un SVM non linéaire on présente tout d'abord les fonctions noyaux (où kernel en anglais)

2.3.3.2.1 Fonctions noyaux

Généralités

Le classificateur à marge maximale que nous venons de présenter, permet d'obtenir de très bons résultats lorsque les données sont linéairement séparables. L'intérêt principal d'un classificateur de ce type réside dans le fait que l'on en contrôle facilement la capacité et donc le pouvoir de généralisation. Naturellement, **un grand nombre de jeux de données sont non linéairement séparables**.

Pour classer ce genre de données on pourrait utiliser une fonction de décision non linéaire. Géométriquement, cela reviendrait à avoir une (hyper)courbe qui marquerait la frontière entre les exemples positifs et négatifs.

L'idée retenue dans SVM va dans un autre sens : on va tenter de trouver une transformation (mapping) de l'espace d'entrée vers un autre espace appelé « espace de re-description » (feature space) dans lequel les données sont linéairement séparables **[4].**

La figure (2.9) donne une représentation imagée de ce genre de transformation. Nous pouvons alors appliquer une méthode à marge maximale dans l'espace de re-description et

garder le contrôle (qui est indépendant de la dimension de l'espace dans lequel nous travaillons) que nous avions sur la capacité.

La dimension de l'espace des caractéristiques est généralement très élevée. Cela ne pose pas de problème pour notre classificateur à marge maximale vu que sa formulation duale fixe le nombre de variables à déterminer en fonction **de la taille de l'ensemble d'apprentissage.**

Nous noterons l'espace de caractéristique F, et la transformation vers cet espace : Φ

Φ : χ ===> F

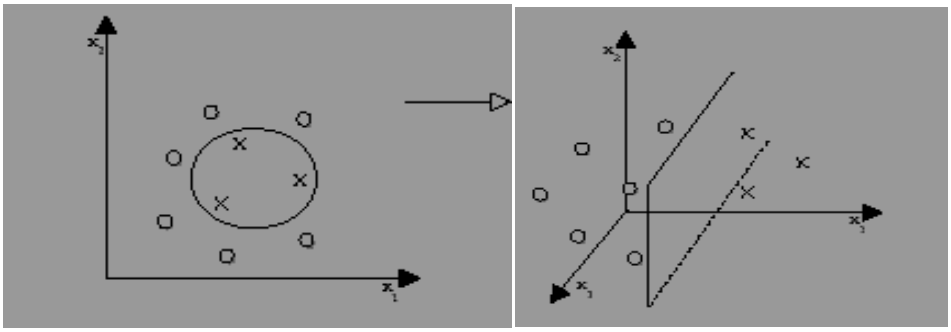


Figure 2.9 Une transformation Φ rendant les exemples linéairement séparables [4]

Comme on peut le remarquer à la figure (2.9) l'espace de caractéristiques est de dimension supérieure à celle de l'espace de départ. Les nouveaux axes contiennent une surgénération d'information par rapport aux précédents, ce qui permet idéalement d'effectuer une discrimination linéaire.

Avec cette logique, deux étapes se dégagent pour construire un SVM non linéaire :

1 Une transformation non linéaire pour placer les données dans l'espace de caractéristiques.
2 Un SVM linéaire pour classifier les vecteurs.

La normale de l'hyperplan séparateur dans le nouvel espace devient :

$$w = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i)$$

Les α_i doivent être calculés dans l'espace de caractéristiques. La classification d'un vecteur x inconnu se fait par la fonction : $f(x) = sign(w.\phi(x) + b)$

Ainsi, le classificateur résultant $f(x)$ exprime un séparateur non linéaire dans l'espace des exemples χ.

A titre d'exemple, étudions un ensemble de données à deux attributs x_1, x_2, que l'on transpose dans un espace des caractéristiques à cinq dimensions à l'aide de la transformation suivante :

$$\phi(x_1, x_2) = (x_1^2,\ x_2^2\ , x_1 x_2\ , x_1\ , x_2\ )$$

Par cette transformation, l'exemple x_a = (2 , 7) est associé au vecteur de caractéristiques

$\phi(x_a) = (4, 49, 14, 2, 7)$. Le classificateur linéaire dans $f(x)$ dans l'espace des caractéristiques exprime un séparateur quadratique dans l'espace des exemples (χ) :

$f(x_1, x_2) = sign(w_1 x_1^2 + w_2 x_2^2 + w_3 x_1 x_2 + w_4 x_1 + w_5 x_2 + b)$,

En particulier, si w=(1,1,0,0,0) et b=1, la frontière de séparation entre les exemples classés positivement et ceux classés négativement est le cercle de rayon unité dans l'espace des exemples.

- **Noyaux**

Le fait de représenter une fonction de décision complexe par un séparateur linéaire dans un espace de caractéristiques de très haute dimensionnalité peut engendrer un coût computationnel élevé. Fréquemment, l'étude des algorithmes d'apprentissage révèle que les seules opérations manipulant les vecteurs de caractéristiques consistent en des produits scalaires. La stratégie du noyau permet alors de substituer ces produits scalaires par une fonction $K(x, x')$ rapidement calculable. Le résultat de $K(x, x')$ doit être équivalent en produit scalaire entre $\phi(x) et\ \phi(x')$ dans un certain espace de caractéristiques F. Ainsi le noyau K est associé à ϕ lorsque : $K(x, x') = \phi(x).\phi(x')$

Nous établissons par la suite une condition permettant de déterminer si une fonction

χ x $\chi \rightarrow \Re$ correspond au produit scalaire dans un espace F est un noyau. Cette condition est appelée condition de Mercer.

- **Condition de Mercer [4]**

La matrice contenant les similarités entre tous les exemples de l'ensemble d'apprentissage.

$$G = \begin{pmatrix} K11\ k12 & k1n \\ & \\ Kn1 & knn \end{pmatrix}$$

est appelée matrice de Gram.

Nous énonçons à présent le théorème de Mercer fournissant une condition suffisante et nécessaire pour qu'une fonction soit un noyau.

- **Théorème** (condition de Mercer) [4]

La fonction k(x; z) : χ x $\chi \rightarrow \Re$ est un noyau si et seulement si:

$G = (K(x_i, x_j))_{i,j=1}^{n}$ est définie positive.

Notons qu'une fonction : $G = (K(x_i, x_j))_{i,j=1}^{n}$ générant une matrice définie positive possède les trois propriétés fondamentales du produit scalaire :

1. Positivité : $K(x_i, x_j) \geq 0$
2. Symétrie : $K(x_i, x_j) = K(x_j, x_i)$
3. Inégalité de Cauchy-Shwartz : $K(x_i, x_j) \leq \|x_i\|.\|x_j\|$

- Remarque

La matrice $G = (K(x_i, x_j))_{1 \leq i,j \leq n}$ dite matrice de gram a une importance cruciale dans les algorithmes à noyaux car c'est elle qui définit la complexité numérique de l'apprentissage: pour le problème de la classification SVM, elle permet de définir la partie quadratique de la forme quadratique à optimiser et elle contient aussi toutes les informations sur les données d'apprentissage et la fonction k .

- **Exemple de kernels (noyaux)**

- Le noyau linéaire : est un simple produit scalaire : $k(x, z) = \langle x, z \rangle$
- Le noyau polynomial permet de représenter des frontières de décision par des polynômes de degré d.

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

- Le noyau RBF : (Radial Basis Function), noyau très utilisé dans la pratique qui s'évalue selon

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

Ou σ est un réel positif qui représente la largeur de bande du noyau.

Les classificateurs exprimés par un noyau linéaire sont équivalents aux séparateurs linéaires exprimés directement dans l'espace des exemples (sans utiliser la stratégie du noyau). Aussi les variables d et σ des noyaux polynômiaux et RBF constituent des hyperparamètres des algorithmes d'apprentissage. Leur valeur modifie grandement le type de frontières de décision générées et, conséquemment la qualité des classificateurs résultants.

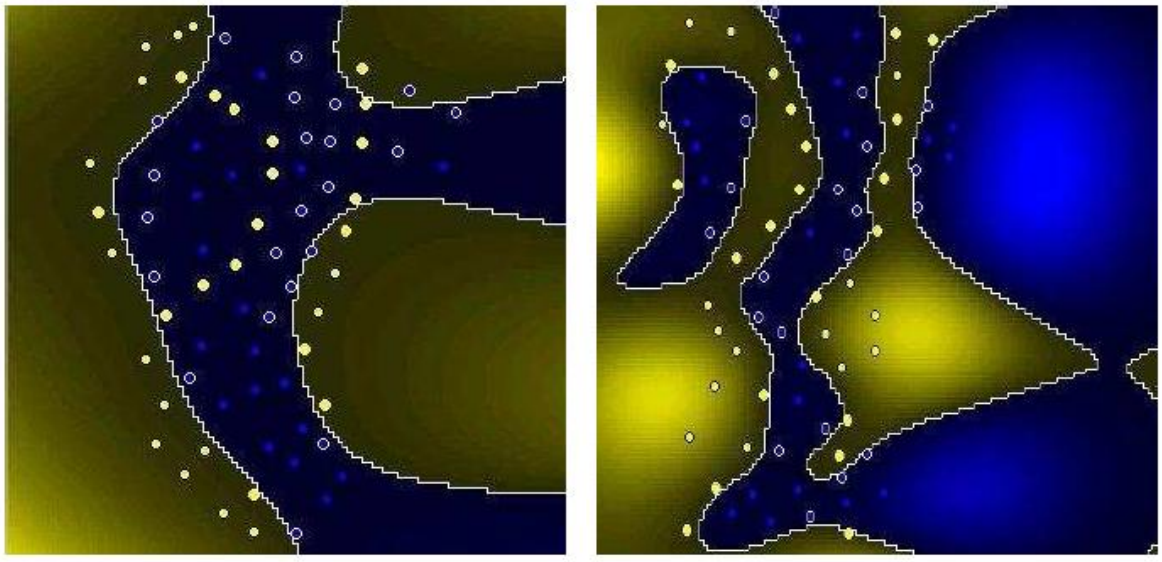


Figure 2.10 : à gauche noyau linéaire , à droite noyau polynômial [59]

La figure 2.10 illustre le type de frontières de décision qu'ils permettent d'obtenir sur un exemple d'ensemble de données dont le noyau polynômial classifie mieux les données que le noyau linéaire.

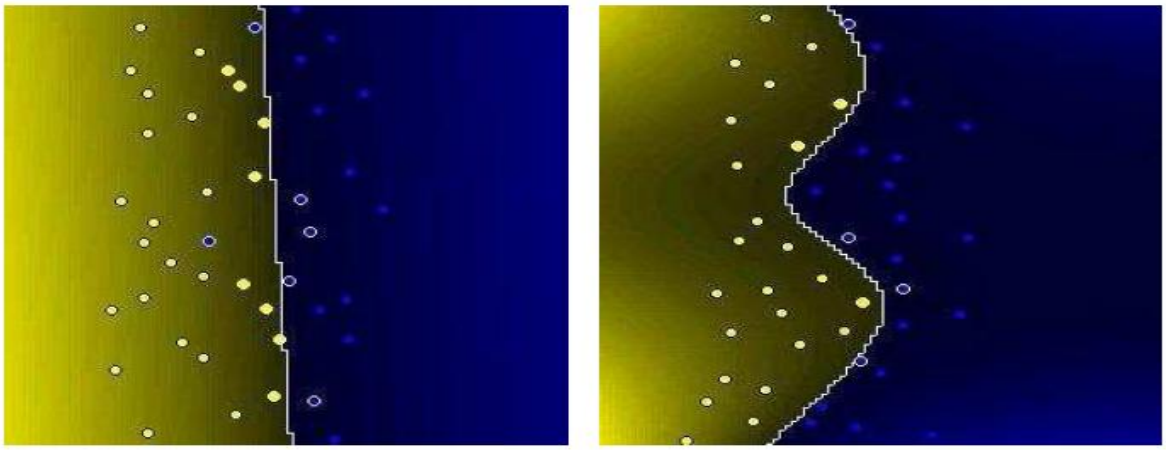


Figure 2.11 : à gauche noyau polynômial , à droite noyau RBF [59]

La figure 2.11 illustre le type de frontières de décision qu'ils permettent d'obtenir sur un exemple d'ensemble de données dont le noyau RBF classifie mieux les données que le noyau polynômial.

Composition de noyaux

Il est possible de composer des nouveaux noyaux en utilisant des noyaux existants. En prenant k1(: , :) et k2(: , :) des fonctions satisfaisant à la condition de Mercer, $a \in \Re^+$ et B une matrice définie positive, alors les fonctions suivantes sont des noyaux :

$$1) k(x,z) = k_1(x,z) + k_2(x,z)$$
$$2) k(x,z) = a k_1(x,z)$$
$$3) k(x,z) = k_1(x,z) k_2(x,z)$$
$$4) k(x,z) = x^T B z$$

La formulation duale d'un SVM non linéaire est la suivante :

$$\text{Maximiser} \qquad L_D(\alpha) = \sum_{i=0}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K_{ij}$$

$$\text{Tel que} \quad \begin{cases} \sum_{i=1}^{n} \alpha_i y_i = 0 \\ 0 \le \alpha \le C \end{cases} \qquad \text{QP2}$$

La fonction de décision pour la classification de vecteurs inconnus **x** reste :

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

L'introduction de noyaux permet donc aux machines à vecteurs supports de déterminer une surface de décision non linéaire en gardant un formalisme provenant d'une approche linéaire de la classification.

2.3.3.2.2 Principe de fonctionnement

La figure 2.12 montre l'enchaînement des traitements pour les méthodes à noyaux.

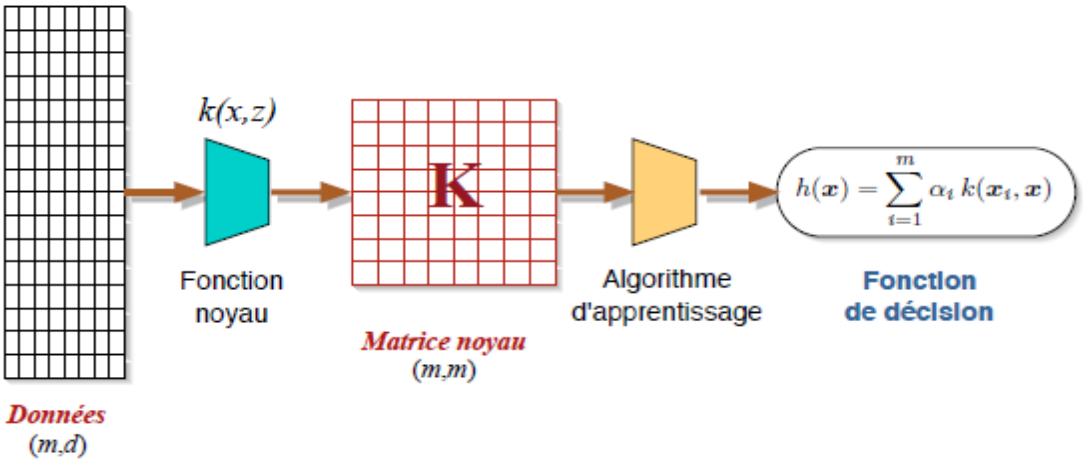


Figure 2.12 Chaîne de traitements génériques d'une méthode à noyau [22]

2.3.3.2.3 Architecture générale d'une machine à vecteurs supports

Une machine à vecteur support, recherche à l'aide d'une méthode d'optimisation, dans un ensemble d'exemples d'entrainement, des exemples, appelés vecteurs support, qui caractérisent la fonction de séparation. La machine calcule également des multiplicateurs associés à ces vecteurs.

Les vecteurs supports et leurs multiplicateurs sont utilisés pour calculer la fonction de décision pour un nouvel exemple. Le schéma de la figure 2.13 résume l'architecture générale d'une SVM dans le cas de la reconnaissance des chiffres manuscrits.

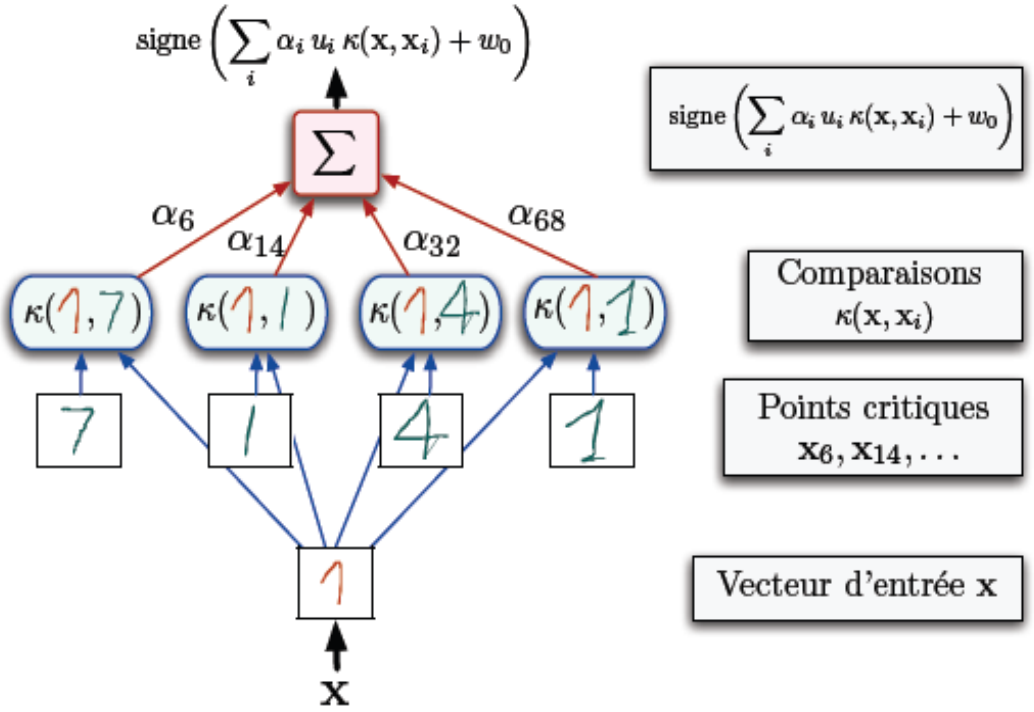


Figure 2.13 : Principe de fonctionnement d'un SVM (cas des chiffres manuscrits) [22]

La fonction noyau K est utilisée pour calculer la distance entre le vecteur à tester x et chaque vecteur support dans l'espace de caractéristique. Les résultats sont ensuite linéairement combinés en utilisant les multiplicateurs de Lagrange α_i et ajoutés au biais b. Le résultat final f permet de décider à propos du nouveau vecteur : si f(x) est positive, il s'agit du chiffre "1", sinon, il s'agit d'un autre chiffre.

2.3.4 Méthodes de résolution d'un SVM

On utilisera souvent la version matricielle du problème. Pour ce faire, on introduit la matrice Q telle que $Q_{ij} = y_i y_j k(x_i, x_j)$.Cette matrice, appelée hessienne, est donc très proche de la matrice de Gram et possède les propriétés de symétrie et de définition positive.

$$Maximiser\ \ w(\alpha) = \frac{1}{2}\,\alpha^T Q\,\alpha - 1^T\alpha$$

$$\text{Tel que} \begin{cases} y^T\ \alpha = 0 \\ 0\ \leq \alpha_i \leq C\ \ \forall\, i = 1, \dots, n \end{cases}$$

La résolution d'un problème d'optimisation quadratique est souvent amenée, en pratique, à résoudre certaines équations par des méthodes numériques. Plusieurs techniques sont proposées qui abordent plusieurs aspects pratiques intéressants pour les SVM pour la reconnaissance de formes [48, 66]

- **Le gradient conjugué avec contraintes :** c'est un gradient conjugué classique, dont les directions sont projetées dans les sous espaces définis par les contraintes

$$\sum_{i=1}^{n} \boldsymbol{\alpha_i\, y_i} = \mathbf{0}$$

- **Des méthodes de projection :** également basées sur le gradient conjugué.

- **La décomposition de Bunch-Kaufman :** qui utilise le *Hessienne*, tout en s'appuyant sur le fait que la plupart des $\boldsymbol{\alpha_i}$ **so**nt nuls.

- **Les méthodes de points intérieurs [50] :** par exemple l'algorithme de *Vanderbei*, méthode qui semble particulièrement intéressante lorsque les « *vecteurs de support VS* » sont nombreux par rapport à la taille de la base d'exemples.

2.4 Extension du SVM binaire au cas multi-classes

Les machines à vecteur supports sont dans leurs origines binaires. Cependant, les problèmes du monde réel sont dans la plupart des cas multi classe, l'exemple le plus simple en est la reconnaissance des caractères optiques (OCR). Dans de tels cas, on ne cherche pas à affecter un nouvel exemple à l'une de deux classes mais à l'une parmi plusieurs, c'est à dire que la décision n'est plus binaire et un seul hyperplan ne suffit plus.

Les méthodes des machines à vecteur support multi classe, réduisent le problème multi classe à une composition de plusieurs hyperplans bi classes permettant de tracer les frontières de décision entre les différentes classes [11]. Ces méthodes décomposent l'ensemble d'exemples en plusieurs sous ensembles représentant chacun un problème de classification binaire. Pour chaque problème un hyperplan de séparation est déterminé par la méthode SVM binaire.

On construit lors de la classification une hiérarchie des hyperplans binaires qui est parcourue de la racine jusqu'à une feuille pour décider de la classe d'un nouvel exemple. On trouve dans la littérature plusieurs méthodes de décomposition :

2.4.1 Approche Un-contre-Tous (1vsR)

L'idée de cette stratégie est de construire autant de classificateurs que de classes. Ainsi, durant l'apprentissage, tous les exemples appartenant à la classe considérée sont étiquetés positivement (+1) et tous les exemples n'appartenant pas à la classe sont étiquetés négativement (-1).

Un hyperplan H_k est défini pour chaque classe k par la fonction de décision suivante :

$$H_k(x) = sign(\langle w_k, x\rangle + b_k)$$

$$= \begin{cases} +1, & si\ f_k(x) > 0 \\ 0, & sinon \end{cases}$$

$$k^* = \text{Arg}_{1\leq k\leq K}\ Max(H_k(x) \qquad (2.2)$$

Si une seule valeur $H_k(x)$ est égale à 1 et toutes les autres sont égales à 0, on conclut que x appartient à la classe k.

Or, il est possible que plusieurs sorties soient positives pour un exemple de test donné. Ceci est particulièrement le cas des données ambiguës situées près des frontières de séparation des classes. On utilise dans ce cas un vote majoritaire pour attribuer l'exemple x à la classe k selon la formule (2.3).

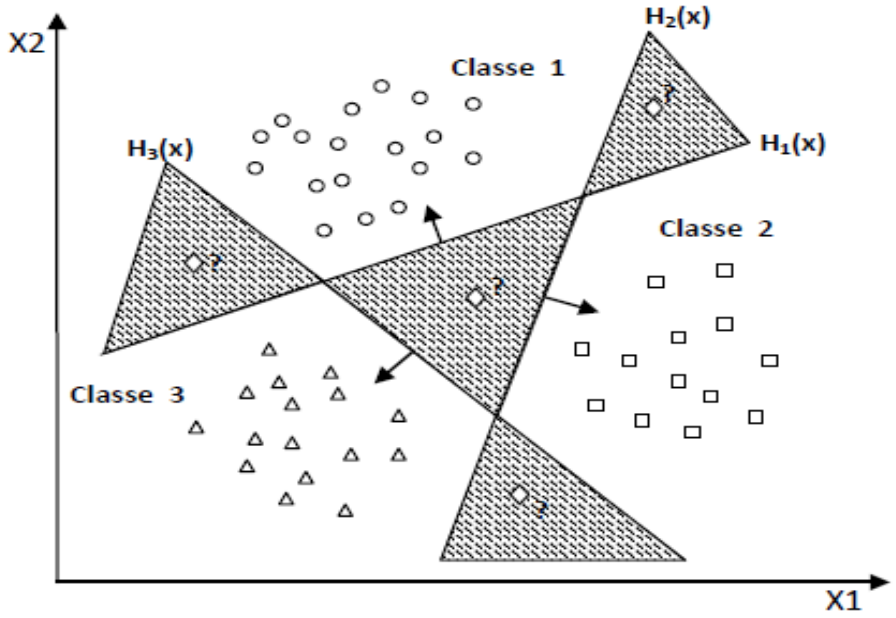


Figure 2.14 : Nuage de points à 3 classes : l'approche un contre Tous[57]

$$k^* = \text{Arg}_{1 \leq k \leq K} \; Max(\langle w_k, x \rangle + b_k) \quad (2.3)$$

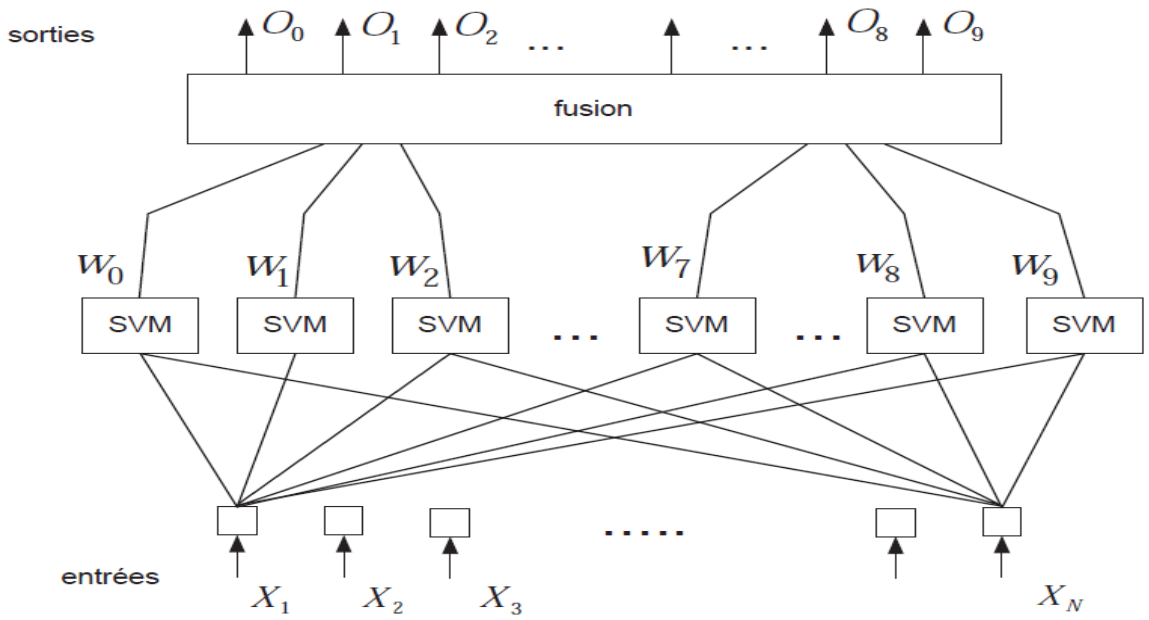


Figure 2.15 : Architecture du système en stratégie Un-contre-Tous [20]

Souvent, la méthode 1vsR est critiquée à cause de son asymétrie [33], puisque chaque hyperplan est entrainé sur un nombre d'exemples négatifs beaucoup plus important que le nombre d'exemples positifs. Par exemple dans le cas de l'OCR, le classificateur du caractère 'A' est entrainé sur des exemples positifs représentant 'A' et des exemples négatifs représentant tous les autres caractères. La méthode une contre une suivante est une méthode symétrique qui corrige ce problème.

2.4.2 Approche Un-contre-Un (1vs1) [20]

L'approche Un-Contre-Un est un cas spécial des méthodes de décomposition proposées par Dietterich et al. [29] pour résoudre des problèmes à plusieurs classes. Cette approche requiert la construction de $K(K-1)/2$ SVM chacun séparant un couple de classes (i, j) parmi ceux existants.

Pendant la classification, un vecteur d'entrée x est présenté à l'ensemble des classificateurs construits. La sortie de chaque SVM fournit un vote partiel concernant uniquement le couple de classes(w_i, w_j). En considérant que chaque SVM calcule un estimé $\hat{p}_{ij}$ de la probabilité :

$$p_{ij} = P(x \in w_i | x, x \in w_i \cup w_j) \quad (2.3)$$

alors la règle de classification la plus simple peut s'écrire :

$$\arg max_{1 \leq i \leq k} \sum_{j \neq i} [\hat{p}_{ij} > 0.5] \quad (2.4)$$

L'opérateur [ ] est défini :

$$[\eta] = \begin{cases} 1 \; si \; \eta \; est \; vrai \\ 0 \quad sinon \end{cases}$$

Cette combinaison considère que les sorties des SVM sont des valeurs binaires de 1 ou 0. Une autre approche de reconstruction pourrait tirer avantage de l'information de confiance associée à chacune des sorties $\hat{p}_{ij}$. Dans l'hypothèse que ces valeurs représentent des probabilités, il est possible d'estimer une approximation $\hat{p}_i$ de la probabilité à posteriori

$$p_i = P(x \in w_i | x)$$

En considérant la matrice carrée $\hat{P}$ avec les entrées $\hat{p}_{ij}$ tels que $(i,j)_{i,j=1,\ldots,K \; ,i \neq j}$

et avec $\hat{p}_{ji} = 1 - \hat{p}_{ij}$. les valeurs de $\hat{p}_i$ peuvent être calculées par :

$$\hat{p}_i = \frac{2}{K(K-1)} \sum_{j \neq i} \hat{p}_{ij} \qquad (2.5)$$

Et la règle de décision s'écrit : $\arg max_{1 \leq i \leq K} \hat{p}_i$ (2.6)

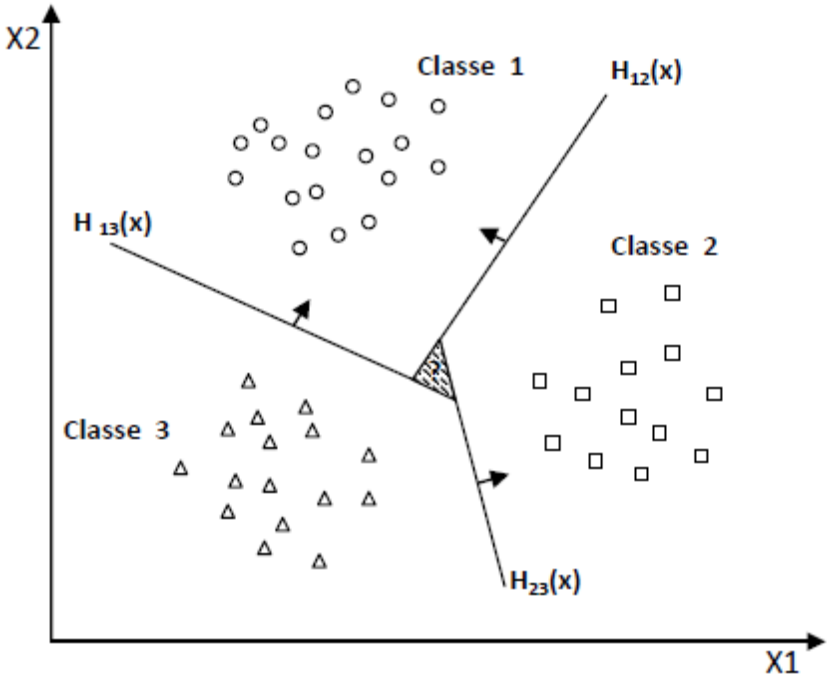


Figure 2.16 : Nuage de points à 3 classes : l'approche un contre un [57]

La figure 2.17 présente l'architecture simplifiée du système en stratégie Un contre- Un.

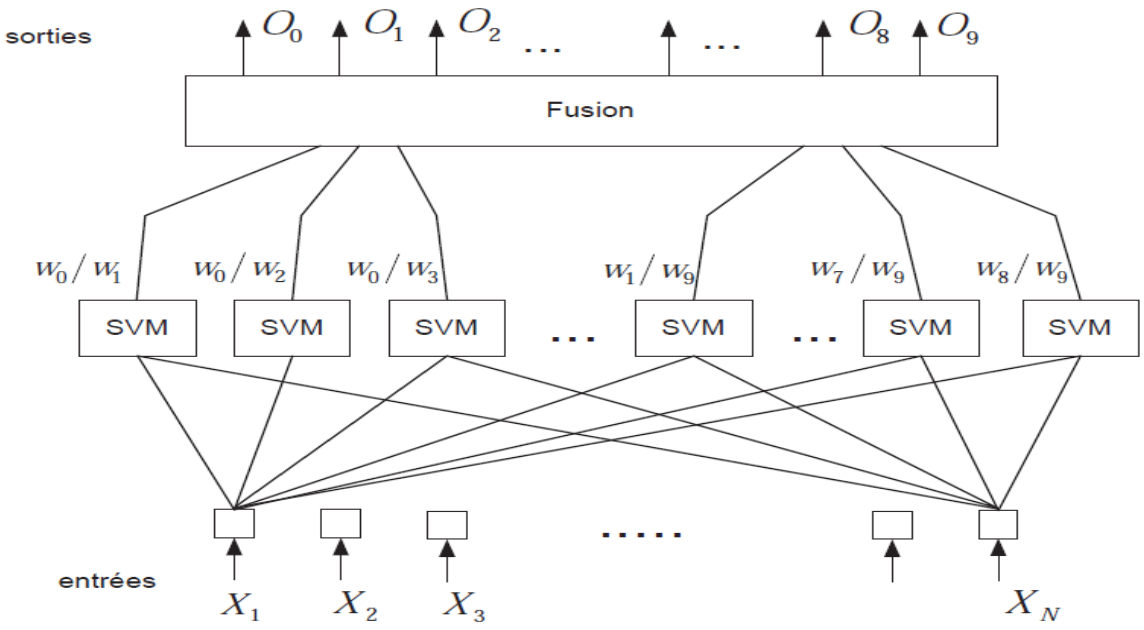


Figure 2.17 Architecture du système en stratégie Un-contre-Un [20]

2.4.3 Graphe de décision [57]

C'est une méthode développée par Platt et all [87] pour résoudre le problème des zones d'indécision dans la méthode 1vs1. Premièrement, l'entrainement est effectué par la même méthode 1vs1 de la section précédente pour obtenir K(K - 1)/2 hyperplans. Puis, au lieu d'utiliser le vote pour l'affectation des nouveaux exemples, on construit un graphe de décision. Pour cela, on définit une mesure E_{ks} de la capacité de généralisation sur les différents hyperplans obtenus c-àd pour chaque paire de classes. Cette mesure représente le rapport entre le nombre de vecteurs supports de l'hyperplan et le nombre d'exemples des deux classes correspondante.

$$E_{ks} = \frac{N_{vs}}{N_{exemples}}$$

Après la phase d'apprentissage on construit un graphe de décision qui sera utilisé pour la classification selon les étapes suivantes [57] :

1. Créer une liste L contenant toutes les classes,
2. Si L contient une seule classe, créer un noeud étiqueté de cette classe et arrêter.
3. Calculer pour chaque paire de classes (i, j) la capacité de généralisation E_{ij} de l'hyperplan obtenu dans la phase d'entrainement 1vs1,
4. Rechercher les deux classes k et s dont E_{ks} est maximum,
5. Créer un noeud N du graphe étiqueté de (k, s).
6. Créer un graphe de décision à partir de la liste $L - \{k\}$, de la même manière, et l'attacher au fils gauche de N,
7. Créer un graphe de décision à partir de la liste $L - \{s\}$, de la même manière, et l'attacher au fils droit de N. On obtient ainsi un graphe de décision similaire à l'exemple de la figure 2.18 dont les feuilles sont les classes et les noeuds internes sont les hyperplans :

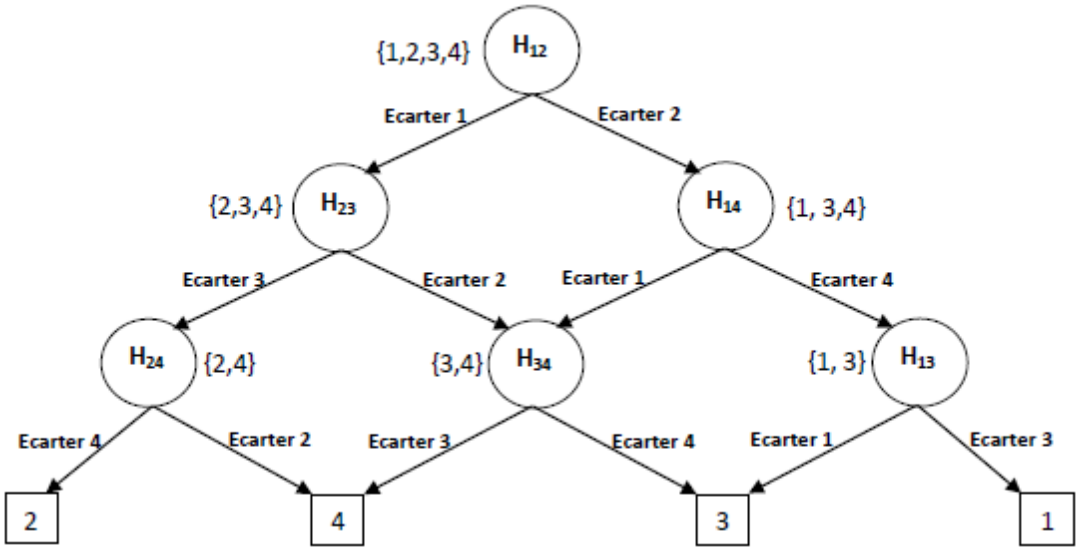


Figure 2.18 : graphe de décision acyclique orienté à quatre classes [57]

2.4.4 SVMs basées sur les arbres de décision [57]

Dans cette méthode, on apprend pour K classes, (K - 1) hyperplans. Chaque hyperplan sépare une ou plusieurs classes du reste, selon un découpage choisi. On peut choisir, par exemple, un découpage semblable à la méthode 1vsR où l'hyperplan H_i sépare la classe i des classes i+1, i + 2,...,K (cf. figure 2.19).

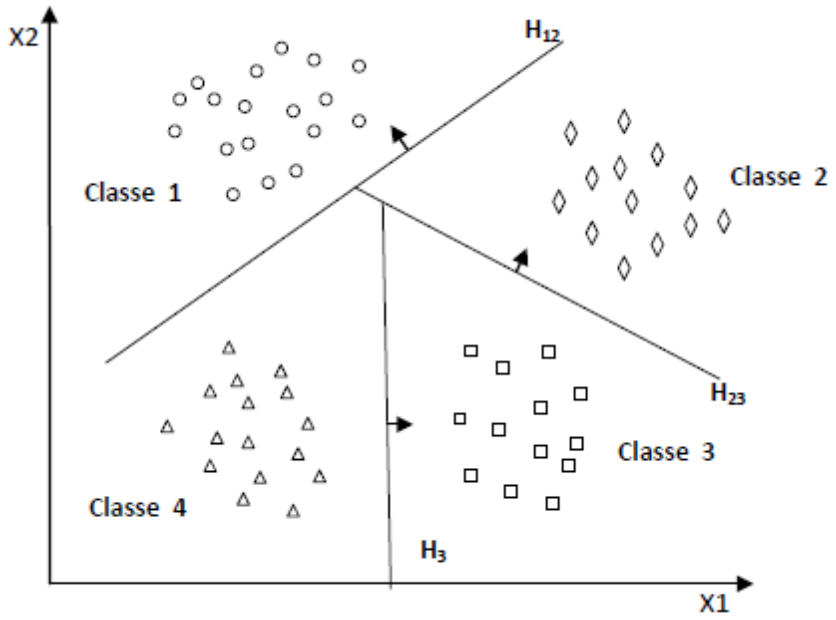


Figure 2.19 : SVM multiclasse par arbre de décision [57]

Dans la phase de classification, pour classer un nouvel exemple x, on teste les hyperplans dans l'ordre croissant et on s'arrête sur le premier hyperplan qui retourne une valeur de décision positive. L'exemple x appartient alors à la classe positive de cet hyperplan.

Il existe plusieurs types d'arbres de décision. Certains trient les classes selon l'ordre décroissant du nombre de leurs exemples, pour placer les classes volumineuses dans des zones importantes. D'autres méthodes [88] subdivisent, à chaque fois, les classes en deux ensembles équilibrés, ce qui génère un arbre de décision binaire équilibré.

Les méthodes basées sur les arbres de décisions sont généralement plus rapides que la méthode 1vsR. Cela est dû au fait que la méthode 1vsR utilise, pour entrainer chaque hyperplan, tous les exemples, tandis que dans les méthodes basées sur les arbres de décision, le nombre d'exemples d'entrainement diminue en descendant dans l'arbre.

2.5 SVM monoclasse

Dans les machines à vecteur support binaires et multiclasse précédentes, nous avons toujours des exemples positifs et d'autres négatifs c-à-d des exemples et des contre-exemples. De telles informations ne sont pas disponibles dans tous les cas d'application. Parfois, il est très coûteux, voire impossible, de trouver des contre-exemples qui représentent réellement la classe négative.

Pour la classification SVM monoclasse, il est supposé que seules les données de la classe cible sont disponibles. L'objectif est de trouver une frontière qui sépare les exemples de la classe cible du reste de l'espace, autrement dit, une frontière autour de la classe cible qui accepte autant d'exemples cibles que possible [89,57]. Cette frontière est représentée par une fonction de décision positive à l'intérieur de la classe et négative en dehors. La figure 2.20 représente, en deux dimensions, un cas de séparation d'une classe de toute autre classe.

Le problème est modélisé par le problème primal de programmation quadratique de l'équation (2.7) dont l'objectif est de maximiser la marge et minimiser les erreurs de classification. La contrainte est la bonne classification des exemples d'entrainement [57].

$$\begin{cases} min_{w,\xi,\rho} \dfrac{1}{2}\|w\|^2 + \dfrac{1}{v^N}\displaystyle\sum_{i=1}^{l}\xi_i - \rho \\ \langle w, \phi(x_i)\rangle \geq \rho - \xi_i \\ \xi_i \geq 0 \quad i = 1,2,\ldots.N \end{cases} \qquad (2.7)$$

Où N est le nombre d'exemples de la classe cible, (w, ρ) les paramètres permettant de localiser l'hyperplan, ξ_i représentent les erreurs permises sur les exemples, pénalisés par le paramètre v et ϕ est une transformation d'espace semblable à celle du cas binaire.

Une fois (w, ρ) déterminés, tout nouvel exemple pourra être classé par la fonction de décision de l'équation (2.8) :

$$f(x) = \langle w, \phi(x) \rangle - \rho \quad (2.8)$$

x appartient à la classe cible si $f(x)$ est positive.

En fait, la résolution du problème de l'équation 2.7 est réalisée par l'introduction des multiplicateurs de Lagrange pour obtenir le problème dual de l'équation (2.9) :

$$\begin{cases} min_{\alpha} & \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \\ & sc \sum_{i=1}^{n} \alpha_i \\ & 0 \leq \alpha_i \leq \frac{1}{v^N} \end{cases} \qquad (2.9)$$

Où K est un noyau qui représente la transformation d'espace ϕ .

Une fois les α_i déterminés ils peuvent être dans l'un des trois cas suivants :

- $\alpha_i = 0$: correspondent aux exemples bien classés c-à-d qui se situent au dessus de l'hyperplan,
- $\alpha_i = \frac{1}{v^N}$: correspondent aux exemples qui se situent à l'intérieur de la marge (au dessous de l'hyperplan),
- $0 \leq \alpha_i \leq \frac{1}{v^N}$: correspondent aux exemples vecteurs support qui se situent sur l'hyperplan.

La fonction de décision pour tout exemple x est donnée par l'équation 2.10

$$f(x) = \sum_{i=1}^{l} \alpha_i K(x_i, x) - \rho \qquad (2.10)$$

Où ρ peut être déterminé à partir d'un exemple x_i d'apprentissage dont $\alpha_i \neq 0$ par l'équation 2.11 : $\rho = \sum_j \alpha_j K(x_j, x)$ (2.11)

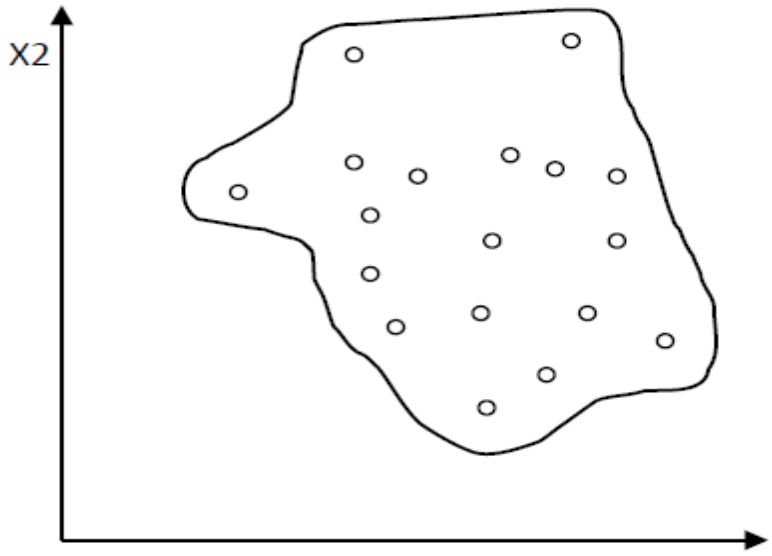


Figure 2.20 : séparation des exemples d'une classe du reste de l'espace [57]

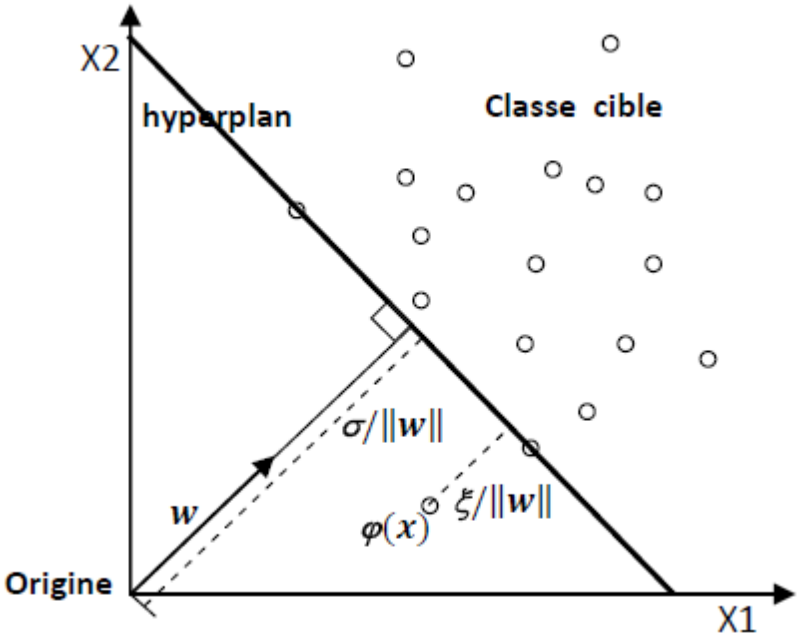


Figure 2.21 : SVM monoclasse à marge maximale [57]

2.6 Conclusion

SVM est une méthode de classification qui montre de bonnes performances dans la résolution de problèmes variés. Cette méthode a montré son efficacité dans de nombreux domaines d'applications tels que le traitement d'image, la catégorisation de textes ou le diagnostique médical et ce même sur des ensembles de données de très grandes dimensions

La réalisation d'un programme d'apprentissage par SVM se ramène à résoudre un problème d'optimisation impliquant un système de résolution dans un espace de dimension conséquente. L'utilisation de ces programmes revient surtout à sélectionner une bonne famille de fonctions noyau et à régler les paramètres de ces fonctions. Ces choix sont les plus souvent faits par une technique de validation croisée, dans laquelle on estime la performance du système en la mesurant sur des exemples n'ayant pas été utilisés en cours d'apprentissage.

L'idée est de chercher les paramètres permettant d'obtenir la performance maximale. Si la mise en œuvre d'un algorithme de SVM est en général peu coûteuse en temps, il faut cependant compter que la recherche des meilleurs paramètres peut requérir des phases de test assez longues.

La méthode des SVM est applicable pour des tâches de classification à deux classes, mais il existe des extensions pour la classification multi classe. Les SVM peuvent également s'utiliser pour des tâches de régression, c'est-à-dire de prédiction d'une variable continue en fonction d'autres variables, comme c'est le cas par exemple dans de la prédiction de consommation électrique en fonction de la période de l'année, de la température, etc.

Le champ d'application des SVM est donc large. Elles représentent une classe de méthodes très séduisantes. L'un des axes de recherche actuel est de parvenir à coder des connaissances *a priori* dans ces systèmes, c'est-à-dire à mieux comprendre le rôle des fonctions noyau.

Chapitre 3

Mise En Œuvre Des Machines à Vecteurs Supports

3.1 Introduction

Un algorithme résolvant les SVM identifie parmi les exemples d'apprentissage quels sont les vecteurs supports et construit la frontière (ou fonction de décision) avec une combinaison linéaire de cette sélection. Résoudre ce problème équivaut à résoudre un programme quadratique sous contraintes de boîtes.

Dans ce chapitre nous nous intéressons à une étude détaillée de deux méthodes hors ligne dédiées aux SVM. Notre choix est fixé à deux implémentations les plus reconnues.

- LIBSVM : Utilise l'algorithme SMO de « Platt »
- SVMLIGHT : Utilise l'algorithme de « Joachims »

L'objectif de cette étude est double, En premier lieu pour pouvoir étudier et implémenter des méthodes incrémentales, on doit tout d'abord comprendre la mise en œuvre des méthodes classiques ou hors ligne. En second lieu comparer les performances en généralisation de ces deux types de méthodes (hors ligne et incrémental) en faisant des expérimentations sur différentes bases de données. Les deux méthodes incrémentales vont êtres exposées dans le suivant chapitre.

3.2 Formulations du problème SVM

Avant d'entamer l'étude de la mise en œuvre des machines à vecteurs supports rappelons tout d'abord la formulation duale du problème d'optimisation (cas général)

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

$$\text{Tel que} \quad \begin{cases} \sum_{i=1}^{n} \alpha_i y_i = 0 \\ 0 \le \alpha \le C \end{cases} \tag{3.1}$$

3.3 Résolution du problème lié à l'apprentissage d'une SVM

L'entraînement d'une machine à Vecteurs de Support consiste à résoudre le problème d'optimisation quadratique convexe (3.1). Le choix de la technique de résolution numérique est critique car les performances de l'implémentation en seront directement tributaires.

En raison de son immense taille, le problème (3.1) qui résulte de l'approche SVM ne peut pas être résolu facilement par l'intermédiaire des techniques standards de programmation quadratique (PQ).

La forme quadratique dans (3.1) implique une matrice qui a un nombre d'éléments égal au carré du nombre d'exemples d'entraînement. Cette matrice ne peut pas être traitée correctement avec une RAM de 128 méga-octets s'il y a plus de quatre milles (4000) exemples d'entraînement ou chaque élément de la matrice est stocké en double précision (8 octets) .

Seulement, il est possible de dériver des algorithmes qui exploitent la forme particulière de la fonction objective duale du SVM. Dans cette section, nous allons présenter deux approches différentes pour la résolution du problème quadratique du SVM.

3.3.1 Méthode de chunking

La résolution de la fonction objective duale de l'équation (3.1) avec un très grand nombre d'exemples donne lieu à un vecteur α creux. Selon les données, plusieurs des paramètres α sont soit nuls ou égales à C. S'il y a moyen de savoir a priori lesquels α seront nuls, il est possible de réduire la taille de la matrice K sans altérer la valeur de la fonction objective. Aussi, une solution α est valide si et seulement si elle respecte les conditions de KKT.

Vapnik [30] était le premier à décrire une méthode qui exploite cette propriété en prenant en compte seulement les α non nuls ou ceux violant les conditions de Karush Kuhn Tucker. La taille de ce sous ensemble dépend du nombre de vecteurs supports, de la taille des données et de la complexité du problème de classification. Cette méthode se comporte assez bien sur des problèmes de quelques centaines de vecteurs supports.

3.3.2 Méthode de décomposition successive

Cette méthode est similaire à celle du «Chunking» dans la mesure où elle considère aussi une succession de sous problèmes quadratiques à résoudre. La différence est que la taille des sous problèmes retenus est fixe. Cette méthode est basée sur la constatation qu'une succession de sous-problèmes quadratiques ayant au moins un exemple qui ne vérifie pas les conditions de KKT converge toujours vers une solution optimale [2,7].

Osuna et al [2] Suggèrent de conserver la taille du sous-problème fixe et d'ajouter ou d'enlever un exemple à la fois. Ceci permet d'entraîner de gros ensembles de données. En pratique, cette stratégie peut converger lentement si diverses heuristiques ne sont pas prises en compte. En effet, il est possible d'adopter des stratégies sophistiquées afin d'inclure ou d'exclure quelques exemples de la fonction objective.

Différentes stratégies de cache peuvent aussi accélérer la convergence même avec quelques milliers de vecteurs supports. L'algorithme de SVMlight est une implémentation de cette méthode [12]. Un package d'optimisation quadratique reste toutefois nécessaire. Nous décrirons en profondeur cet algorithme dans ce qui suit :

3.3.2.1 Algorithme de Joachims

L'algorithme de Joachims [12] est une implémentation de SVMlight. Il peut traiter un gros ensemble de données allant jusqu'à quelques centaines de milliers d'exemples.

Comme vu précédemment, la fonction objective duale à minimiser s'écrit :

$$w(\alpha) = -\alpha^T 1 + \frac{1}{2}\alpha^T K\alpha \qquad (3.2)$$

Sujet à $$\alpha^T y = 0 \qquad (3.3)$$

$$0 \leq \alpha \leq C \qquad (3.4)$$

La taille du problème à optimiser dépend du nombre d'exemples d'apprentissage l. Comme la taille de la matrice K est l^2, résoudre 10000 données d'apprentissage ou plus devient impossible à moins de garantir un espace mémoire suffisant de l'ordre de 10^8. Une alternative serait d'estimer la matrice K chaque fois qu'il est nécessaire

Une autre approche possible à cette limitation est de décomposer le problème en une série de tâches moins complexes et plus faciles à résoudre. SVMlight est une implémentation modifiée de l'algorithme de décomposition successive de Osuna et al. [2]. Cette méthode intègre les idées suivantes :

- Une stratégie de sélection d'un ensemble actif (sous ensemble des données dont les multiplicateurs α_i sont variables).

- Décomposition successive du problème. Ceci exploite les deux propriétés suivantes :

- Il existe beaucoup moins de vecteurs supports que d'exemples d'apprentissage,
- Beaucoup de vecteurs supports ont leurs multiplicateurs α_i égaux à C.

- Accélération de la convergence en utilisant la technique du cache ('caching') et la mise à jour incrémentale des valeurs de α_i.

- **Stratégie de décomposition**

L'idée s'inspire de la stratégie de décomposition utilisée par Osuna et al. [2] qui utilise une stratégie similaire à celle des ensembles actifs [39]. A chaque itération, les paramètres α_i sont partagés en deux catégories.

– Un ensemble B de paramètres variables.

– Un ensemble N de paramètres fixes.

Les paramètres de B sont mis à jour à chaque itération, alors que les paramètres de N sont temporairement fixés à leurs valeurs précédentes. L'ensemble B est aussi appelé ensemble actif. Il a une taille q largement inférieure à l.

L'algorithme fonctionne comme suit :

- Tant que les conditions de KKT ne sont pas remplies :
- Sélectionner q paramètres pour l'ensemble actif B. Les variables l−q restantes sont fixées à leurs valeurs courantes.
- Décomposer le problème initial en sous-problème QP et optimiser sur l'ensemble actif B.
- Terminer et retourner α.

Comme la matrice Hessienne K est garantie semi-définie positive, et toutes les contraintes de l'objective sont linéaires, le problème est convexe, et les conditions de KKT sont alors des conditions nécessaires et suffisantes d'optimalité.

- **Sous-problème QP**

Si les conditions d'optimalité ne sont pas satisfaites, l'algorithme décompose la fonction objective originale et résout une succession de sous-problèmes dérivés.

La décomposition de la fonction objective originale W(α) (équation 3.2) est garantie de converger vers une solution optimale si l'ensemble actif B respecte quelques conditions [2]. En particulier, l'ensemble des paramètres variables B est séparé de N. Aussi, nous supposons que α, y et K sont adéquatement arrangés tel que :

$$\alpha = \begin{vmatrix} \alpha_\beta \\ \alpha_N \end{vmatrix}, y = \begin{vmatrix} y_B \\ y_N \end{vmatrix}, \qquad K = \begin{vmatrix} K_{BB} & K_{BN} \\ K_{NB} & K_{NN} \end{vmatrix}$$

Le problème d'optimisation quadratique original peut s'écrire :

$$\min_{\alpha} w(\alpha) = -\alpha_B^T(1 - K_{BN}\alpha_N) + \frac{1}{2}\alpha_B^T K_{BB}\alpha_B + \frac{1}{2}\alpha_N^T K_{NN}\alpha_N - \alpha_N^T\, 1$$

$$\text{Sujet à :} \qquad \alpha_B^T y_B + \alpha_N^T y_N = 0$$
$$0 \le \alpha \le C$$

Comme les paramètres dans N sont fixes, les termes $\frac{1}{2}\alpha_N^T K_{NN}\alpha_N \;\; et \;\; \alpha_N^T\, 1$ sont constants et peuvent être omis de la fonction objective. Il en résulte un sous problème beaucoup moins complexe qui peut être résolu avec des packages standards d'optimisation quadratique.

Le choix de l'ensemble actif B contenant les paramètres variables est traité ci-dessous.

- **Sélection de l'ensemble actif**

La stratégie de sélection de l'ensemble actif consiste à choisir les paramètres α_i qui garantissent la convergence la plus rapide vers le minimum de (α). Cette technique repose sur la méthode de Zoutendjik qui utilise une approximation du premier ordre de la fonction désirée [48].

L'idée est de trouver la direction de la plus grande pente de descente d utilisant seulement q valeurs non nulles parmi les α_i. Ces q paramètres constituent l'ensemble actif B. Cette approche nécessite la résolution de la fonction objective suivante :

$$\min_{\alpha} v(d) = g(\alpha^t)^T d \quad (3.5)$$

Sujet à $y^T d = 0$ (3.6)

$d_i \geq 0 \quad pour\ i:\ \alpha_i = 0$ (3.7)

$d_i \leq 0 \quad pour\ i: \alpha_i = C$ (3.8)

$$-1 \leq d \leq 1 \quad (3.9)$$

$$|\{d_i : d_i \neq 0\}| = q \quad (3.10)$$

La fonction objective de l'équation 3.5 cherche une direction de descente. Une direction de descente a un produit scalaire négatif avec le gradient g(α T) au point courant α (t). Les contraintes des équations 3.6, 3.7 et 3.8 permettent de projeter la direction de descente sur le plan défini par l'égalité de l'équation 3.3 et de forcer la contrainte de l'équation 3.4. La contrainte donnée dans l'équation 3.9 normalise la taille du vecteur pour que le problème d'optimisation soit bien posé. Enfin, l'équation 3.10 garantie que **d** contient seulement q variables non nulles. Ces dernières seront incluses dans l'ensemble actif B.

3.3.2.2 La méthode d'Optimisation Séquentielle Minimale (SMO)

John Platt a proposé en1999 un nouvel algorithme pour l'entraînement des SVM qu'il a appelé algorithme d'Optimisation Séquentielle Minimale (SMO)[3]. Cet algorithme, décompose le problème de PQ (3.1) mais choisi de résoudre le plus petit sous-problème possible à chaque étape d'optimisation de la fonction objectif.

La méthode d'optimisation par minimisation séquentielle peut être perçue comme le cas extrême des méthodes de décomposition successive.

Cela est réalisable en choisissant d'optimiser, non pas un ensemble de multiplicateurs de Lagrange à la fois, mais deux seulement. En effet, il nécessite plus d'itérations pour converger mais chaque itération effectue moins d'opérations.

3.3.3.2.1 Implémentation de l'algorithme SMO

L'algorithme SMO est basé sur trois éléments :

1) Une méthode analytique pour résoudre les deux multiplicateurs de Lagrange.

2) Deux heuristiques pour choisir quels multiplicateurs à optimiser.

3) Une méthode pour le calcul du paramètre b .

- **Principe de l'algorithme** [92]

L'idée principale de cet algorithme est de décomposer le problème à l'extrême en optimisant uniquement deux points à chaque itération. L'avantage de ceci est qu'optimiser une équation à deux variables est un problème qui a une solution analytique.

- Optimiser deux α_i

Il s'agit de l'itération principale de SMO, nous allons la détailler point par point en se basant sur des relations entre les variables.

- Lien entre les α

Considérons deux coefficients quelconques choisis parmi l'ensemble des alphas. Dans un but de simplification des calculs, nous supposerons par la suite que ces deux coefficients sont respectivement le premier et le second des α ce sont donc α_1 et α_2

Ces deux coefficients ont chacun une ancienne valeur à optimiser : α_1^{old} et α_2^{old}
Pour la phase d'initialisation, ces deux anciennes valeurs pourront être fixées à 0.

Les autres coefficients α de l'équation sont considérés comme étant constants et fixés d'après

la contrainte $\sum_{i=1}^{l} \alpha_i y_i = 0$, nous pouvons établir que :

$$y_1\alpha_1 + y_2\alpha_2 = y_1\alpha_1^{old} + y_2\alpha_2^{old} = constante \qquad (3.11)$$

Il est donc possible d'établir une relation entre α_1 et α_2 :

$$\begin{aligned} \alpha_1 &= \frac{1}{y_1}\left(y_1\alpha_1^{old} + y_2\alpha_2^{old} - y_2\alpha_2\right) \\ &= \alpha_1^{old} + \frac{y_2}{y_1}\alpha_2^{old} - \frac{y_2}{y_1}\alpha_2 \\ &= \alpha_1^{old} + y_1y_2(\alpha_2^{old} - \alpha_2) \end{aligned}$$

Le calcul d'un α_2 optimal seul permet alors de résoudre le problème d'optimisation de ces deux points.

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

- Ajustement des α selon les contraintes de faisabilité

Afin de s'assurer d'avoir toujours une solution réalisable, il faut vérifier que les valeurs qui seront calculées soient conformes aux contraintes de faisabilité. En effet, les deux coefficients α sont bornés par le coefficient de marge d'erreur.

$$0 \le \alpha_1, \alpha_2 \le C$$

Appliqué à une valeur α_2^* résultant d'une itération d'optimisation de l'algorithme, cela donne une première contrainte

$$0 \le \alpha_2^* \le C \qquad (3.12)$$

Etant donné l'équation (3.11) et le fait que ($y_1y_2 \in \{-1, +1\}^2$, il est possible de prévoir une contrainte supplémentaire changeant selon les cas le résultat de classification :

1- Les deux valeurs d'appartenance aux classes sont égales, y1=y2

$$y_1 = y_2 \Longrightarrow \alpha_1 + \alpha_2 = \alpha_1^{old} + \alpha_2^{old}$$

Donc $\alpha_1 = \alpha_1^{old} + \alpha_2^{old} - \alpha_2$

$0 \leq \alpha_1 \leq C$, donc $0 \leq \alpha_1^{old} + \alpha_2^{old} - \alpha_2 \leq C$

Soit $\alpha_1^{old} + \alpha_2^{old} - C \leq \alpha_2 \leq \alpha_1^{old} + \alpha_2^{old}$ (3.13)

Donc finalement, en prenant en compte les deux contraintes (3.11) et (3.13) , il advient que pour y1=y2 il faut avoir :

$$\max(0, \alpha_2^{old} + \alpha_1^{old} - C) \leq \alpha_2 \leq \min(C, \alpha_1^{old} + \alpha_2^{old})$$

2- Les deux valeurs d'appartenance aux classes différentes, $y_1 \neq y_2$

$$y_1 \neq y_2 \Longrightarrow \alpha_1 - \alpha_2 = \alpha_1^{old} - \alpha_2^{old}$$

Donc $\alpha_1 = \alpha_1^{old} - \alpha_2^{old} + \alpha_2$

$0 \leq \alpha_1 \leq C$, donc $0 \leq \alpha_1^{old} - \alpha_2^{old} + \alpha_2 \leq C$

Soit $\alpha_2^{old} - \alpha_1^{old} \leq \alpha_2 \leq C - \alpha_1^{old} + \alpha_2^{old}$ (3.14)

Donc finalement, en prenant en compte les deux contraintes (3.11) et (3.14), il advient que pour

$y_1 \neq y_2$ il faut avoir :

$$\max(0, \alpha_2^{old} - \alpha_1^{old}) \leq \alpha_2 \leq \min(C, C - \alpha_1^{old} + \alpha_2^{old})$$

Une fois le coefficient est calculé, sa valeur est ramenée à la borne la plus proche de l'intervalle si celle-ci la dépassait.

Par la suite, ces deux bornes inférieure et supérieure seront respectivement dénommées L et H.

$$L = \begin{cases} \max(0, \alpha_2^{old} + \alpha_1^{old} - C) & si\ y_1 = y_2 \\ \max(0. \alpha_2^{old} - \alpha_1^{old}) & si\ y_1 \neq y_2 \end{cases}$$

$$H = \begin{cases} \min(C, \alpha_1^{old} + \alpha_2^{old}) & si\ y_1 = y_2 \\ \min(C, C - \alpha_1^{old} + \alpha_2^{old}) & si\ y_1 \neq y_2 \end{cases}$$

- Calcul d'un α_2 optimal

  Considérons maintenant la formulation de la fonction objective du problème dual :

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

  Celle-ci peut être réécrite pour faire apparaître α_1 et $\alpha_2$en considérant tous les autres α comme étant connus et fixés.

$$W(\alpha) = \alpha_1 + \alpha_2 - \frac{1}{2}\left(\alpha_1 \alpha_1 y_1 y_1 K(x_1, x_1) + \alpha_2 \alpha_2 y_2 y_2 K(x_2, x_2)\right)$$

$$+ \sum_{i=3}^{n} \alpha_i - \frac{1}{2}\left[2\alpha_1 y_1 \sum_{i=3}^{n} \alpha_i y_i K(x_1, x_i) + 2\alpha_2 y_2 \sum_{i=3}^{n} K(x_2, x_i)\right]$$

Cette écriture peut être simplifiée en posant quelques nouvelles variables :

$$K_{11} = K(x_1, x_1)$$
$$K_{12} = K(x_1, x_2)$$
$$K_{22} = K(x_2, x_2)$$

$$v_i = \sum_{j=3}^{n} \alpha_j y_j K(x_i, x_j)$$

Ce qui donne :

$$W(\alpha_1, \alpha_2) = \alpha_1 + \alpha_2 - \frac{1}{2}(\alpha_1^2 K_{11} + \alpha_2^2 K_{22}) - \alpha_1 \alpha_2 y_1 y_2 K_{12} - \alpha_1 y_1 v_1 - \alpha_2 y_2 v_2 + const$$

Etant donné que l'on cherche à optimiser la valeur de α_2 , il s'agit maintenant d'éliminer α_1 de la formulation de la fonction objective. Or d'après l'équation (3.11) et en posant

$$s = y_1 y_2 = \frac{y_1}{y_2} = \frac{y_2}{y_1}$$

Il est possible d'écrire que $\alpha_1 + s\alpha_2 = \alpha_1^{old} + s\alpha_2^{old} = \gamma$ (3.15)

Donc $\alpha_1 = \gamma - s\alpha_2$

Et par conséquent

$$W(\alpha_2) = \gamma - s\alpha_2 + \alpha_2 - \frac{1}{2}((\gamma - s\alpha_2)^2 K_{11} + \alpha_2^2 K_{22}) - \alpha_2 s(\gamma - s\alpha_2) K_{12} - (\gamma - s\alpha_2) y_1 v_1 - \alpha_2 y_2 v_2 + const \quad (3.16)$$

Une valeur optimale de α_2 est atteinte quand la dérivée de cette fonction objective s'annule.

$$\frac{\partial W(\alpha_2)}{\partial \alpha_2} = 0$$

$$\begin{aligned}\frac{\partial W(\alpha_2)}{\partial \alpha_2} &= -s + 1 - \frac{1}{2}[2(\gamma - s\alpha_2)(-s)K_{11} + 2\alpha_2 K_{22}] \\ &\quad -s(\gamma - s\alpha_2)K_{12} - \alpha_2 s(-s)K_{12} - (-s)y_1 v_1 \\ &\quad -y_2 v_2\end{aligned}$$

$$\begin{aligned}&= 1 - s + s\gamma K_{11} - \alpha_2 K_{11} - \alpha_2 K_{22} \\ &\quad -s\gamma K_{12} + 2\alpha_2 K_{12} + y_2 v_1 - y_2 v_2\end{aligned}$$

En égalisant cette dernière équation avec 0, on obtient une nouvelle équation permettant d'obtenir le nouveau coefficient α_2 optimal : α_2^*

$$\begin{aligned}\alpha_2^*(K_{11} + K_{22} - 2K_{12}) &= 1 - s + s\gamma(K_{11} - K_{12}) + y_2(v_1 - v_2) \\ &= y_2(y_2 - y_1 + y_1\gamma(K_{11} - K_{12}) + v_1 - v_2)\end{aligned}$$

En posant $k = K_{11} + K_{22} - 2K_{12}$ et en multipliant par y_2 , il est possible de simplifier cette écriture pour obtenir :

$$\alpha_2^* k y_2 = y_2 - y_1 + y_1\gamma(K_{11} - K_{12}) + v_1 - v_2$$

Il s'agit maintenant d'éliminer les variables simplificatrices γ et v_1, v_2. Pour ce faire, la formulation de $f(x_i)$ est réécrite en fonction des v_i :

$$f(x_i) = \sum_{j=1}^{n} \alpha_j y_j K(x_j, x_i) + b$$

$$= \sum_{j=3}^{n} \alpha_j y_j K(x_j, x_i) + \sum_{j=1}^{2} \alpha_j y_j K(x_j, x_i) + b$$

$$= v_i + \sum_{j=1}^{2} \alpha_j y_j K(x_j, x_i) + b$$

Donc $v_i = f(x_i) - \sum_{j=1}^{2} \alpha_j y_j K(x_j, x_i) - b$ (3.17)

Par conséquent

$$\begin{aligned}\alpha_2^* k y_2 &= y_2 - y_1 + (y_1\alpha_1 + y_2\alpha_2)(K_{11} - K_{12}) \\ &\quad + f(x_1) - \sum_{i=1}^{2} \alpha_i y_i K(x_i, x_1) - f(x_2) + \sum_{i=1}^{2} \alpha_i y_i K(x_i, x_2)\end{aligned}$$

$$= y_2 - y_1 + f(x_1) - f(x_2) + y_2\alpha_2 K_{11} - 2y_2\alpha_2 K_{12} + y_2\alpha_2 K_{22}$$

$$(f(x_1) - y_1) - (f(x_2) - y_2) + y_2\alpha_2 k$$

Ce qui donne finalement

$$\alpha_2^* = \alpha_2^{old} + \frac{(f(x_1) - y_1) - (f(x_2) - y_2)}{ky_2}$$

Ceci n'est évidemment valable que si k > 0. Dans le cas ou k=0, la division est impossible et il faut effectuer les calculs différemment.

En posant $E_i^{old} = f(x_i - y_i)$ la solution optimale pour le coefficient α_2 est

$$\alpha_2^* = \alpha_2^{old} + \frac{y_2(E_1^{old} - E_2^{old})}{k}$$

Valeur qui est ensuite tronquée pou satisfaire les contraintes de faisabilité :

$$Si\ k > 0, \qquad \alpha_2 = \begin{cases} \alpha_2^*\ si\ \ L < \alpha_2^* < H \\ H\ si\ \alpha_2^* \geq H \\ L\ si\ \alpha_2^* \leq L \end{cases}$$

Dans le cas ou k=0, cette valeur tend vers l'infini mais celle-ci devra toujours être bornée pour que $L \leq \alpha_2^* \leq H$. Etant donné qu'il s'agit d'une phase d'optimisation, la valeur qui sera retenue serra celle qui maximise la fonction objectif.

Reprenons le calcul de la fonction objectif (3.16) afin de la réécrire en fonction de $E_1^{old}, E_2^{old}, \alpha_2^{old}\ et\ \alpha_2$

$$\begin{aligned} W(\alpha_2) = {} & \gamma - s\alpha_2 + \alpha_2 - \frac{1}{2}((\gamma - s\alpha_2)^2 K_{11} + \alpha_2^2 K_{22}) \\ & -\alpha_2 s(\gamma - s\alpha_2)K_{12} - (\gamma - s\alpha_2)y_1 v_1 \\ & -\alpha_2 y_2 v_2 + const \\ = {} & \alpha_2 - s\alpha_2 + \gamma - \frac{\gamma^2 K_{11}}{2} + const \\ & -\frac{1}{2}(\alpha_2^2 K_{11} - 2\gamma s\alpha_2 K_{11} + \alpha_2^2 K_{22}) \\ & +\alpha_2^2 K_{12} - K_{12}\alpha_2 s\gamma - \gamma y_1 v_1 + s\alpha_2 y_1 v_1 - \alpha_2 y_2 v_2 \end{aligned}$$

Or $\gamma - \frac{\gamma^2 K_{11}}{2}$ est une constante qui sera donc incluse dans la variable constante (const) ,donc

$$\begin{aligned} W(\alpha_2) = {} & \frac{1}{2}k\alpha_2^2 + \alpha_2(1 - s + s\gamma K_{11} - s\gamma K_{12} + y_2 v_1 - y_2 v_2) \\ & + const \end{aligned}$$

Le calcul du second membre de $W(\alpha_2)$ est décomposable en plusieurs parties, d'après les équations (3.15) et (3.17) : $\gamma = \alpha_1^{old} + s\alpha_2^{old}$

$v_1 = f(x_1) - \alpha_1^{old} y_1 K_{11} - \alpha_2^{old} y_2 K_{21} - b^{old}$

$v_2 = f(x_2) - \alpha_1^{old} y_1 K_{12} - \alpha_2^{old} y_2 K_{22} - b^{old}$

Ce qui permet de calculer d'une part :

$$s\gamma K_{11} - s\gamma K_{12} = s\gamma(K_{11} - K_{12})$$
$$= s\left(\alpha_1^{old} + s\alpha_2^{old}\right)(K_{11} - K_{12})$$
$$= \left(s\alpha_1^{old} + \alpha_2^{old}\right)(K_{11} - K_{12})$$
$$= s\alpha_1^{old} K_{11} + \alpha_2^{old} K_{11} - s\alpha_1^{old} K_{12} - \alpha_2^{old} K_{12}$$

Et d'autre part :

$$y_2 v_1 - y_2 v_2 = y_2\left(f(x_1) - \alpha_1^{old} y_1 K_{11} - \alpha_2^{old} y_2 K_{21} - b^{old}\right)$$
$$-y_2\left(f(x_2) - \alpha_1^{old} y_1 K_{12} - \alpha_2^{old} y_2 K_{22} - b^{old}\right)$$
$$= y_2 f(x_1) - \alpha_1^{old} s K_{11} - \alpha_2^{old} K_{21} - y_2 b^{old}$$
$$-y_2 f(x_2) + \alpha_1^{old} s K_{12} + \alpha_2^{old} K_{22} + y_2 b^{old}$$
$$= s\alpha_1^{old} K_{12} + \alpha_2^{old} K_{22} - s\alpha_1^{old} K_{11} - \alpha_2^{old} K_{21} + y_2(f(x_1) - f(x_2))$$

Donc

$$1 - s + s\gamma K_{11} - s\gamma K_{12} + y_2 v_1 - y_2 v_2 = 1 - s + y_2\left(f(x_1) - f(x_2)\right)$$
$$+s\alpha_1^{old} K_{11} + \alpha_2^{old} K_{11} - s\alpha_1^{old} K_{12} - \alpha_2^{old} K_{12}\text{onst}$$
$$+s\alpha_1^{old} K_{12} + \alpha_2^{old} K_{22} - s\alpha_1^{old} K_{11} - \alpha_2^{old} K_{21}$$
$$= 1 - s + y_2\left(f(x_1) - f(x_2)\right)$$
$$+\alpha_2^{old} K_{11} + \alpha_2^{old} K_{22} - 2\alpha_2^{old} K_{12}$$

Or

$1 - s = y_1^2 - y_1 y_2 = y_2^2 - y_1 y_2$

donc

$$1 - s + s\gamma K_{11} - s\gamma K_{12} + y_2 v_1 - y_2 v_2 = y_2(y_2 - y_1 + f(x_1) - f(x_2)) + k\alpha_2^{old}$$
$$= y_2\left(E_1^{old} - E_2^{old}\right) + k\alpha_2^{old}$$

La fonction objectif est donc finalement

$$W\left(\alpha_2, E_1^{old}, E_2^{old}, \alpha_2^{old}\right) = \frac{1}{2}k\alpha_2 + \alpha_2\left(y_2\left(E_1^{old} - E_2^{old}\right) + k\alpha_2^{old}\right) + const$$

Comme annoncé antérieurement à ces calculs, cette équation va être utilisée pour deux valeurs extrêmes de α_2.

Ces deux valeurs porteront la dénomination de objectifL et objectifH

$$objectifL = W(\alpha_2 = L,\ E_1^{old}, E_2^{old}, \alpha_2^{old})$$

$$objectifH = W(\alpha_2 = H,\ E_1^{old}, E_2^{old}, \alpha_2^{old})$$

Soit $$objectifL = \frac{1}{2}kL + L\left(y_2\left(E_1^{old} - E_2^{old}\right) + k\alpha_2^{old}\right) + const$$

$$objectifH = \frac{1}{2}kH + H\left(y_2\left(E_1^{old} - E_2^{old}\right) + k\alpha_2^{old}\right) + const$$

La valeur retenue pour α_2^* est alors choisie afin de maximiser la valeur de la fonction objectif.

$$Si\ k = 0 \qquad \alpha_2 = \begin{cases} L & si \quad objectifL > objectifH \\ H & si \quad objectifL < objectifH \\ \alpha_2^{old} & sinon \end{cases}$$

- Calcul du coefficient d'ajustement b

Le lecteur aura certainement remarqué la mention d'un coefficient b^{old} au cours des calculs précédents. Celui-ci est utilisé dans la formulation de $f(x)$:

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

Soit $b = f(x) - \alpha_1 y_1 K(x_1, x) - \alpha_2 y_2 K(x_2, x) - \sum_{i=3}^{n} \alpha_i y_i K(x_i, x)$

Cette valeur dépendant des deux coefficients $\alpha_1\ et\ \alpha_2$ qui sont ajustés au cours d'une itération de SMO, il est nécessaire de la mettre à jour elle aussi pour prendre en compte les changements effectués dans les coefficients de lagrange.

Rappelons la définition de la mesure d'erreur appliquée aux anciennes et nouvelles valeurs des α

$$E_i^{old} = f^{old}(x_i) - y_i = \sum_{j=1}^{n} \alpha_j^{old} y_j K\left(x_j, x_i\right) + b^{old} - y_i$$

$$E_i = f(x_i) - y_i = \sum_{j=1}^{n} \alpha_j y_j K(x_j, x_i) + b - y_i$$

Ceci permet de calculer l'écart $\Delta E = E_i - E_i^{old}$

$$\Delta E = \sum_{j=1}^{n} \alpha_j y_j K(x_j, x) + b - y_i - \sum_{j=1}^{n} \alpha_j^{old} y_j K(x_j, x) - b^{old} + y_i$$

Notons de manière similaire $\Delta\alpha_1, \Delta\alpha_2 \; et \; \Delta b$ les changements respectifs des valeurs de $\alpha_1, \alpha_2 \; et \; b$.

$$\Delta E = \Delta\alpha_1 y_1 K(x_1, x) + \Delta\alpha_2 y_2 K(x_2, x) + \Delta b$$

Si pour une valeur donnée de l'ensemble d'apprentissage, on a $0 < \alpha_1 < C$, alors il s'agit d'un vecteur support. Les vecteurs supports sont les éléments délimitant les classes, ils sont donc par nature bien classés. Donc $0 < \alpha_1 < C \implies E_1 = 0$ dans ce cas il est possible de calculer :

$$\Delta b = -E_1^{old} - \Delta\alpha_1 y_1 K_{11} - \Delta\alpha_2 y_2 K_{21}$$

Il en va de même pour $\alpha_2 : 0 < \alpha_2 < C \implies E_2 = 0$ donc

$$\Delta b = -E_2^{old} - \Delta\alpha_1 y_1 K_{12} - \Delta\alpha_2 y_2 K_{22}$$

Dans le cas ou les deux coefficients α ne se conforment pas à ces conditions, ie (α_1, α_2) $\in \{0, C\}^2$, il est impossible de connaître la nouvelle erreur commise aussi bien pour l'un que pour l'autre. Pour néanmoins calculer la nouvelle valeur de b , la solution proposée dans la version originale de l'algorithme SMO et de calculer les deux valeurs en supposant que la nouvelle erreur est nulle et d'en prendre la moyenne.

$b_1 = b^{old} - E_1^{old} - \Delta\alpha_1 y_1 K_{11} - \Delta\alpha_2 y_2 K_{21}$

$b_2 = b^{old} - E_2^{old} - \Delta\alpha_1 y_1 K_{12} - \Delta\alpha_2 y_2 K_{22} b$

$b = \dfrac{b_1 + b_2}{2}$

Finalement b est ajusté de façon suivante :

$$b = \begin{cases} b_1 & si \;\; 0 < \alpha_1 < C \\ b_2 & si \;\; 0 < \alpha_2 < C \\ \dfrac{b_1 + b_2}{2} & si \;\; (\alpha_1, \alpha_2) \in \{0, C\}^2 \end{cases}$$

- **Heuristiques pour le choix du multiplicateur à optimiser [40]**

L'algorithme SMO permet d'optimiser deux multiplicateurs de Lagrange à chaque étape, avec un des multiplicateurs ayant violé les conditions de KKT avant cette étape. Par conséquent on maintiendra toujours le vecteur des multiplicateurs de Lagrange de l'étape précédente.

La fonction objective globale augmentera à chaque étape et l'algorithme convergera asymptotiquement. Afin d'accélérer la convergence, des heuristiques pour le choix des deux multiplicateurs de Lagrange à optimiser conjointement sont utilisées.

Deux heuristiques sont employées : une pour le premier multiplicateur de Lagrange et l'autre pour le second. La première heuristique détermine les exemples qui violent les conditions de KKT. Si un exemple viole les conditions de KKT, il est alors candidat à l'optimisation immédiate.

Une fois qu'un exemple violant les conditions de KKT est trouvé, un deuxième multiplicateur est choisi en utilisant la deuxième heuristique. Les deux multiplicateurs sont alors conjointement optimisés.

La deuxième heuristique essaye de maximiser $|E_1 - E_2|$.

- Si E_1 est positif, on choisit un exemple avec une erreur minimale E_2
- Si E_1 est négatif, on choisit un exemple avec l'erreur maximale E_2

On peut tomber dans des cas ou l'algorithme SMO ne permet pas d'avoir une progression positive en utilisant la deuxième heuristique. Par exemple, le premier et le deuxième exemple liés à α_1 et α_2 respectivement partagent des vecteurs d'entrée identiques.

Pour éviter ce problème, on cherche séquentiellement un α_2 qui viole les conditions de KKT et qui est compris entre $0\ et\ C$ $(0 < \alpha_i < C)$ en partant d'un indice choisi au hasard.

Si l'on ne trouve pas un α_2 qui satisfait ces deux conditions, on recommence la recherche séquentielle d'un$\alpha_2$qui viole les conditions de KKT seulement en commençant par un indice pris au hasard.

Après avoir trouvé les deux multiplicateurs à optimiser conjointement, on calcule leurs valeurs et on met à jour la valeur de la fonction objective.

L'algorithme SMO s'arrête lorsqu'il ne reste aucun multiplicateur de Lagrange qui viole les conditions de KKT.

3.4 Résultats expérimentaux

La phase de mise en œuvre d'un algorithme est l'ultime test permettant d'éprouver ses qualités. Elle peut révéler une certaine instabilité ou un autre défaut fatal ayant échappé à l'analyse théorique.

L'analyse des deux algorithmes hors lignes est illustrée à travers divers expériences faites sur différentes bases de données standards, de différentes tailles et de différentes dimensions. Chaque base est testée plusieurs fois avec de différents noyaux et en faisant varier les hyper paramètres. Trois noyaux ont été utilisés dans nos expériences:

- Polynomial
- RBF
- Linéaire

Nous allons donc faire des simulations en utilisant deux implémentations concernant les deux algorithmes. La première est connue sous le nom de LIBSM utilisant l'algorithme SMO et la seconde connue sous le nom SVMLIGHT utilisant l'algorithme de JOACHIMS.

Une première simulation va être faite sur des exemples d'apprentissages de deux classes, les bases de données choisies sont celles des benchmarks extraites de l'adresse [8].

Une deuxième simulation va être appliquée au domaine de la reconnaissance des chiffres manuscrits dont on utilise deux choix de bases de données : la base de données USPS et la base de données MNIST.

Une troisième simulation va être appliquée à trois paires de chiffres manuscrits les plus fréquemment confondues. L'objectif de cette simulation est de pouvoir comparer les résultats obtenus avec ceux des algorithmes incrémentaux qui seront présentés au chapitre 4.

3.4.1 Classification binaire : Application à des bases de données standards

- **Base de données : LEUKEMIA**

Cette base représente un diagnostic du cancer du sang. Elle est constituée de 38 exemples d'apprentissage (patients) et de 34 exemples pour le test , chaque exemple est décrit par 7129 attributs (ou caractéristiques).

Chaque patient (exemple) est classé dans l'une des deux classes suivantes :

Classe 1 : individu sain

Classe -1 : individu malade (réaction positive au test de la leucémie)

- **Résultats avec LIBSVM**

a) Noyau : linéaire

Valeur de C	# Itérations	nvst	nvsb	Perf %	Correct/Incorrect
0.00001	43	29	13	58.82	20/14
0.0001	54	29	9	70.58	24/10
1, 10 , 100	78	29	0	**82.35**	28/6

Tableau 3.1 : Résultats de Simulation (Noyau : linéaire, Base :LEUKEMIA)

b) Noyau : RBF

Paramètres	#itéra	nvst	nvsb	Perf %	Corr/Incor
C = 1 sigma=0.001	89	38	11	58.82	20/14
C = 1 sigma=0.0001	58	37	9	67.64	23/11
C = 10 sigma=0.00001	66	30	4	73.52	25/9
C = 100,sigma=0.00001	76	30	0	**82.35**	28/6

Tableau 3.2 : Résultats de Simulation (Noyau :RBF, Base :LEUKEMIA)

Remarque : nvst : représente le nombre total des Vecteurs supports dont nvsb représente le nombre des des vecteurs supports bornés(à l'intérieur de la marge).

- **Résultats avec SVMLIGHT**

Type noyau	Paramètres	#Itéra	nvst	nvsb	Perfor %	Corr/inc
Linéaire	C=1 (10,100)	33	29	0	**82.35**	28/6
RBF	C=10 $\sigma = 0.1$	20	38	0	58.82	20/14
RBF	C=100, $\sigma = 0.00001$	20	36	0	70.59	24/10

Tableau 3.3 : Résultats de Simulation (SVMLIGHT / BASE : LEUKEMIA)

- **Base de données ADULT**

Une base de données contenant des informations sur des personnes , la tâche consiste à prévoir si une personne gagne plus d'une certaine somme par an.

La base de données ADULT est de 14 caractéristiques dont 6 sont continues et 8 discrètes. Les deux types de caractéristiques sont convertis sous forme binaire, et donc s'augmentent au nombre 123.

- Base d'apprentissage est de 1605 exemples et la base de test est de 30956.
- Cardinalité des caractéristiques est de 123

- **Résultats avec LIBSVM**

type noyau	Paramètres	# Itér	nvst	Nvsb	Perfor%	Corr/Incorr	t-cpu
Linéaire	C=10 C=1 C=0.1	142106 9408 1281	569 588 642	490 522 595	83.76 83.81 84.31	25930/5026 25947/5009 26099/4857	2.32 2.10 2.12
RBF	C=10 $\sigma = 0.25$ C=1 $\sigma = 0.25$ C=1 $\sigma = 0.10$	2886 1419 948	1024 1009 728	31 402 511	82.31 83.49 84.20	25481/5475 25846/5110 26066/4890	4.11 4.05 2.80

Tableau 3.4 : Résultats de Simulation (LIBSVM / BASE : ADULT)

- **Résultats avec SVMLIGHT**

type noyau	Paramètres	# Itér	nvst	nvsb	Perfor %	Correct/ Incorrect	t-cpu
Linéaire	C=10 C=1 C=0.1	59140 3435 500	569 589 642	490 523 595	83.77 83.84 84.33	25933/5023 25953/5003 26104/4852	6.06 0.49 5.21
RBF	C=10 $\sigma = 0.25$ C=1 σ=0.25 C=1 $\sigma = 0.10$	942 389 364	1024 1012 730	31 401 509	82.31 83.50 84.20	25481/5475 25847/5109 26065/4891	0.65 0.41 0.36

Tableau 3.5 : Résultats de Simulation (SVMLIGHT / BASE : ADULT)

- **Base de données W8A (web spam dataset) :**

Cette base représente la description de deux types de mails (spam ou non spam). elle est constituée de 49749 exemples pour l'apprentissage et de 14951 exemples pour le test. Chaque exemple est décrit par 300 caractéristiques.

Chaque mail (exemple) est classé dans l'une des deux classes :

Classe 1 : non spam mail,

Classe -1 : spam mail

- **Résultats avec LIBSVM**

type noyau	Paramètres	#Itér	Nvst	Nvsb	Perfor %	Correct/ Incorrect	t-cpu
Linéaire	C=10 C=1 C=0.1	696140 65155 12392	1611 1669 2159	1299 1437 1947	98.67 98.67 98.56	14753/198 14753/198 14736/215	104.39 42.02 18.26
RBF	C=1 σ = 0.25 C=1 $\sigma = 0.10$ C=10 $\sigma = 0.10$	20381 8645 14415	13363 4596 4660	1228 13336 53	99.15 99.01 99.32	14824/129 14804/149 14850/101	400.86 136.29 138.87
Poly	C=1 d=2 C=1 d=3 C=1 d=1	2044 1555 2506	3183 3030 2997	2788 2938 2891	96.96 96.96 97.18	14497/454 14497/454 14530/420	29.85 19.82 21.71

Tableau 3.6 : Résultats de Simulation (LIBSVM / BASE : W8A)

- **Résultats avec SVMLIGHT**

type noyau	Paramètres	#Itér	nvst	nvsb	Perfor%	Correct/ incorrect	t-cpu(s)
Linéaire	C=10 C=1 C=0.1	448837 35569 6995	1636 1702 2197	1293 1430 1937	98.68 **98.68** 98.56	14753/198 14753/198 14735/216	65.82 5.72 3.07
RBF	C=1 $\sigma = 0.25$ C=1 $\sigma = 0.10$ C=10 $\sigma = 0.10$	5397 4675 9468	10132 4729 4785	1187 1314 635	99.13 97.52 **99.32**	14821/130 14804/147 14850/101	303.80 120.21 187.72
Poly	C=1 d=2 C=1 d=3 C=1 d=1	51275 259718 33762	2470 2627 1744	512 479 1426	**99.44** 99.45 98.68	14868/83 14869/82 14753/198	344.89 1581.35 191.53

Tableau 3.7 : Résultats de Simulation (SVMLIGHT / BASE : W8A)

- **Interprétation des résultats**

Paramètre C : c'est la constante qui représente un compromis entre la maximisation de la marge et la minimisation des erreurs, comme déjà vue théoriquement au chapitre 2.

Si C prend une grande valeur lors de la phase d'apprentissage, on tient compte beaucoup plus à la minimisation des erreurs lors de l'apprentissage que de la maximisation de la marge. Donc on aura une petite marge et l'erreur empirique est minimale, mais l'erreur en généralisation est grande (phénomène de l'apprentissage par cœur).

Dans le cas ou C prend une valeur très petite, c'est-à-dire on tolère des erreurs et on tient compte de la maximisation de la marge, dans ce cas les exemples des deux classes ne peuvent pas être séparés par une frontière.

Exemple : au niveau de la base de données LEUKEMIA, cas du noyau linéaire tableau (3.1)

On remarque à travers les deux premières lignes du tableau (cas C=0.00001 et C=0.0001), que nous avons donc trop toléré aux erreurs. Ce qui induit donc une mauvaise performance pour les deux cas 58.82% pour le premier cas et 70.58% pour le deuxième cas.

Pour la troisième ligne, on remarque que pour C=1, C=10, C=100 la performance est la même 82.35%, même en augmentant encore la valeur de C , on trouve toujours ce résultat .

- En comparant les résultats des deux algorithmes LIBSVM et SVMLIGHT appliqués à la base de donnés 'LEUKEMIA', on peut dire que pour le noyau linéaire et les valeurs du paramètre C (1, 10, 100), les deux algorithmes donnent les mêmes résultats. la performance en généralisation est 82.35 %, le nombre de vecteurs supports est 29 mais le nombre d'itérations n'est pas le même, c'est logique puisqu'ils n'utilisent pas les mêmes heuristiques.

- Mais pour le noyau RBF on trouve que l'algorithme LIBSVM a donné une meilleure performance en généralisation qui est 82.35% dans le cas ou C=100 et $\sigma = 0.00001$ mais pour ces mêmes paramètres l'algorithme SVMLIGHT donne une performance de 70.59%.
- En comparant les résultats des deux algorithmes appliqués à la base de données 'ADULT', on remarque pour les mêmes type de noyau et pour les mêmes valeurs des paramètres que ce soit la constante de régularisation C ou le paramètre du noyau RBF σ , les algorithmes donnent des résultats très proches. Cette même remarque s'applique pour la base de données W8a.

3.4.2 Classification multi classe : Application aux bases de données USPS et MNIST

Dans cette section, nous évaluons la classification multi-classe en utilisant l'approche un Contre un pour les différents noyaux et différents paramètres des noyaux.

- **Base de données USPS :**

La base de données fournie par les services postaux américains contient près de 9000 images représentant les chiffres de 0 à 9 extraits de codes postaux. Les données sont réparties entre des données d'apprentissage et des données de test. Chaque image est de dimension 16×16 pixels en niveaux de gris. L'erreur humaine en reconnaissance de la base de test est de 2.5%.

- **Résultats de Simulation avec LIBSVM**

Noyau RBF				Noyau Polynomial			
C	σ	Perf %	Temps	C	d	Perf %	Temps
10	2	26.74	81.19	1	2	93.72	9.84
10	1	29.04	81.72	10	2	94.86	4.96
10	0.5	34.72	75.33	100	2	94.91	4.63
100	0.1	86.34	53.49	1	3	93.77	11.84
0.01	0.01	81.41	48.84	10	3	95.11	6.14
0.1	0.01	92.62	17.45	100	3	**95.41**	5.64
1	0.01	94.96	7.72	0.1	2	90.13	26.72
10	**0.01**	**95.46**	7.44	0.01	2	46.83	58.79
100	0.01	95.36	7.76	0.001	2	**17.88**	64.02

Tableau 3.8 : Résultats de simulation multi classe (LIBSVM / BASE : USPS)

- **Résultats de Simulation avec SVMLIGHT**

Noyau RBF				Noyau Polynomial			
C	σ	Perf %	Temps (s)	C	D	Perf %	Temps
10	0.01	58.80	6203.96	10	2	95.12	325.12
10	0.1	91.18	375.98	0.01	2	45.12	224.22
10	1	94.82	386.26	10	3	94.75	175.22
0.1	0.01	93.12	274.23	0.001	2	18.12	125.12

Tableau 3.9 : Résultats de simulation multi classe (SVMLIGHT / BASE : USPS)

- **Base de données MNIST :**

MNIST est une base de données de reconnaissance d'écriture manuscrite contenant les chiffres de 0 à 9. La partie utilisée ici contient 60 000 images pour l'apprentissage et 10 000 pour le test. Les images sont de dimension 28×28 en niveaux de gris. 56 images sont a priori non identifiables par un humain.

- **Simulation avec LIBSVM**

Noyau Polynomial				Noyau RBF			
C	D	Perf %	Temps(s)	C	σ	Perf %	Temps(s)
10	2	98.05	461.52	10	0.01	98.25	8175.23
10	3	97.91	457.58	10	0.1	87.01	5725.23
10	4	97.37	486.39	10	1	47.85	4852.10

Tableau 3.10 : Résultats de simulation multi classe (LIBSVM / BASE MNIST)

- **Simulation avec SVMLIGHT**

Noyau Polynomial				Noyau RBF			
C	D	Perf %	Temps(s)	C	σ	Perf %	Temps(s)
10	2	95.15	875.20	10	0.01	62.15	12745.58
10	3	94.12	814.02	10	0.1	90.25	8751.89
10	4	92.01	975.23	10	1	94.25	8912.45

Tableau 3.11 : Résultats de simulation multi classe (SVMLIGHT / BASE MNIST)

- **Interprétation des résultats**

A partir des différents tests effectués voire tableau 3.8, on trouve que les meilleurs paramètres pour le noyau RBF Sont C=10 et σ = 0.01 (performance =95.46%) et pour le noyau polynomial les meilleurs paramètres sont C=100, d =3 (performance = 95.41 %).

On remarque qu'en diminuant le paramètre C, la performance en généralisation se dégrade, Exemple (Noyau polynomial, C = 0.01 d=2), la performance est de 46.83% et pour C=0.001, avec le même noyau et le même degré, la performance est de 17.88%), donc ce qui est prouvé en théorie.

En examinant les résultats fournis par le noyau RBF, On trouve que la meilleure performance en généralisation est atteinte pour le couple de paramètre (C= 10,σ =0.01), elle est de 95.46%. On voit que pour la même valeur de σ et en minimisant C, la performance diminue (C=1,σ =0.01), la performance est de 94.96%, et pour le couple (C=0.1, σ =0.01), la performance est de 92.62%.

En examinant les résultats pour le noyau RBF, en augmentant le paramètre σ, ce qui accroit la similarité entre les exemples. La mesure de similarité ne peut plus distinguer les

exemples de classes opposées, donc l'algorithme est incapable de déduire une surface de décision traduisant la frontière entre les classes. On remarque donc que la performance se dégrade, on voit pour le couple (C=10, σ =1), la performance est de 29.04%, et pour le couple (C=10, σ =1), la performance est de 26.74%.

On remarque que pour les mêmes valeurs des deux hyper paramètres (C=10, σ =1), les deux algorithmes donnent des résultats totalement différents. Pour LIBSVM, la performance est de 29.04%(voir tableau 3.8), par contre pour SVMLIGHT elle est de 95.07%(voir tableau 3.9). Donc on peut conclure que les valeurs optimales des hyper paramètres dépendent des méthodes utilisés.

En comparant les temps d'exécutions, l'algorithme SMO est plus rapide que celui de SVMLIGHT.

3.4.3 Application aux paires de chiffres manuscrits.

Dans cette expérimentation, nous voulons expérimenter la performance des SVMs pour séparer deux classes les plus fréquemment confondues. Ces classes correspondant aux paires (9,4), (5,6) et (7,1). Nous avons effectué trois opérations de filtrage à partir de la base de données USPS. Ensuite dans le vecteur des étiquettes résultant nous avons remplacé le label 9 par 1 et le label 4 par -1 pour la paire de classe(9,4) pour pouvoir appliquer une classification binaire. En faisant la même chose pour les deux autres paires.

Classe 9 et 4 :

- Base d'apprentissage est de 1200 exemples
- Base de Test est de 332 exemples

Classe 5 et 6 :

- Base d'apprentissage est de 1208 exemples
- Base de Test est de 360 exemples

Classe 1 et 7 :

- Base d'apprentissage est de 1858 exemples
- Base de Test est de 529 exemples

3.4.3.1 Résultats de Simulation avec LIBSVM

Cas 1 : Classes 4 et 9

type noyau	Paramètres	nvst	nvsb	Perfor %	t-cpu(s)
Poly	C=10 d=1 C=10 d=2 C=10 d=3	98 118 151	63 52 61	96.68 97.59 97.89	0.12 0.11 0.12
RBF	C=10 $\sigma = 0.01$ C=10 $\sigma = 0.1$ C=10 $\sigma = 1$	127 932 1200	5 0 0	97.59 89.75 50 .00	0.10 0.81 1.10

Tableau 3.12 : Résultats de Simulation (LIBSVM : Classes 4 et 9)

Cas 2 : Classes 5 et 6

type noyau	Paramètres	Nvst	nvsb	Perfor%	t-cpu(s)
Poly	C=10 d=1 C=10 d=2 C=10 d=3	80 92 124	45 23 15	98.33 98.33 98.61	0.09 0.09 0.10
RBF	C=10 $\sigma = 0.01$ C=10 $\sigma = 0.1$ C=10 $\sigma = 1$	121 1031 1208	1 0 0	98.61 91.94 55.55	0.093 0.82 1.07

Tableau 3.13 : Résultats de Simulation (LIBSVM : Classes 5 et 6)

Cas 3 : Classes 1 et 7

type noyau	Paramètres	nvst	nvsb	Perfor %	t-cpu(s)
Poly	C=10 d=1 C=10 d=2 C=10 d=3	79 87 112	44 27 30	99.62 99.24 99.43	0.12 0.12 0.17
RBF	C=10 $\sigma = 0.01$ C=10 $\sigma = 0.1$ C=10 $\sigma = 1$	98 1125 1858	0 0 0	99.62 95.84 67.86	0.17 1.85 2.68

Tableau 3.14 : Résultats de Simulation (LIBSVM : Classes 1 et 7)

3.4.3.2 Résultats de Simulation avec SVMLIGHT

Classe 4 et 9 :

type noyau	Paramètres	nvst	nvsb	Perfor %	t-cpu(s)
Poly	C=10 d=1 C=10 d=2 C=10 d=3	50 90 123	0 0 0	99.96 99.97 99.5	0.04 0.45 0.49
RBF	C=10 $\sigma = 0.01$ C=10 $\sigma = 0.1$ C=10 $\sigma = 1$	128 932 1200	5 0 0	99.98 99.90 99.50	0.45 1.07 1.65

Tableau 3.15 : Résultats de Simulation (SVMLIGHT : Classes 4 et 9)

Classe 5 et 6 :

type noyau	Paramètres	Nvst	nvsb	Perfor %	t-cpu(s)
Poly	C=10 d=1 C=10 d=2 C=10 d=3	73 98 120	0 0 0	99.98 99.98 99.99	0.98 0.51 0.51
RBF	C=10 $\sigma = 0.01$ C=10 $\sigma = 0.1$ C=10 $\sigma = 1$	121 1030 1208	1 0 0	99.97 99.97 99.50	0.42 1.18 1.76

Tableau 3.16 : Résultats de Simulation (SVMLIGHT : Classes 5 et 6)

Classe 1 et 7 :

type noyau	Paramètres	nvst	nvsb	Perfor%	t-cpu(s)
Poly	C=10 d=1 C=10 d=2 C=10 d=3	41 68 92	0 0 0	99.4 99.4 99.4	0.85 0.76 0.82
RBF	C=10 $\sigma = 0.01$ C=10 $\sigma = 0.1$ C=10 $\sigma = 1$	98 1126 1858	0 0 0	99.4 99.4 99.5	0.79 2.41 3.58

Tableau 3.17 : Résultats de Simulation (SVMLIGHT : Classes 1 et 7

3.4.3.3 Interprétations des résultats :

Commençons par le noyau polynomial, nous voyons que les taux de reconnaissances (ou performance en généralisation) sont presque similaires pour les trois paires de classes, mais on doit fixer notre attention sur le nombre des vecteurs supports qui augmente en fonction du degré du polynôme. Vu que la fonction de décision du SVM obtenue dépend uniquement des vecteurs supports et non pas de tous les vecteurs de la base d'apprentissage, donc on doit favoriser une solution qui donne moins de vecteurs supports.

A titre d'exemple, à partir du tableau (3.12), nous observons deux cas, le premier est de degré 2, sa performance égale à 97.59 %, et le second est de degré 3 et sa performance est de 97.89% , On préfère le premier cas (degré 2) car il a moins de vecteurs supports. Donc on choisi la solution qui généralise mieux et en même temps de moindre complexité calculatoire.

Passons maintenant au noyau RBF, nous observons la même remarque déjà mentionnée dans les simulations précédentes, c'est-à-dire en augmentant le paramètre σ **,** la performance en généralisation diminue.

En comparant les deux algorithmes LIBSVM et SVMLIGHT, on peut dire que les résultats sont très proches.

En examinant la figure (3.1), l'algorithme Svmlight est plus performant en taux de reconnaissance que l'algorithme Libsvm, seulement dans le cas du noyau RBF. Par contre dans le cas du noyau polynomial, les deux algorithmes ont de bons résultats et pour toutes les paires de classes (voire les tableaux 3.12 à 3.17)

En examinant la figure (3.2), qui illustre l'influence du paramètre de régularisation C sur le taux de reconnaissance appliquée à l'algorithme LIBSVM. En diminuant la valeur de C, le taux diminue.

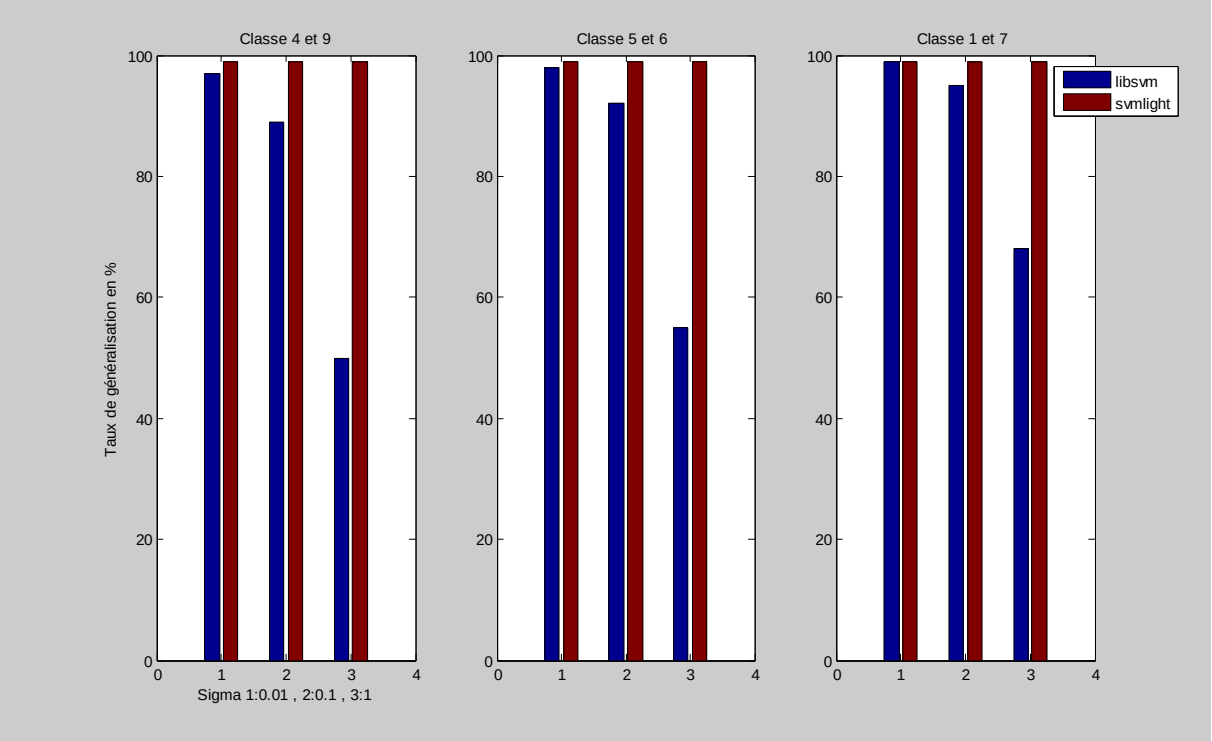


Figure 3.1 : Taux de reconnaissance en fonction de sigma(Noyau RBF),C=10

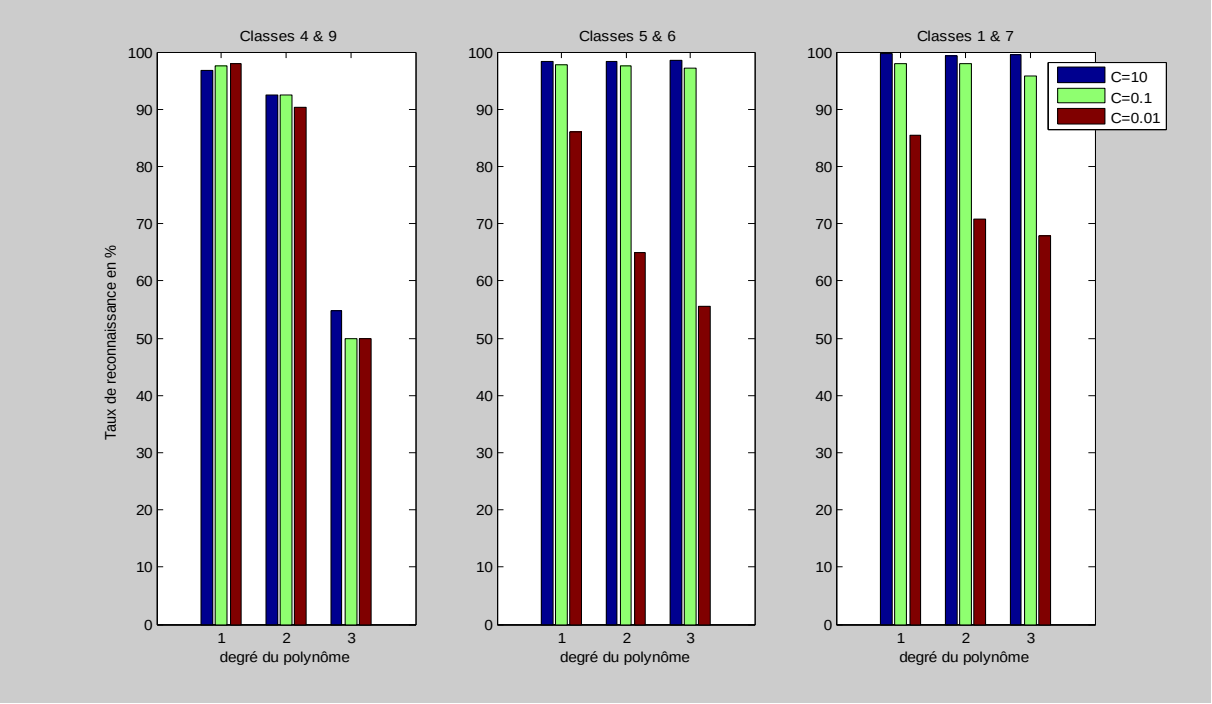


Figure 3.2 : Taux de reconnaissance en fonction du degré du noyau polynomial, Selon 3 valeurs du paramètre C.

3.5 Conclusion

Ce chapitre met en avant la prudence nécessaire à la comparaison de méthodes de résolution. En effet l'ordre de grandeur des hyper paramètres est un facteur influant sur la rapidité de résolution et sur sa performance en généralisation.

Nous pouvons dire que les deux algorithmes hors ligne ont pu montrer leurs performances en un temps d'exécution négligeable pour les données de cardinalités moyennes, mais dés que la cardinalité augmente ils montrent leurs limites du point de vu temps d'exécution. Ce qui est très logique puisque la résolution du problème d'optimisation se fait sur tous les exemples de la base d'apprentissage.

Enfin, nous pouvons dire que tous les fondements théoriques sur lesquels les SVMs s'appuient ont été prouvés à travers cette expérimentation.

Chapitre 4

Algorithmes Incrémentaux

Des Machines à Vecteurs Supports

4.1 Introduction

L'apprentissage hors ligne correspond à l'apprentissage d'un modèle sur un jeu de données représentatif du problème et disponible au moment de l'apprentissage. Ce type d'apprentissage est réalisable sur des volumes de taille faible à moyenne (jusqu'à quelques Go). Au-delà le temps d'accès et de lecture des données devient prohibitif et il devient difficile de réaliser un apprentissage rapide (qui ne prend pas des heures ou des jours). Ce type d'algorithme montre ses limites dans le cas où

- les données ne sont pas entièrement chargeables en mémoire ou arrivent de manière continue.
- la complexité calculatoire de l'algorithme d'apprentissage est supérieure à une complexité dite quasi-linéaire.

L'apprentissage incrémental est bien souvent une alternative intéressante face à ce genre de problèmes. Le principe des algorithmes incrémentaux est de ne charger qu'un petit bloc de données en mémoire à la fois, de construire un modèle partiel et de le mettre à jour en chargeant consécutivement des blocs de données. Cela permet de traiter de très grandes quantités de données sans difficulté de mémoire sur une machine standard. Un algorithme incrémental peut donner des résultats approximatifs ou identiques à ceux obtenus par un algorithme d'apprentissage chargeant une seule fois l'ensemble entier des données.

Dans ce chapitre, on va présenter le principe de l'apprentissage incrémental, ensuite on va donner un état de l'art des algorithmes incrémentaux dédiés aux machines à vecteurs supports, ensuite on va détailler le principe d'un algorithme incrémental de cauwenbergh et poggio suivi de la présentation du **nouvel algorithme incrémental proposé** en détaillant son principe de fonctionnement.

Finalement les deux algorithmes seront testés sur une base de données artificielle en interprétant les résultats à l'aide des figures sur plusieurs types de noyaux et pour différentes valeurs d'hyper paramètre, ensuite des résultats expérimentaux sont établis en testant les deux algorithmes sur des paires de chiffres manuscrits les plus confondus.

4.2 Apprentissage hors-ligne et Apprentissage en-ligne

Un système de reconnaissance peut être utilisé soit de manière en-ligne soit de manière hors ligne [17]. On parle bien ici de la manière dont est utilisé le système et non pas de la manière dont est réalisé l'apprentissage comme on le verra ensuite.

- **Apprentissage hors-ligne**

Le mode de fonctionnement hors-ligne sépare les phases d'apprentissage et d'utilisation. Dans un premier temps, le système est entrainé de manière statique ou incrémentale à partir d'un ensemble d'entrainement. Ensuite, dans un deuxième temps, le système préalablement appris est utilisé.

- **Apprentissage en-ligne**

Le mode de fonctionnement en-ligne ne sépare pas l'apprentissage et l'utilisation du système, les deux tâches sont effectuées de manière conjointe. Tout au long de son utilisation, le système continue d'apprendre dès qu'une nouvelle donnée est disponible afin d'améliorer ses performances [93].

L'avantage des systèmes en-ligne par rapport aux systèmes hors-ligne est qu'ils vont pouvoir s'ajuster très précisément à l'utilisateur final en continuant à apprendre tout au long de leur utilisation. L'ensemble d'apprentissage initial n'a plus besoin d'être aussi conséquent et diversifié puisque le système s'adaptera pendant son utilisation [93].

L'apprentissage d'un système fonctionnant en-ligne est réalisé le plus souvent incrémentalement, même s'il pourrait également être fait de manière statique en reconstruisant tout le système (en stockant toutes les données).

4.3 Apprentissage statique et apprentissage incrémental

L'entrainement d'un système d'apprentissage peut se faire de deux manières : de manière statique ou de manière incrémentale. Nous allons voir les principales différences entre ces deux approches.

- **Apprentissage statique**

En apprentissage statique (*batch learning*), le jeu de données d'entrainement est prédéfini et disponible lors de l'entrainement du système. L'algorithme d'apprentissage utilise l'ensemble du jeu de données pour minimiser le taux d'erreur.

- **Apprentissage incrémental**

En apprentissage incrémental, le jeu de données n'est pas forcément disponible dès le début de l'entrainement. Les données sont introduites au fur et à mesure et le système doit être capable de se modifier et d'ajuster ses paramètres après l'observation de chaque exemple pour apprendre à partir de celui-ci ; mais néanmoins sans oublier la connaissance acquise à partir des exemples précédents.

Ce mode d'apprentissage est utilisé soit lorsque le jeu de données est trop grand pour être utilisé en une seule fois, soit lorsque l'ensemble d'apprentissage n'est pas disponible dans son intégralité et que les données d'entrainement arrivent de manière incrémentale.

L'apprentissage incrémental peut facilement être réalisé en-ligne et permet d'utiliser le système pendant son apprentissage, ou formulé autrement, d'améliorer le système tout au long de son utilisation.

4.4 Système adaptatif et système évolutif

Les systèmes d'apprentissage incrémental sont donc dynamiques car ils apprennent au fur et à mesure que les données arrivent, contrairement aux systèmes d'apprentissage classique, dits statiques. Néanmoins, ces systèmes incrémentaux peuvent encore être séparés en deux catégories : les Systèmes adaptatifs et les systèmes évolutifs [17].

- **Système adaptatif**

On appelle systèmes adaptatifs les systèmes d'apprentissage incrémental qui vont apprendre de manière incrémentale leurs paramètres mais dont la structure est fixe. Cet apprentissage des paramètres peut être considéré comme un algorithme « d'adaptation »[93]. La structure de ces systèmes est initialisée au départ et reste ensuite inchangée pendant l'apprentissage.

- **Système évolutif**

Les systèmes évolutifs sont une sous-catégorie des systèmes incrémentaux capables de changer leur structure. Ils vont non seulement apprendre leurs paramètres de manière incrémentale, mais également modifier leur structure, comme par exemple lors de l'apparition d'une nouvelle classe. Il est souhaitable qu'un système évolutif n'ait pas de paramètres dépendants du problème qui seraient difficilement ajustables de manière incrémentale [93].

Un système évolutif est donc bien plus flexible qu'un système adaptatif puisqu'il permet, en cours d'utilisation, d'ajouter des classes supplémentaires lorsque cela s'avère nécessaire.

4.5 Mémoire des données et mémoire des concepts

Un système d'apprentissage a besoin de mémoriser l'information apprise. Pour cela il peut soit mémoriser les données, soit mémoriser les concepts extraits des données, ou encore combiner les deux [18].

- **Mémoire des données**

Mémoriser toutes les données est une méthode simple qui permet de mettre à jour l'ensemble d'apprentissage facilement. Cette méthode est par exemple utilisée par les classificateurs de type plus proches voisins. Elle peut également être utilisée pour reconstruire tout un système à l'arrivée de nouvelles données.

Cependant, cette méthode possède plusieurs inconvénients. Le nombre de données à mémoriser augmente tout au long de l'utilisation du système et cette méthode va nécessiter beaucoup d'espace mémoire. De plus, si l'on reconstruit tout le système, son apprentissage va être de plus en plus long et complexe au fur et à mesure que la taille du jeu de données d'entrainement augmente.

Une approche dérivée limitant ces inconvénients consiste à sélectionner et maintenir un sous ensemble des données d'apprentissage sur lequel reconstruire le système. Cette méthode est appelée mémoire partielle des données, à opposer à la mémoire complète des données.

- **Mémoire des concepts**

Une autre méthode consiste non pas à mémoriser les données, mais plutôt à mémoriser les concepts, c'est-à-dire la connaissance extraite de ces données. Cette méthode est par exemple utilisée par les classificateurs de type réseau connexionnistes. La connaissance, le modèle, doit alors être mis à jour à l'arrivée de nouveaux exemples de manière à prendre en compte ces nouvelles informations mais sans pour autant oublier celles issues des exemples précédents.

4.6 Etat de l'art sur l'apprentissage incrémental des SVM

Des versions incrémentales des SVM ont été proposées, parmi celles-ci :

- (Domeniconi et Gunopulos, 2001) propose un découpage des données en partitions et quatre techniques différentes pour réaliser l'apprentissage incrémental [27] :

- **ED Error Driven technique** : Lors de l'arrivée de nouveaux exemples, ceux qui sont mal classifiés sont conservés pour modifier le SVM.
- **FP - Fixed Partition technique** : Une approche consiste à apprendre les nouvelles données en oubliant toutes celles précédemment rencontrées à l'exception des vecteurs supports (Syed et al., 1999). L'idée sous jacente est que les vecteurs supports sont des points qui résument bien l'ensemble des données et qu'ils peuvent donc permettre à eux seuls de réaliser un apprentissage de bonne qualité.
- **EM - Exceeding-Margin technique** : Lors de l'arrivée de nouveaux exemples, ceux qui se situent dans la zone de marge de l'hyperplan sont conservés. Lorsqu'un nombre assez important de ces exemples est collecté le SVM est mis à jour.
- **EM+E - Exceeding-margin+errors technique** : Utilisation de "ED" et "EM", les exemples qui se situent dans la zone de marge et ceux qui sont mal classifiés sont conservés.

- L'algorithme **Kernel Adatron** (Friess et al , 1998) [66] permet d'approximer très rapidement la solution de l'apprentissage classique par vecteurs supports et correspond à un algorithme d'optimisation qui approche le vecteur directeur de l'hyperplan composante par composante.
  Il a été utilisé avec succès par ses auteurs pour l'adaptation dynamique des paramètres du noyau des machines, en réalisant une phase de sélection de modèle au cours de l'apprentissage.

- (Fung et Mangasarian, 2002) [10] propose un PSVM - **Proximal SVM**, qui au lieu de voir une frontière comme étant un plan, la voit comme plusieurs plans (un espace) à proximité du plan de frontière.
  Les exemples supportant les vecteurs supports ainsi qu'un ensemble de points situés dans l'espace proche autour de l'hyperplan frontière sont conservés. En faisant évoluer cet ensemble, on enlève certains exemples trop anciens et on rajoute les nouveaux exemples ce qui permet de rendre le SVM incrémental.

- (Cauwenbergh et Poggio , 2001) [9] propose une méthode pour résoudre de manière incrémentale le problème d'optimisation global afin d'en trouver une solution exacte. l'aspect inversible de cette méthode permet de réaliser un désapprentissage «décremental» qui permet de calculer efficacement des estimations (leave one out) des capacités de généralisation de la machine apprise.

- (Liva Ralaivola 2002) [53] propose un algorithme d'apprentissage en ligne incrémental pour les SVM à noyau gaussien fondé sur la méthode SMO de Platt , et opère un réapprentissage local sur un sous ensemble des exemples disponibles voisins des nouvelles données. Le voisinage de réapprentissage est déterminé automatiquement par la minimisation d'une fonction de coût instantanée traduisant

l'objectif de l'algorithme est de réaliser un compromis entre de bonnes performances en généralisation et une complexité en temps de calcul réduite.

- (Bordes et Bottou, 2005; Bordes et al., 2005) [14] propose l'algorithme LASVM. Il s'appuie sur une sélection active des points à intégrer dans la solution et donne des résultats très satisfaisants pour une utilisation en ligne. L'apprentissage peut être interrompu à tout moment, avec une étape éventuelle de finalisation (qui correspond à éliminer les vecteurs supports devenus obsolètes). Cette étape de finalisation faite régulièrement au cours de l'apprentissage en ligne permet de rester proche des solutions optimales. Du côté de la complexité calculatoire il n'y a qu'une résolution analytique à chaque étape. Du côté mémoire, l'algorithme peut être paramétré et c'est alors une question de compromis entre temps de calcul et espace mémoire.

Le principal inconvénient des SVM incrémentales est leur difficulté à gérer l'ajout de nouvelles classes. Une classe récemment ajoutée ne possèdera que peu d'individus, et en maximisant la marge, une SVM aura tendance à isoler ces quelques points sans vraiment généraliser la représentation de cette nouvelle classe.

4.7 Algorithme de Cauwenberghs [9]

4.7.1 Principe de l'algorithme

C'est un algorithme en ligne, récursif pour apprendre un SVM, un nouveau vecteur (ou exemple) est ajouté à la base d'apprentissage, si ce point est bien classé par la solution courante alors aucun changement ne sera effectué, dans le cas contraire une mise à jour de la solution courante est nécessaire en faisant des corrections sur les multiplicateurs de lagrange tout en respectant les conditions d'optimalité KKT.

Les exemples d'apprentissage sont partitionnés dans trois groupe, le groupe des points les bien classés, le groupe des vecteurs supports et le groupe des vecteurs erreurs.

Lors de la mise à jour de la solution, des exemples parmi ces trois groupes peuvent changer d'état. après la phase de l'apprentissage, une procédure décrémentale est exécutée, c'est la procédure standard LOO (leave one out) .

4.7.2 Conditions d'optimalité KKT

Dans la classification d'un SVM, la fonction séparatrice se réduit à une combinaison linéaire de la fonction noyau (kernel) sur les données d'apprentissage

$$f(x) = \sum_i \alpha_i y_i k(x_i, x) + b$$

Avec x_i les points d'apprentissage et y_i leurs classes correspondantes.

Dans la formulation duale du problème d'apprentissage, les coefficients α_i sont minimisés par la fonction objective quadratique et convexe sous les contraintes:

$$\min_{0<\alpha_i<C} : w = \frac{1}{2}\sum_{i,j} \alpha_i Q_{ij} \alpha_j - \sum_i \alpha_i + b \sum_i y_i \alpha_i \qquad (4.1)$$

Avec Q c'est la matrice semi définie positive de la fonction noyau : $Q_{ij} = y_i y_j K(x_i, x_j)$

Les conditions du premier ordre de la fonction W se réduisent aux conditions de KKT suivantes :

$$g_i = \frac{\partial w}{\partial \alpha_i} = \sum_j Q_{ij}\alpha_j + y_i b - 1 \;=\; y_i f(x_i) - 1 \quad \begin{cases} > 0 & \alpha_i = 0 \\ = 0 & 0 \le \alpha_i \le C \\ < & 0 \;\; \alpha_i = C \end{cases} \qquad (4.2)$$

$$\frac{\partial w}{\partial b} = \sum_j y_j \alpha_j = 0 \qquad (4.3)$$

Les données d'apprentissage sont partitionnées dans trois groupes :

- Le groupe S contenant les vecteurs supports ($0 < \alpha_i < C$).
- Le groupe E contenant les vecteurs erreurs ($\alpha_i = C$).
- Le groupe R contenants les vecteurs restants (bien classés $\alpha_i = 0$)

- **Méthode**

Les coefficients des vecteurs de marge changent de valeurs durant chaque étape incrémentale pour garder tous les éléments dans D en équilibre c'est-à-dire leurs conditions KT satisfaites. Les conditions KT sont exprimées d'une manière différentiable comme suit :

$$\Delta g_i = Q_{ic}\Delta\alpha_c + \sum_{j \in S} Q_{ij}\Delta\alpha_j + y_i \Delta b \qquad \forall i \in D \cup \{c\} \qquad (4.4)$$

$$0 = y_c \Delta\alpha_c + \sum_{j \in S} y_j \Delta\alpha_j \qquad (4.5)$$

Ou α_c est le coefficient qui doit être incrémenté, il prend la valeur nulle ($\alpha_c = 0$) à l'état initial du vecteur candidat à ajouter à l'ensemble D

$$Q.\begin{bmatrix} \Delta b \\ \Delta\alpha_{s_1} \\ . \\ . \\ . \\ \Delta\alpha_{s_{l_s}} \end{bmatrix} = -\begin{bmatrix} y_c \\ Q_{s_1 c} \\ . \\ . \\ . \\ Q_{s_{l_s} c} \end{bmatrix} \Delta\alpha_c \qquad (4.6)$$

Ou Q est la matrice jacobienne symétrique mais non définie positive :

$$Q = \begin{bmatrix} 0 & y_{s_1} & \dots & y_{s_{l_s}} \\ y_{s_1} & Q_{s_1 s_1} & \dots & Q_{s_1 s_{l_s}} \\ . & & . & \\ & & . & \\ & & . & \\ y_{s_{l_s}} & Q_{s_{l_s} s_1} & & Q_{s_{l_s} s_{l_s}} \end{bmatrix} \qquad (4.7)$$

Alors en équilibre on a :

$$\Delta b = \beta \Delta\alpha_c \qquad (4.8)$$

$$\Delta\alpha_j = \beta_j \Delta\alpha_c \quad \forall j \in D \qquad (4.9)$$

Les coefficients β sont calculés par le système d'équations suivant :

$$\begin{bmatrix} \beta \\ \beta_{s_1} \\ . \\ . \\ . \\ \beta_{s_{l_s}} \end{bmatrix} = - R . \begin{bmatrix} y_c \\ Q_{s_1 c} \\ . \\ . \\ . \\ Q_{s_{l_s} c} \end{bmatrix} \qquad (4.10)$$

Ou $R = Q^{-1}$ et $\beta_j \equiv 0 \quad \forall j \notin S$

$$\Delta g_i = \gamma_i \Delta\alpha_c \quad \forall i \in D \cup \{c\} \qquad (4.11)$$

$$\gamma_i = Q_{ic} + \sum_{j \in S} Q_{ij} \beta_j + y_i \beta \qquad \forall i \notin S \qquad (4.12)$$

Et $\gamma_i \equiv 0 \quad \forall i \in S$

- **Limite supérieure de l'incrément $\Delta\boldsymbol{\alpha_c}$**

En ajoutant un nouveau point candidat (c) aux exemples d'apprentissage , les coefficients $\boldsymbol{\alpha_j}$ et les quantités des dérivées partielles $\boldsymbol{g_i}$ peuvent changer de valeurs à travers les formules (4.9) et (4.11) respectivement , quelques calculs sont exigés afin de vérifier chacune des conditions suivantes et de déterminer le pas ($\Delta\alpha_c$) le plus large possible

1. $\boldsymbol{g_c \leq 0}$, **si** $\boldsymbol{g_c = 0}$, le point c devient vecteur support **(** $\boldsymbol{c \in S}$ **)**
2. $\boldsymbol{\alpha_c \leq C}$, **si** $\boldsymbol{\alpha_c = C}$, le point c devient vecteur erreur **(**$\boldsymbol{c \in E}$)
3. Parcourir tous les éléments vecteurs supports et vérifier pour chacun la nouvelle valeur de , la condition nécessaire pour les vecteurs supports est : $\mathbf{0} < \alpha_j < C \quad \forall j \in S$
   - Si $\boldsymbol{\alpha_j}$**= 0 ,** donc l'exemple j quitte l'ensemble S , et sera intégré dans l'ensemble R (sera un exemple bien classé , loin de la frontière)
   - Si $\boldsymbol{\alpha_j}$**= C ,** donc l'exemple j quitte l'ensemble S , et sera intégré dans l'ensemble E (sera un vecteur erreur , entre la marge)
4. Parcourir tous les éléments vecteurs erreurs et vérifier pour chacun la nouvelle valeur de g , la condition nécessaire pour les vecteurs erreurs est : $\boldsymbol{g_i} < 0 \quad \forall i \in E$

   Si $g_i = 0$ alors l'exemple i quitte l'ensemble E et sera intégré dans l'ensemble

   S (devient vecteur support)

5. Parcourir tous les éléments vecteurs restants (initialement bien classés) et vérifier pour chacun la nouvelle valeur de g , la condition nécessaire pour les vecteurs bien classés est : $\boldsymbol{g_i} > 0 \quad \forall i \in R$

Si $g_i = 0$ alors l'exemple i quitte l'ensemble R et sera intégré dans l'ensemble S (devient vecteur support)

- **Mise à jour de la matrice R**

Pour ajouter un point candidat c à l'ensemble des vecteurs supports (S), La matrice sera étendue comme suit :

$$R \leftarrow \begin{bmatrix} & & 0 \\ & R & . \\ & & . \\ & & 0 \\ 0 & \dots 0 & 0 \end{bmatrix} + \frac{1}{\gamma_c} \begin{bmatrix} \beta \\ \beta_{s_1} \\ . \\ . \\ . \\ \beta_{s_{l_s}} \\ 1 \end{bmatrix} . \left[\beta, \beta_{s_1}, \dots . , \beta_{s_{l_s}}, 1\right]$$

Pour supprimer un vecteur support K du groupe S , la matrice sera contractée comme suit :

$$R_{ij} \leftarrow R_{ij} - R_{kk}^{-1} R_{ik} R_{kj} \qquad \forall\, i,j \in S \cup \{O\}; i,j \neq k$$

L'indexe O se réfère au terme b

4.7.3 Pseudo code de la procédure incrémentale

- **Algorithme1** (pseudo code de la procédure incrémentale)

Début

1. Initialiser α_c à 0
2. Si $g_c > 0$, fin { c n'est ni vecteur de marge , ni vecteur erreur }
3. Si $g_c \leq 0$, changer α_c en appliquant l'incrément $\Delta\alpha_c$

   Alors l'un des trois cas se rencontre :

   - $g_c = 0$
     ajouter le nouveau point c à l'ensemble S{c devient un vecteur support} , mettre à jour la matrice R et terminer
   - $\alpha_c = C$
     ajouter c à l'ensemble E {c devient un vecteur erreur) et terminer
   - Vérifier l'état de chaque point $\in D$ pour chacun des trois ensembles
     - S (vecteurs supports)
     - E (vecteurs erreurs)
     - R (vecteurs bien classés)

     Si S change mettre à jour la matrice R
4. Répéter si nécessaire

Fin

4.7.4 Pseudo code de la procédure de validation

- **Algorithme2** (procédure décrémentale l → l -1)

Début

1. Si un point ⊂ (à quitter la solution) n'est ni vecteur support, ni vecteur erreur
   Le point est bien classé (a déjà quitté la solution, ie ne participe pas à la fonction de décision).
2. Si ⊂ est un vecteur support ou vecteur erreur
   Avec $g_c < -1$ fin (par défaut est un point erreur)
3. Si ⊂ est vecteur support ou vecteur erreur
   Avec $g_c > -1$ appliquer la mise à jour de α *par le décrément* $\Delta\alpha_c$
   Alors , on peut rencontrer une des trois conditions suivantes :
   - $g_c < -1$ fin le point est incorrect
   - $\alpha_c = 0$ fin le point est correct
   - Vérifier l'état de chaque point ∈ D pour chacun des trois ensembles
     - S (vecteurs supports)
     - E (vecteurs erreurs)
     - R (vecteurs bien classés)

     Si S change mettre à jour la matrice R
- Répéter si nécessaire

Fin

4.8 l'algorithme incrémental proposé

Basé sur le principe de l'algorithme de référence, nous avons proposé une méthode qui représente une extension de la méthode de laquelle nous nous sommes inspirés c'est-à-dire au lieu que l'algorithme incrémental procède d'ajouter un seul nouvel élément à la solution, donc on peut ajouter plusieurs nouveaux exemples et nous n'allons pas intégrer le scalaire b dans la fonction objective. la valeur de b sera déduite en résolvant un système d'équations linéaire après avoir trouvé les valeurs des coefficients de lagrange concernant les vecteurs supports. Nous avons choisi une fonction objective de maximisation avec la notation matricielle. Dans ce qui suit on commence à montrer le principe de la méthode.

$$\max \mathcal{F} = -\frac{1}{2}\alpha^T G\alpha + e^T\alpha$$

$$\text{Avec} \quad \alpha^T y = 0$$

Ou G est la matrice d'influence de terme général $G_{ij} = y_i y_j K(x_i, x_j)$ et $e^T = (1,1,1,1,\dots,1)^T$

4.8.1 Conditions KKT

$$d_i = \frac{\partial \mathcal{F}}{\partial \alpha_i} = -G\alpha^T + e^T \qquad \begin{cases} > 0 & \alpha_i = 0 \\ = 0 & 0 \le \alpha_i \le C \\ < & 0 \ \alpha_i = C \end{cases} \qquad (4.13)$$

$$m = \alpha^T y = 0 \qquad (4.14)$$

Basé sur les dérivées partielles d_i les données d'apprentissage sont partitionnées dans trois groupes différents :

- Le groupe S contenant les vecteurs supports ($0 < \alpha_i < C$).
- Le groupe E contenant les vecteurs erreurs ($\alpha_i = C$).
- Le groupe R contenant les vecteurs restants (bien classés $\alpha_i = 0$)

4.8.2 Principe de la méthode

Durant l'apprentissage incrémental, de nouveaux exemples avec $(d_i>0)$ sont assignés directement à l'ensemble R et n'interviennent pas dans la solution, tous les autres nouveaux exemples sont initialement des éléments d'un ensemble désigné par N (nouveau) et deviennent éventuellement des vecteurs supports ou vecteurs erreurs.

En ajoutant de nouveaux exemples dans la solution, le but de l'apprentissage est de préserver simultanément les conditions KKT sur les exemples d'apprentissage précédemment vus, les conditions KKT sont vérifiées en variant les coefficients des vecteurs supports et cette variation est provoquée par l'ajout de nouveaux coefficients.

Dans le processus, des éléments de différentes catégories peuvent changer d'état de sorte que l'apprentissage incrémental procède à travers de petites corrections déterminées par une vérification de leurs valeurs suivant la condition (4.13) .

Les dérivées partielles avant l'ajout d'un nouvel ensemble à la solution SVM sont égales à :

$$d_i = -G\alpha^T + e^T = 0 \quad \forall\, i \in S \qquad (4.15)$$

$$m = \alpha^T y = 0 \qquad (4.16)$$

En ajoutant de nouveaux exemples à la base d'apprentissage, les dérivées partielles deviennent comme suit :

$$d_i = -G\alpha^T - G\Delta{\alpha_p}^T - G\Delta{\alpha_t}^T + e^T = 0 \quad \forall\, i \in S \qquad (4.17)$$

$$m = \alpha^T y + \Delta{\alpha_p}^T y + \Delta{\alpha_t}^T y = 0 \quad \forall\, p \in S,\ t \in N \qquad (4.18)$$

Exprimons ces conditions d'une manière différentiable, nous obtenons :

$$\Delta d_i = -G\Delta{\alpha_p}^T - G\Delta{\alpha_t}^T = 0 \quad \forall\, i, p \in S,\ t \in N \qquad (4.19)$$

$$\Delta m = \Delta{\alpha_p}^T y + \Delta{\alpha_t}^T y = 0 \quad \forall p \in S, t \in N \qquad (4.20)$$

Pour une mise à jour donnée par les nouveaux coefficients des vecteurs non encore appris $\{\Delta\alpha_t : \forall t \in N\}$, notre objectif est de déterminer les changements ou les corrections nécessaires des coefficients des vecteurs supports qui sont nommés $\{\Delta\alpha_p : \forall p \in S\}$.

Comme la solution SVM est initialisée par la solution précédente (avant l'ajout d'un nouveau ensemble d'exemples), en ajoutant un nouvel ensemble, la solution va être mise à jour jusqu'à ce que tous les exemples du nouvel ensemble sont appris.

Pour chaque mise à jour, au moins un exemple (appartenant au nouvel ensemble N) peut changer de catégorie.

A l'état final implique que chaque nouvel exemple ($\in N$) atteint une des trois catégories S , E ou R , et tous les exemples satisfont les conditions KKT .

Les quantités $\Delta\alpha_i$ sont exprimées en fonction de Δa et des coefficients β

$$\Delta\alpha_p{}^T = \beta_p{}^T \Delta a \qquad (p \in S)$$

$$\Delta\alpha_t{}^T = \mu_t{}^T \Delta a \qquad (t \in N)$$

Substituons ces expressions dans les équations (4.19) et (4.20) et divisons chacune par Δa

$$\gamma_i = \frac{\Delta d_i}{\Delta a} = -G\beta_p{}^T - G\mu_t{}^T = 0 \quad \forall p \in S \; et \; t \in N \qquad (4.21)$$

$$\frac{\Delta m}{\Delta a} = \beta_p{}^T y + \mu_t{}^T y = 0 \qquad (4.22)$$

les coefficients des vecteurs nouveaux devraient avoir comme valeur maximale celle du paramètre de régularisation C, donc tous les coefficients de ces nouveaux exemples peuvent être initialisés à cette valeur C . $\{\mu_t = C \quad \forall t \in N\}$.

Les coefficients $\beta_p \; \forall \; p \in S$ sont calculés par la résolution de ce système d'équations linéaires.

Une fois les coefficients β sont connus , on peut calculer les coefficients γ_i pour les vecteurs erreurs (E) , les vecteurs (R) et les nouveaux vecteurs (N) .

$\{\beta_p \; \forall p \in S\} \; et \; \{\gamma_i, \forall i \in E, R, N\}$ sont nécessaires pour calculer Δa_{min} .

Le tableau (4.1) montre les changements de catégories possibles qui peuvent être durant l'apprentissage incrémental, la plus petite valeur Δa parmi les conditions applicables détermine le changement de catégorie et le pas Δa_{min} .

Une fois Δa_{min} est connu, on met à jour les coefficients des vecteurs supports et les coefficients des vecteurs nouveaux ($\in N$)

$$\{\, \alpha_p \rightarrow \alpha_p + \beta_p \Delta a \quad \forall\, p \in S \,\}\, \{\, \alpha_t \rightarrow \alpha_t + \mu_t \Delta a \quad \forall\, t \in N \,\}$$

Ce processus se répète jusqu'à ce qu'il ne reste aucun élément.

état initial	état final	Valeur de Δa	Condition
Vecteur support	Vecteur reserve	$\frac{-\alpha_i}{\beta_i}$	$\beta_i < 0$
Vecteur erreur	Vecteur support	$\frac{-d_i}{\gamma_i}$	$\gamma_i > 0$
Vecteur reserve	Vecteur support	$\frac{-d_i}{\gamma_i}$	$\gamma_i < 0$
Vecteur support	Vecteur erreur	$\frac{C-\alpha_i}{\beta_i}$	$\beta_i > 0$
Nonappris ($d_i < 0$)	Vecteur support	$\frac{-d_i}{\gamma_i}$	$\gamma_i > 0$
Nonappris ($d_i > 0$)	Vecteur erreur	$\frac{C-\alpha_i}{\mu_i}$	$\mu_i > 0$

Tableau 4.1 : Valeurs de Δa selon les catégories

Interprétation du tableau 4.1

1/ Vecteur Support devient Vecteur Reserve

On sait que pour un vecteur reserve c'est-à-dire il est loin de la frontière (bien classé) , sa valeur du coefficient de lagrange est nulle (α =0) , donc on a :

$\alpha_i + \beta_i \Delta a = 0 \implies \Delta a = \frac{-\alpha_i}{\beta_i}$, la valeur du pas (Δa) doit être positive, donc β_i doit être négative $\beta_i < 0$ (voir la première ligne du tableau 4.1)

2/ Vecteur Erreur devient Vecteur Support

Un vecteur erreur ($d_i < 0$), devient vecteur support ($d_i = 0$)

Donc $\boldsymbol{d_i + \Delta d_i = 0 \implies d_i + \gamma_i \Delta a = 0 \implies \Delta a = \frac{-d_i}{\gamma_i}}$ **($d_i < 0$)** ($\Delta a > 0$)

Donc $\boldsymbol{\gamma_i}$ doit être positive $\boldsymbol{\gamma_i > 0}$ (voir la deuxième ligne du tableau 4.1)

3/ Vecteur Reserve devient Vecteur Support

Un vecteur reserve ($d_i > 0$) devient vecteur support ($d_i = 0$)

Donc $\boldsymbol{d_i + \Delta d_i = 0 \implies d_i + \gamma_i \Delta a = 0 \implies \Delta a = \frac{-d_i}{\gamma_i}}$ **($d_i > 0$)** ($\Delta a > 0$)

Donc $\boldsymbol{\gamma_i}$ doit être négative $\boldsymbol{\gamma_i} < 0$

4/ Vecteur Support devient Vecteur Erreur

Un vecteur erreur $(\alpha = C)$, donc $\alpha_i + \beta_i \Delta a = C \implies \Delta a = \frac{C - \alpha_i}{\beta_i}$,

$C - \alpha_i > 0 \ car \ \forall i \ \ 0 \leq \alpha_i \leq C \quad , \quad donc \ \beta_i > 0$

5/ Vecteur Nouveau ($\boldsymbol{d_i} < 0$)devient Vecteur Support

C'est la même interprétation que le deuxième cas c'est-à-dire un vecteur erreur devient vecteur support.

6/ Vecteur Nouveau $\boldsymbol{d_i} > 0$) devient Vecteur Erreur

Un vecteur erreur $(\alpha = C)$,donc $\alpha_i + \mu_i \Delta a = C \implies \Delta a = \frac{C - \alpha_i}{\mu_i}$,

$C - \alpha_i > 0 \ , donc \ \mu_i > 0$

4.8.3 Initialisation de la solution

- Une considération spéciale doit être donnée pour le cas où la solution initiale du SVM n'existe pas. dans ce cas tous les exemples d'apprentissage sont initialement non appris ($\alpha_t = 0 \quad \forall \ t \in N$) ceci implique que la condition d'égalité (4.14) est immédiatement satisfaite.

4.8.4 Pseudo code de l'algorithme

Algorithme (procédure de l'apprentissage)

Début

$\{\alpha_i, b\} \leftarrow \{\alpha_i, b\}$ {initialiser avec la solution courante }

$\mu_t \leftarrow C \quad \forall t \in N$ {initialiser les coefficients des nouveaux exemples à C }

Répéter

- **Calculer les coefficients** $\{\beta_p \ \forall p \in S\}$ par la résolution du système d'équations linéaires
- Calculer les coefficients $\gamma_i \ \forall i \in \{E, R, N\}$
- Calculer le pas minimal Δa_{min}
- Mettre à jour les coefficients des vecteurs supports

$$\alpha_p = \alpha_p + \beta_p * \Delta a_{min} \quad \forall p \in S$$

- Mettre à jour les coefficients des vecteurs nouveaux

$$\alpha_t = \alpha_t + \mu_t * \Delta a_{min} \quad \forall t \in N$$

- Calcul du scalaire b(résolution système d'équations En fonction des coefficients des vecteurs supports)

Jusqu'à ce qu'il ne reste aucun nouvel exemple non traité

Fin

4.8.5 Environnement d'implémentation

Nous avons choisi MATLAB comme environnement d'implémentation pour les facilités qu'il offre dans la manipulation des matrices et toutes les opérations élémentaires sur les matrices telles que la transposée, l'inverse , etc.. sont prédéfinies dans le langage. Nous avons donc utilisé une matrice dont les colonnes contenant les exemples d'apprentissage et les lignes les caractéristiques(ou dimension), même chose pour la base de test , nous avons choisi des vecteurs à une colonne. deux vecteurs contenant respectivement les classes (+1 ou -1) des exemples d'apprentissage et les exemples de test. Un autre vecteur contenant les valeurs des multiplicateurs de lagrange et bien sûr d'autres vecteurs dont nous avons besoin dans notre implémentation.

4.9 Résultats expérimentaux

4.9.1 Résultats de Simulation : Algorithme (C & P)

Nous allons simuler l'algorithme sur une base d'apprentissage artificielle (30 éléments) de dimension 2 et une base de test de 200 éléments.

- Premier cas : On choisit sigma =0.20 fixe, et on fait varier le paramètre C (C=10 C=1000 C=0.01)

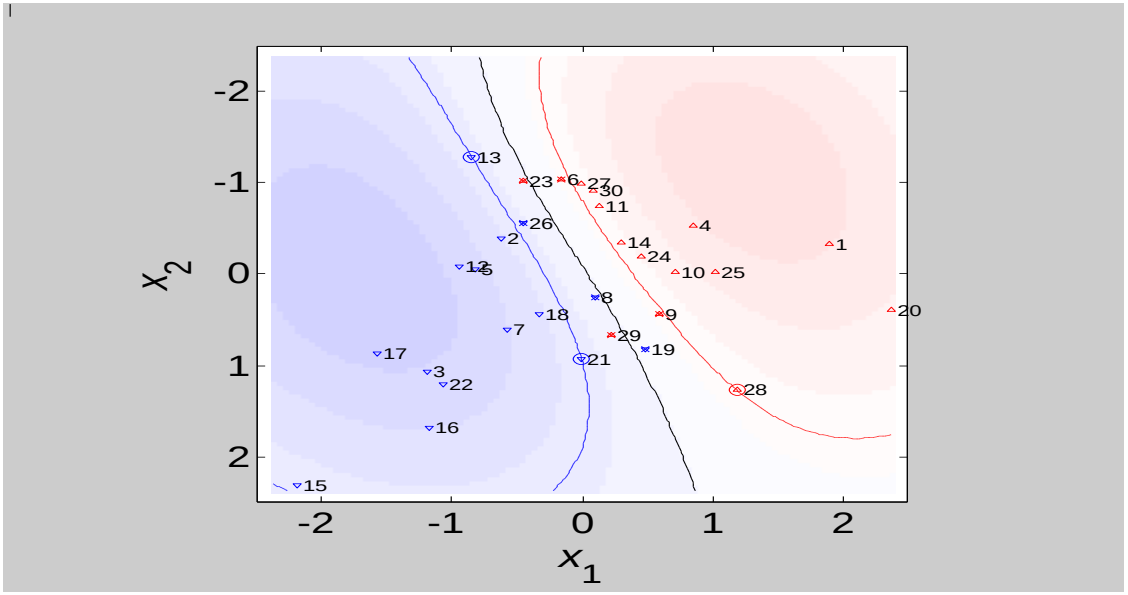


Figure 4.1 C=10 , Noyau RBF , Sigma=0.2

Résultats: Vecteurs Supports =3, Vecteurs Erreurs =7 et performance en généralisation =92%

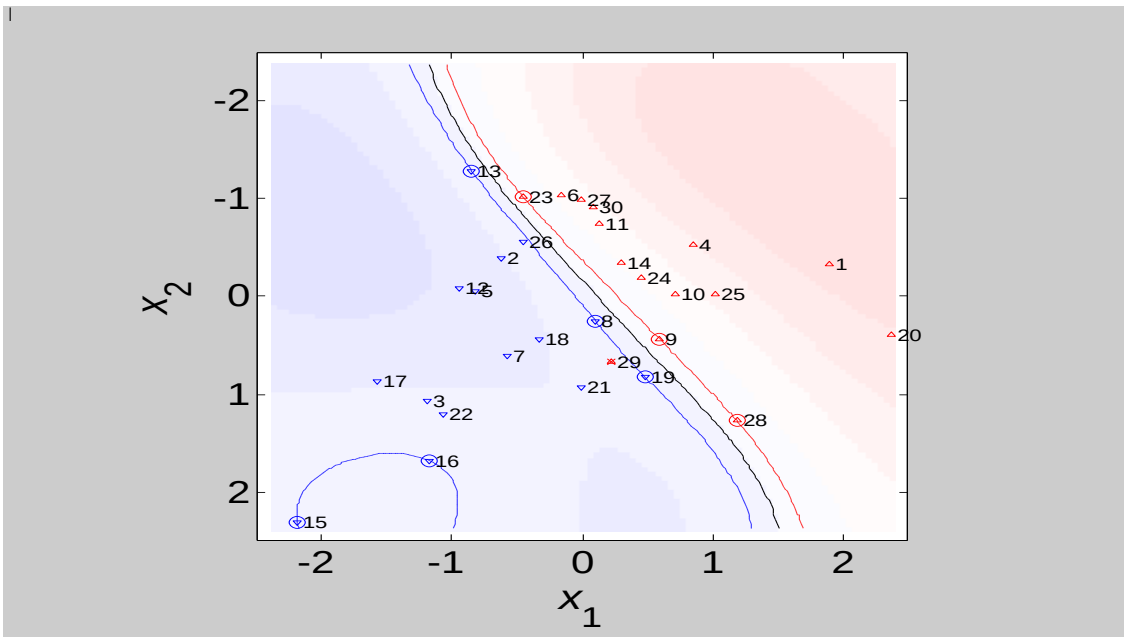


Figure 4.2 C=1000 Noyau RBF Sigma =0.2

Résultats :Vecteurs Supports =8, Vecteurs Erreurs =1 et performance en généralisation =87.5%

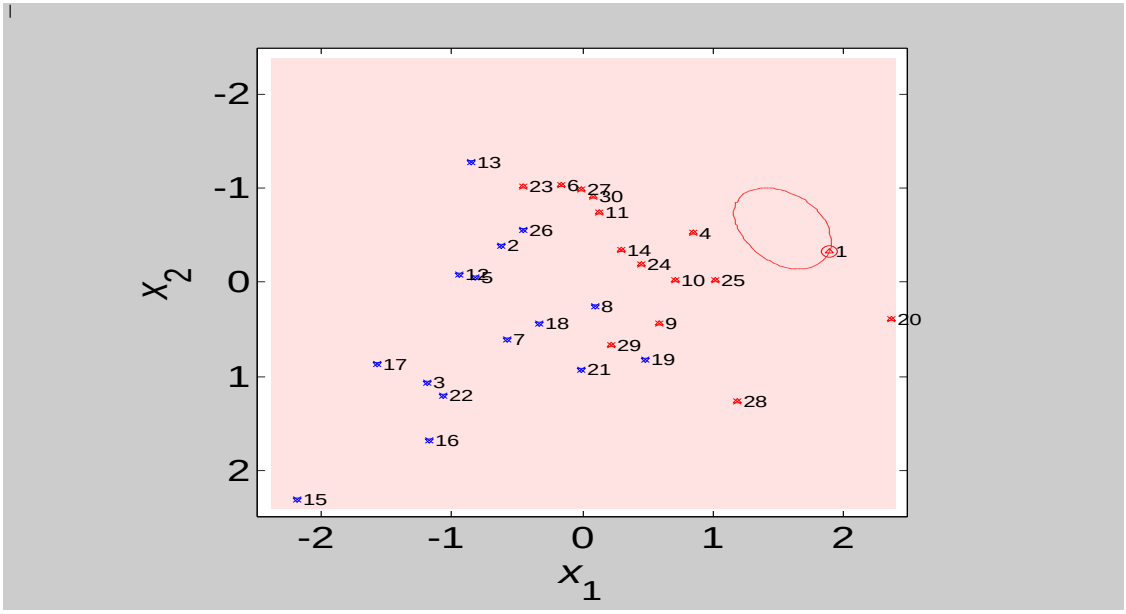


Figure 4.3 C=0.01 Noyau RBF Sigma = 0.2

Résultats : Vecteurs Supports=1 , Vecteurs Erreurs=29 et performance en généralisation =45.5%

- **Interprétation des Résultats :**

En minimisant C, on voit que le nombre de vecteurs erreurs augmente (ce sont les points se trouvant à l'intérieur de la marge), ainsi que la marge augmente, donc la performance en généralisation diminue, voire figure (4.3), la performance en généralisation égale à 45.5% , ainsi qu'on ne pas différencier les points des deux classes c'est le phénomène du sur apprentissage.

Si on observe le deuxième cas (figure 4.2), on trouve que le nombre de vecteurs erreurs est seulement un(1), la marge est petite, la performance n'est pas meilleure (87.50%) c'est le phénomène de l'apprentissage par cœur.

Donc le meilleur cas c'est le premier, on observe le nombre d'erreurs égal 7 c'est-à-dire on a accepté d'avoir des points mal classifiés, pour la base d'apprentissage, mais la performance en généralisation est très bonne 92% .

- Deuxième cas : On fixe C=10 et on fait varier Sigma

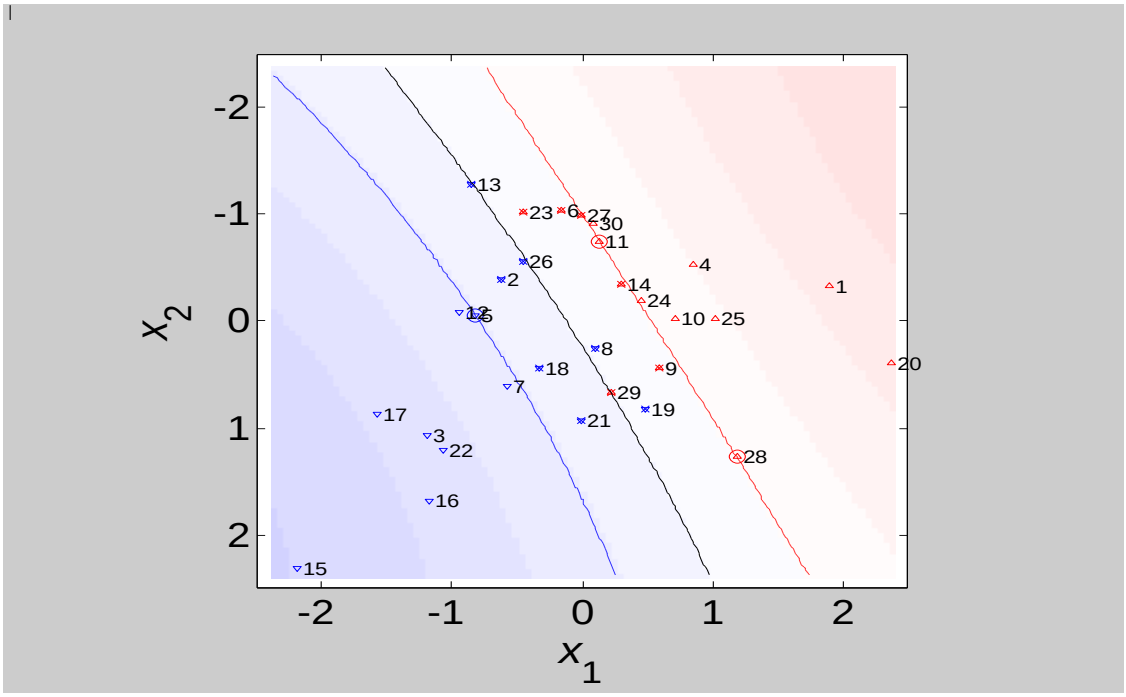


Figure 4.4 C = 10 Noyau RBF Sigma = 0.025

Résultats : Vecteurs Supports= 3, Vecteurs Erreurs= 13, performance = 94%

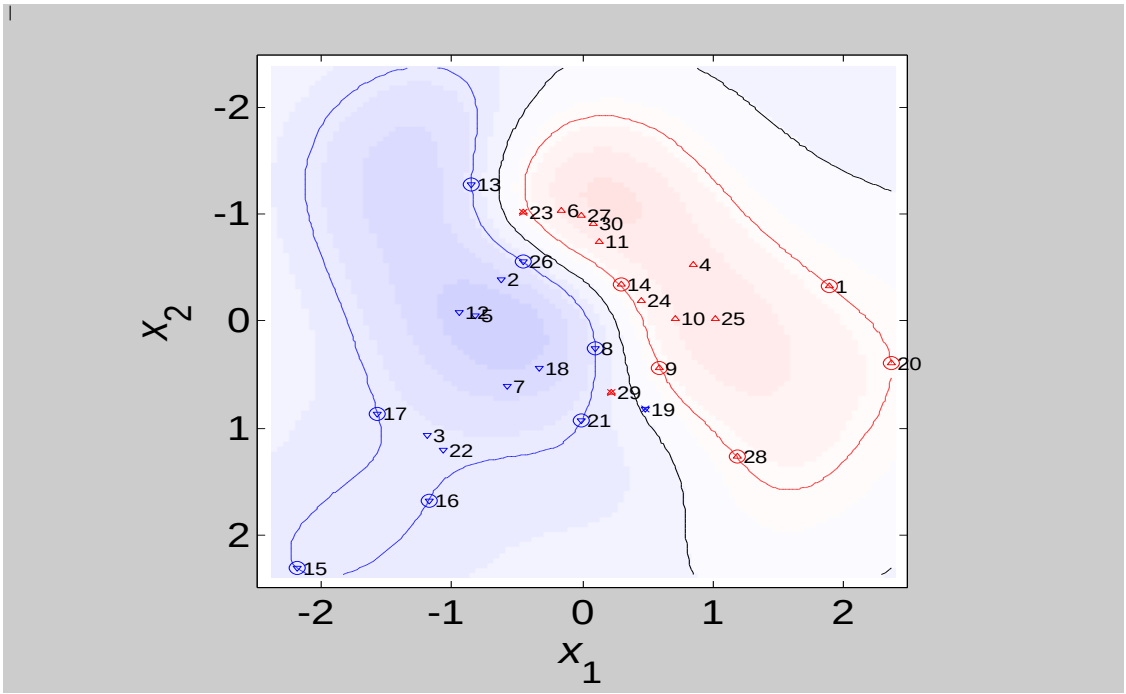


Figure 4.5 C=10 Noyau RBF Sigma = 1

Résultats : Vecteurs Supports=12, Vecteurs Erreurs=3, performance =87.5%.

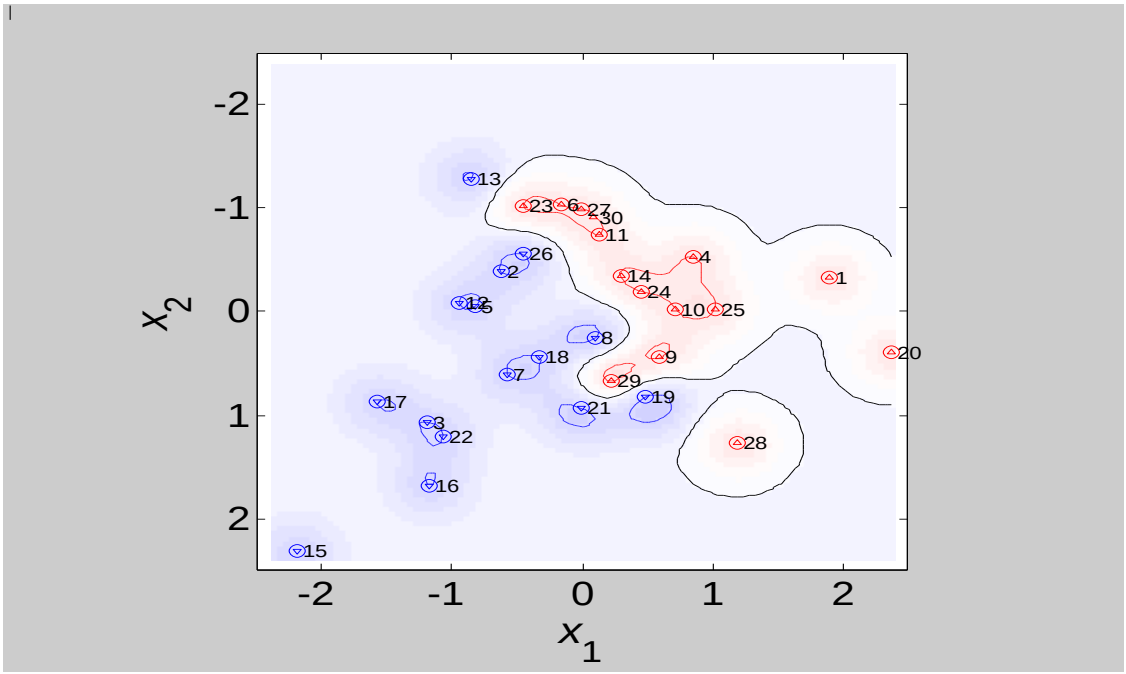


Figure 4.6 C =10 Noyau : RBF Sigma= 10

Résultats : Vecteurs Supports=29, Vecteurs Erreurs=0, performance = 60.5%

- **Interprétation des résultats :**

En augmentant sigma, le terme de l'exponentiel tend vers 0 , la distance diminue c'est-à-dire la similarité entre les exemples diminue, donc la fonction de décision doit inclure plus de vecteurs supports pour couvrir tout l'espace, c'est le cas de l'apprentissage par cœur , comme cas extrême voir figure 4.6 ou le nombre de vecteurs supports est égal à 29 , la performance en généralisation est mauvaise (60.5%) .

En comparant les deux figures 4.5 (sigma = 1) et 4.4 (sigma =0.025)

- Pour la figure (4.5), le nombre de vecteurs supports égal à 12, le nombre de vecteurs erreurs est 3 et la performance en généralisation est égale à 87.5%, par contre dans la figure (4.4), le nombre de vecteur supports est égal à 3, le nombre de vecteurs erreurs est 13 et la performance en généralisation est meilleure (94%).
- Donc on peut dire que les résultats de la figure 4.4 sont meilleurs, du moment qu'on a accepté des erreurs lors de la phase d'apprentissage (13 erreurs) mais nous avons commis moins d'erreurs lors de la phase de test (erreur de 6% ,12 points mal classifiés sur 200).

En comparant la figure (4.5) (sigma = 1) et la figure (4.6) (sigma =10) , on peut dire que les résultats de la figure (4.5) sont meilleurs , nombre de vecteurs erreurs est 3 , mais la performance est de 87.5%, par contre dans la figure 4.6 , on trouve aucune erreur dans la phase d'apprentissage , mais la performance est de 60.50%..

4.9.2 Résultats expérimentaux du nouvel algorithme incrémental

Base de données artificielle, Base d'apprentissage est de 30 exemples (figure N°4.7) et la base de test est de 200 exemples (figure N° 4.8)

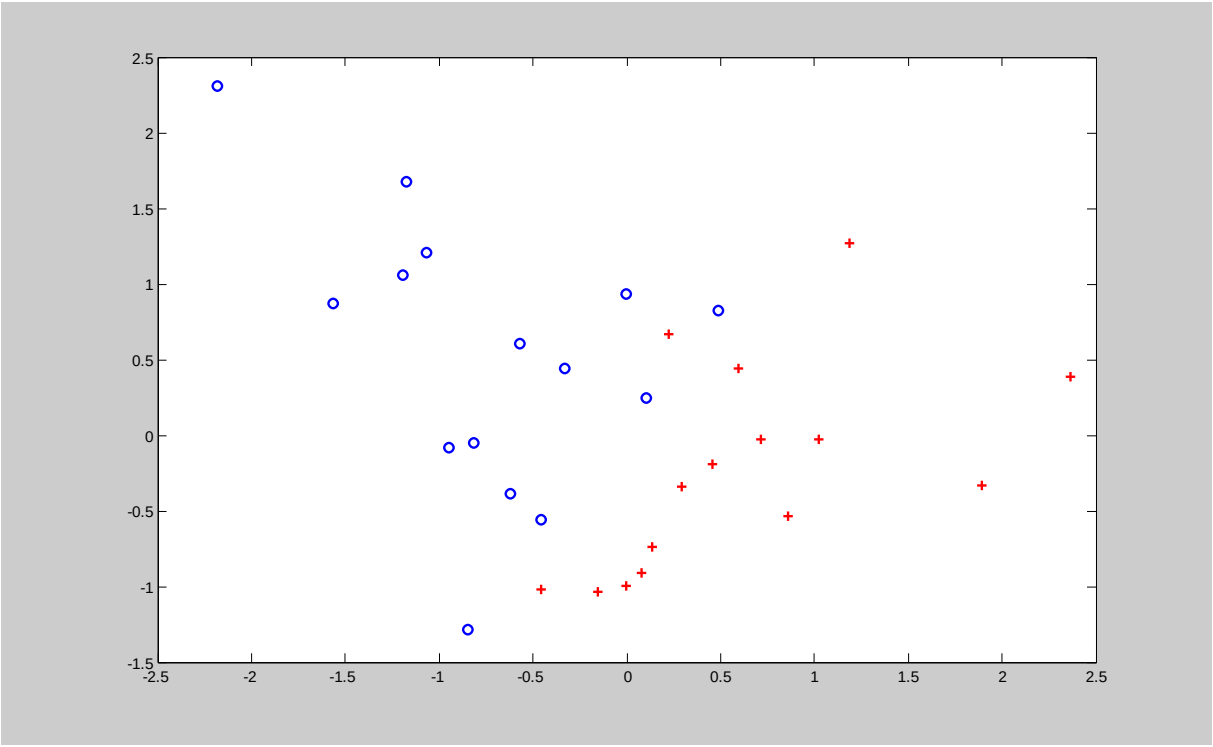


Figure 4.7 : Base d'apprentissage de 30 éléments

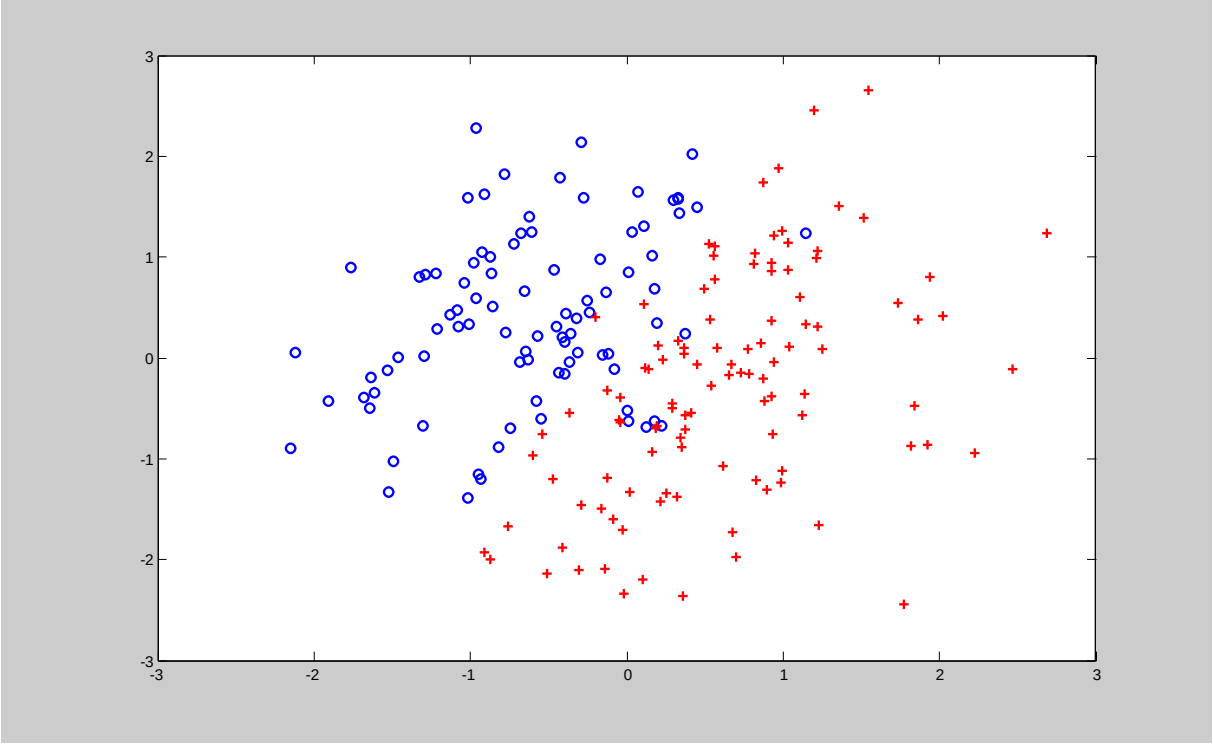


Figure 4.8 : Base de Test de 200 éléments

Lors de la simulation de l'algorithme de (C & P) sur une base artificielle, nous avons utilisé le noyau RBF, on choisi maintenant le noyau linéaire.

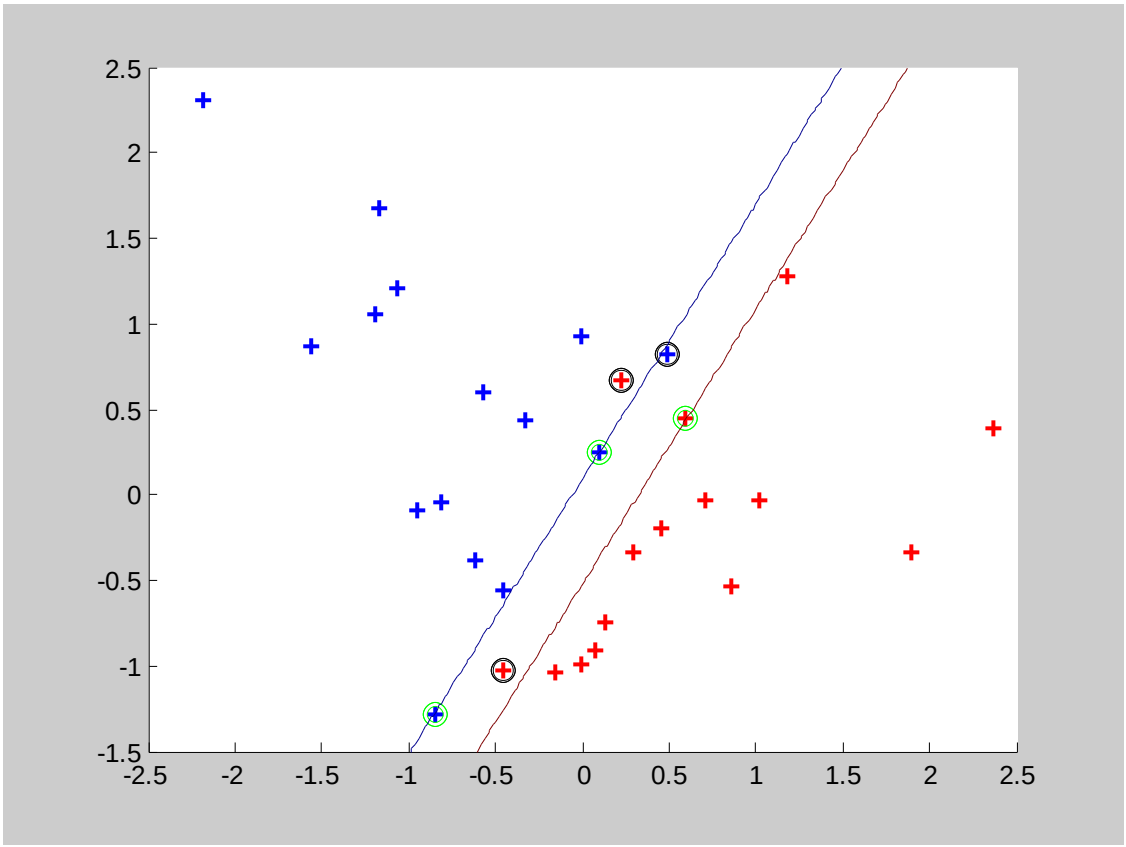


Figure 4.9 : Résultat de Simulation (Noyau linéaire, C=100)

Résultats :VS = 3 , VE =3 , Performance en généralisation = 89.5% ,(23 points incorrects)

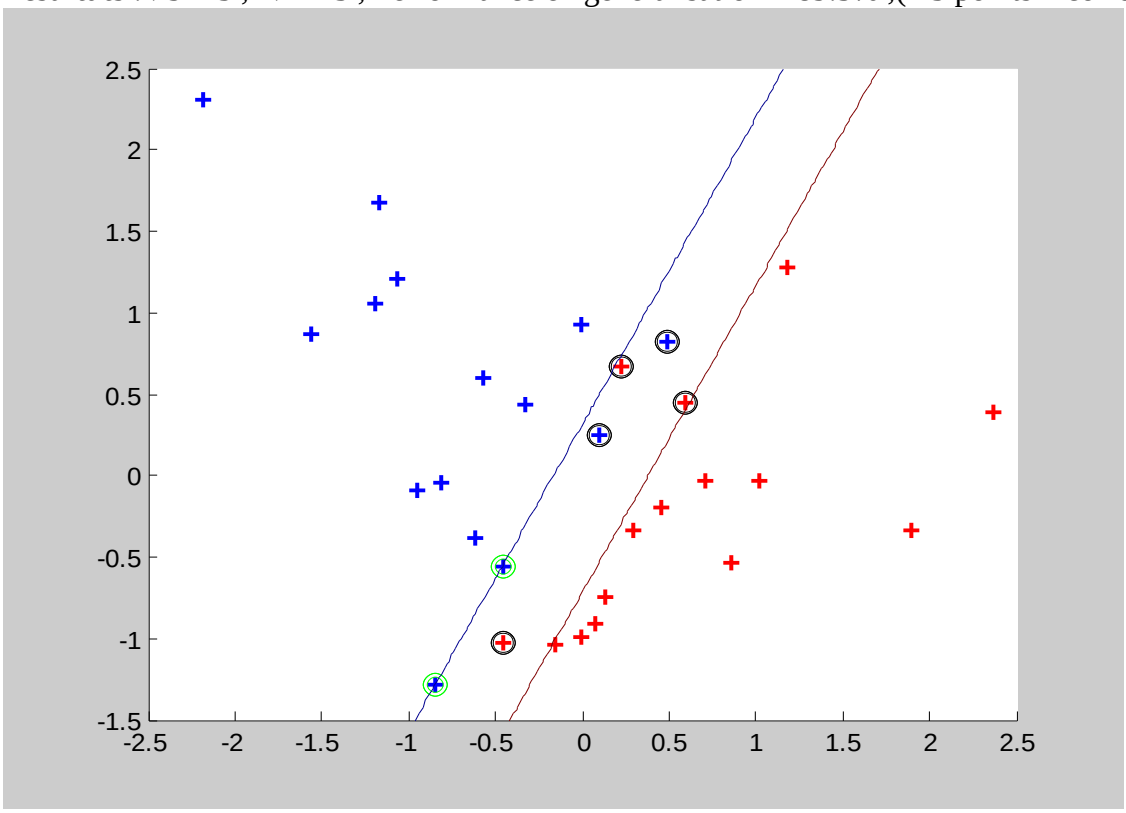


Figure 4.10 : Résultat de Simulation (Noyau linéaire, C=10)

Résultats : VS = 2 , VE = 5 , Performance en généralisation = 90,5 %

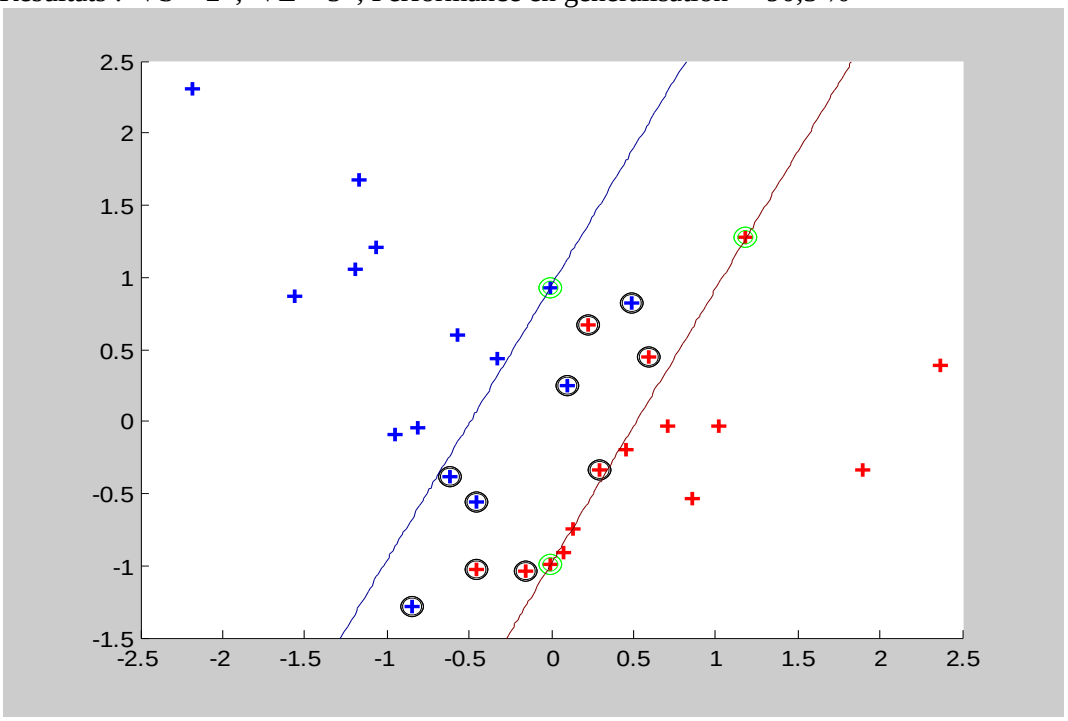
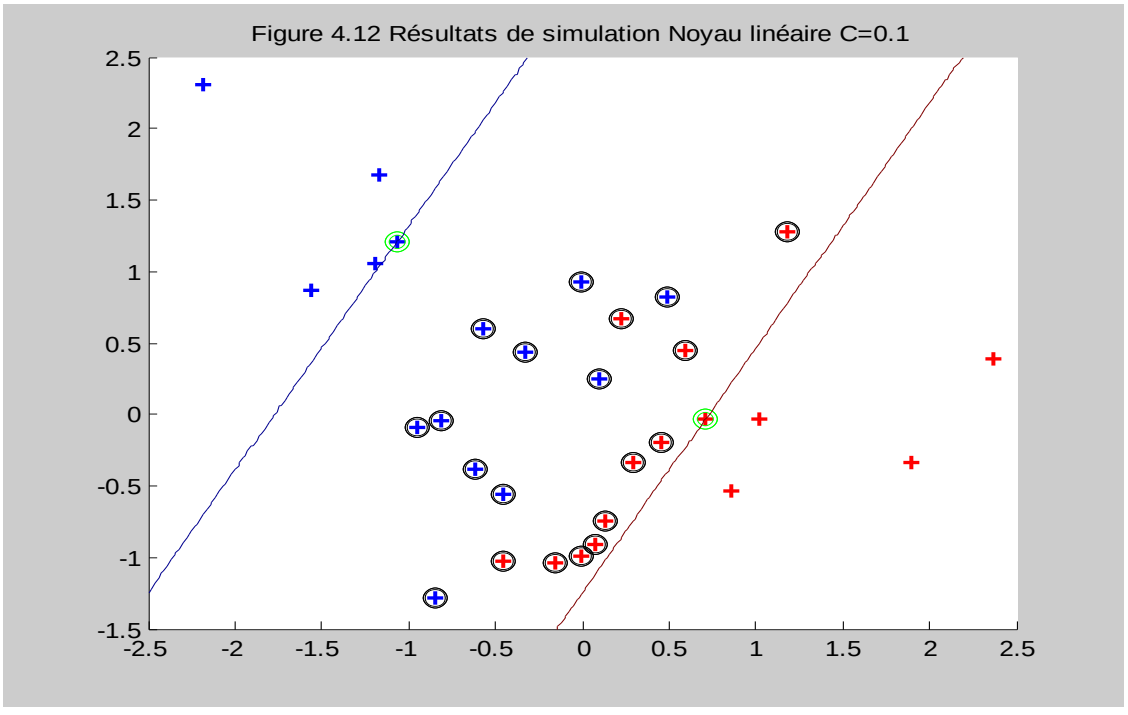


Figure 4.11 : Résultat de simulation (noyau linéaire , C= 1)

Résultats : VS=3 , VE= 10 , Performance = 91.5%



Résultats : Vecteurs Supports = 2 , Vecteurs erreurs =20, Performance = 84.5%

- **Interprétation des résultats**

En appliquant le nouvel algorithme incrémental à la même base de donnée artificielle, du moment que les exemples sont linéairement séparables, on choisit d'utiliser un noyau linéaire

A travers ces quatre cas de simulation, on trouve que nos résultats pratiques s'accordent bien à la théorie, donc en minimisant le paramètre de régularisation C, on voit que le nombre de vecteurs erreurs augmente, la performance en généralisation diminue (voir cas 4)

On choisi maintenant le meilleur cas de simulation de l'algorithme (C & P), c'est bien le cas de la figure (4.4) (Noyau RBF C =10 sigma = 0.025 performance = 94%), donc on utilise les mêmes paramètres et on va simuler avec l'algorithme proposé :

Noyau RBF , C=10 , sigma = 0.025

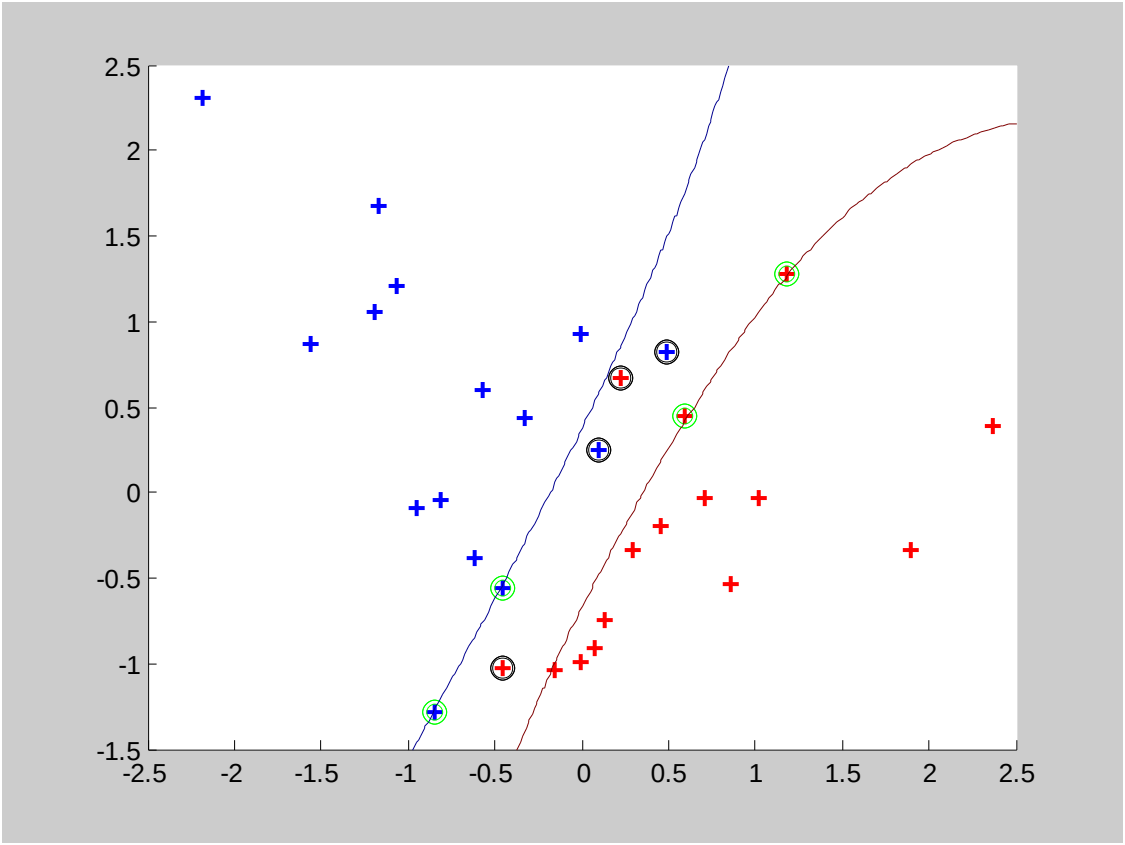


Figure 4.13 : Résultat de Simulation (Noyau RBF, Sigma = 0.025, C=10)

Résultats : Vecteurs Supports = 4 ,Vecteurs erreurs = 4

Performance en généralisation = 93% (14 points incorrects sur 200)

Donc, on voit que les résultats des deux algorithmes sont très proches, 94% pour l'algorithme (C & P) et 93% pour le nouvel algorithme.

4.9.3 Résultats de Simulation sur les chiffres manuscrits

Dans cette section, nous allons simuler les deux algorithmes incrémentaux au cas de la reconnaissance des chiffres manuscrits. Du moment que les deux algorithmes ne traitent pas le cas multi classe, nous avons choisi de les appliquer aux paires des chiffres les plus fréquemment confondus. Différents noyaux seront utilisés avec différents paramètres pour évaluer les performances de ces deux algorithmes incrémentaux et on peut comparer les résultats obtenus avec ceux des algorithmes hors ligne (LIBSVM et SVMLIGHT) vus au chapitre précédent.

1- Base de données USPS

- **Classe 1 et la classe 7**

Base d'apprentissage = 1858, Dimension =256 , Base de test = 529

Noyau Poly d 2	**Vs**	**Vsb**	Perf %	**tex(s)**		**Noyau rbf 0.025**	**Vs**	**Vsb**	**Perf %**	**texec**
C & p	68	0	99.4	40.17		C &P	230	0	99.8	248.55
Inc_nouv	68	0	99.4	14.04		Inc_nouv	1333	0	80	481.16
Libsvm	87	27	99.2	0.15		Libsvm	230	0	99.24	0.31
Svmlight	68	0	99.4	0.89		Svmlight	229	0	99.5	1.13

Tableau 4.1:Tableau comparatif des résultats (classes 1 et 7)

- **Classe 5 et la classe 6**

Base d'apprentissage = 1208, Dimension = 256 , Base de test = 360

Noyau Poly d 2	**Vs**	**Vsb**	**Perf %**	**Texec**		**Noyau rbf 0.025**	**Vs**	**Vsb**	**Perf %**	**Texec**
C & p	98	0	98.3	41.99		C&P	273	0	99.2	318.81
Inc_nouv	98	0	98.3	6.89		Inc_nouv	1116	0	91.0	205.15
Libsvm	92	23	98.3	0.15		Libsvm	272	0	99.16	0.34
Svmlight	98	0	99.8	0.60		Svmlight	273	0	99.4	1.82

Tableau 4.2 : Tableau comparatif des résultats (classes 5 et 6)

- **Classe 4 et la classe 9**

Base d'apprentissage=1200, Dimension=256 , Base de test =332

Noyau Poly d 2	**Vs**	**Vsb**	**Perf %**	**Texec**	**Noyau rbf 0.025**	**Vs**	**Vsb**	**Perf%**	**Texec (s)**
C & p	89	0	97	29.92	C & p	254	0	97.6	282.11
Inc_nouv	89	0	97	6.94	Inc_nouv	1059	0	91.6	270.92
Libsvm	118	52	97.5	0.15	Libsvm	254	0	97.6	0.31
Svmlight	90	0	99.7	0.50	Svmlight	255	0	99.8	0.70

- **Interprétation des résultats**

Selon les résultats mentionnés sur les trois tableaux, on voit que :

- pour le cas du noyau polynomial, les résultats pour les quatre algorithmes sont très proches selon le nombre de vecteurs supports obtenus (vs), la solution obtenue dépend uniquement de ces derniers, donc on peut dire que tous les quatre algorithmes aboutissent à une solution parcimonieuse. Maintenant si nous les comparons selon la performance en généralisation on peut dire que les résultats sont presque les mêmes. Pour la complexité calculatoire on peut remarquer que le nouvel algorithme est meilleur par rapport à l'algorithme de C&P, mais les deux algorithmes classiques, LIBSVM et SVMLIGHT sont les meilleurs en complexité calculatoire.

- Pour le cas du noyau RBF, le nouvel algorithme a donné une solution non parcimonieuse pour chaque paire de classes car le nombre de vecteurs supports est élevé (voir sur les trois tableaux les valeurs (Vs) soulignées) par rapport aux trois autres algorithmes qui ont donné tous le même nombre de vecteurs supports pour chaque cas.

- Sur la figure 4.14, nous avons représenté la variation de la performance en généralisation en fonction du paramètre sigma du noyau RBF pour les deux algorithmes incrémentaux, pour la paire de chiffres 4 et 9. pour la valeur 0.025, la performance de l'algorithme (C&P) est de 97% et pour le nouvel algorithme est de 91%. En augmentant la valeur de sigma la performance en généralisation se dégrade jusqu'à une performance de 55% pour le nouvel algorithme et 51% pour l'algorithme de (C&P).

- Sur la figure 4.15 sont représentés les taux de reconnaissance pour les quatre algorithmes et pour les deux types de noyaux, on voit que pour le noyau polynomial notre algorithme atteint une bonne performance en généralisation de même que les autres algorithmes, mais pour le noyau RBF la performance est peu dégradée par rapport aux autres algorithmes.

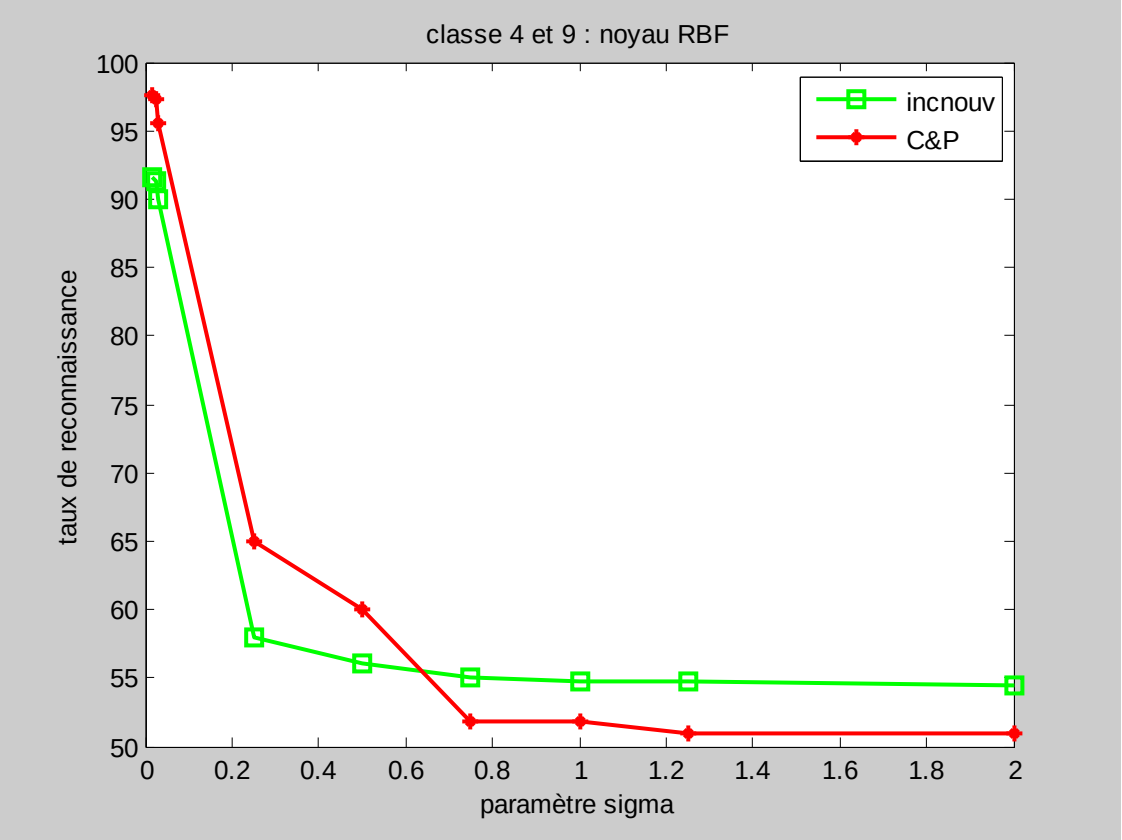


Figure 4.14 : Taux de reconnaissance en fonction du paramètre sigma
Du noyau RBF (Exemple classes 4 et 9)

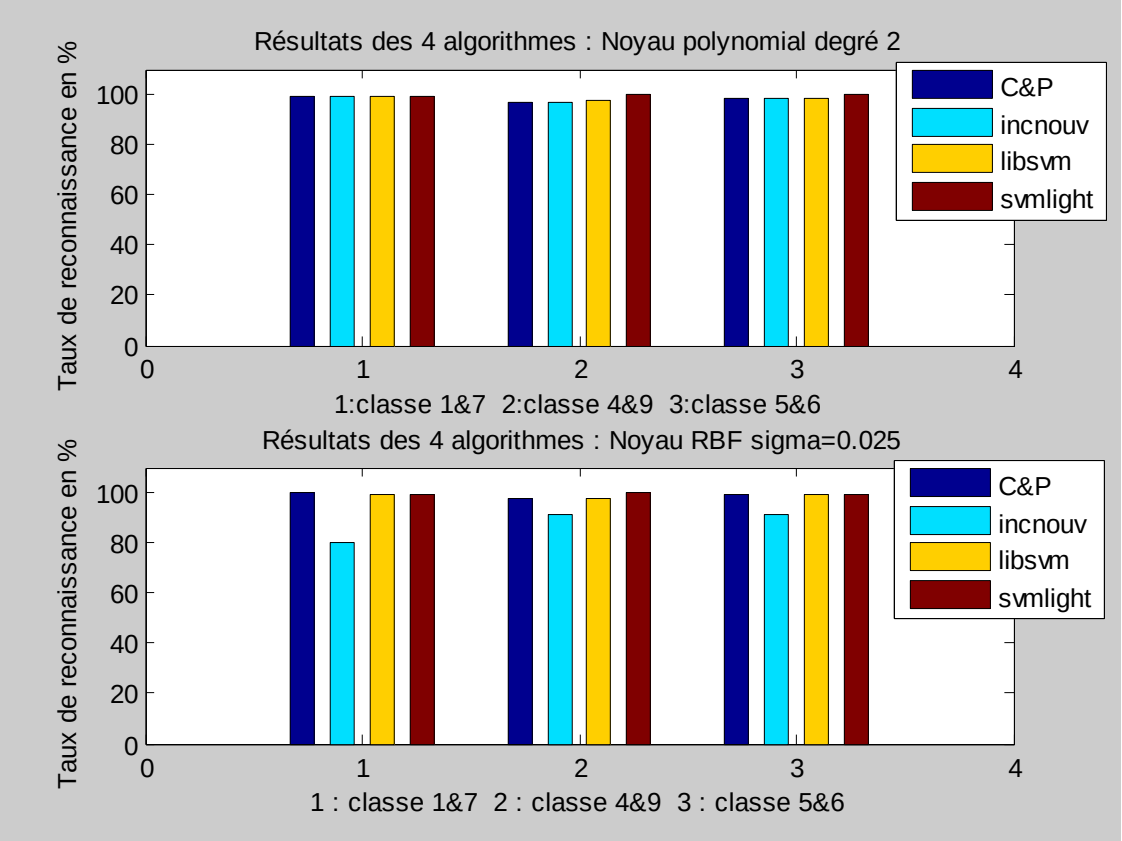


Figure 4.15 : Taux de reconnaissance pour les quatre algorithmes.

2- Base de données MNIST

On va maintenant exécuter l'algorithme proposé aux paires de chiffres manuscrits extraites de la base de données MNIST, vu que chacune des paires est de cardinalité importante, donc on va l'exécuter selon deux modes (hors ligne et en ligne). L'objectif est de montrer l'efficacité de l'algorithme proposé si on l'applique à des bases de données volumineuses et voir ses avantages si on l'exécute de manière incrémentale (en ligne) par rapport à une exécution de manière classique (ou hors ligne).

Les trois paires de classes sont les suivantes :

- Classe 1 et 7 : base d'apprentissage contient 13007 exemples, de dimension 778 et la base de test contient 2163 exemples de dimension 778.
- Classe 4 et 9 : base d'apprentissage contient 11791 exemples, de dimension 778 et la base de test contient 1991 exemples de dimension 778.
- Classe 5 et 6 : base d'apprentissage contient 11339 exemples, de dimension 778 et la base de test contient 1850 exemples de dimension 778.

- Mode hors ligne : Dans ce cas pour chacune des trois expérimentations (3 paires de classes) tous les exemples de la base d'apprentissage seront transmis en une seule passe (totalité des exemples est chargée en mémoire). Nous avons choisi le noyau polynomial de degré 2, Les résultats expérimentaux sont mentionnés dans le tableau 4.4

Classe	Temps cpu (s)	Nombre vecteurs Supports	Performance %
Classe 1&7	985.36	260	99.4
Classe 4&9	1520.23	848	98.9
Classe 5&6	1666.3	591	99.5

Tableau 4.4 Résultats du nouvel algorithme en mode hors ligne sur les 3 classes(MNIST)

- Mode en ligne (incrémental) : dans ce cas, on ne charge pas en mémoire la totalité de la base d'apprentissage, donc nous avons choisi de partager chaque base d'apprentissage en deux sous ensembles dont le premier sous ensemble sera transmis à l'algorithme pour une exécution initiale (première phase), dont une solution SVM est obtenue que l'on doit enregistrer afin de l'utiliser pour la phase suivante. Ensuite une seconde exécution sera faite en transmettant à l'algorithme le deuxième sous ensemble, dans ce cas l'algorithme met à jour la solution précédente (trouvée à la première phase).

  Les sous ensembles sont les suivants :

- Classe 1 et 7 : premier sous ensemble : 6000 exemples, deuxième sous ensemble : 7007 exemples
- Classe 4 et 9 : premier sous ensemble : 6791, deuxième sous ensemble : 5000
- Classe 5 et 6 : premier sous ensemble : 6339, deuxième sous ensemble : 5000

Les résultats expérimentaux sont les suivants :

Classe	Temps cpu (s)	Nombre vecteurs Supports	Performance %
Classe 1&7(phase1)	116.01	173	99.3
Classe 1&7(phase2)	288.96	260	97.5
Classe 4&9(phase1)	709.36	626	98.9
Classe 4&9(phase2)	1610.3	848	98.7
Classe 5&6(phase1)	347.13	413	99.0
Classe 5&6(phase2)	500.38	591	96.8

Tableau 4.5: Résultats du nouvel algorithme en mode incrémental sur les 3 classes (MNIST)

- **Interprétation des résultats**

Selon les résultats des deux tableaux 4.4 et 4.5

- **Nombre de vecteurs supports :** pour chaque paire de classe, on trouve le même nombre de vecteurs supports pour les deux modes d'exécutions.
  Par exemple on prend la classe 1 et 7, le nombre de vecteurs supports est 260 pour le mode hors ligne (voir tableau 4.4), qui est identique à celui du mode incrémental (voir tableau 4.5, la deuxième ligne qui correspond à la phase2 (phase finale)).
  Lors de la première phase, le nombre de vecteurs supports est 173 c'est le résultat de l'apprentissage sur le premier sous ensemble d'exemples.

- **Performance en généralisation :** On remarque qu'en mode incrémental, la performance en généralisation est proche de celle du mode hors ligne. les performances en mode hors ligne pour les trois paires de classes 1&7,4&9,5&6 sont respectivement 99.4%, 98.9% , 99.5% contre celles en mode incrémental 97.5%, 98.7%, 96.8%.

- **Complexité calculatoire :** On remarque que le temps d'exécution en mode incrémental est inférieur à celui du mode hors ligne pour les trois paires de classes.
  Par exemple pour la paire de classe1&7 :
  Le temps d'exécution (hors ligne) = 985.36 s (voir tableau 4.4)
  Le temps d'exécution (incrémental) = temps (phase1) + temps (phase2)
  = 404 .97s (116.01+288.96) (voir tableau 4.5)

4.10 Conclusion

A travers ces expérimentations, Nous pouvons dire que notre algorithme incrémental a donné de bonnes performances en généralisation aussi bien que l'algorithme incrémental de référence (C&P), et une complexité calculatoire minime.

Si nous le comparons avec un algorithme de référence hors ligne tel que SMO de LIBSVM, nous pouvons dire que les performances en généralisation sont très proches, mais l'algorithme SMO a la meilleure complexité calculatoire, que nous la voyons logique puisque tous les échantillons de l'apprentissage sont totalement chargés en mémoire.

Le nouvel algorithme a donné de bons résultats pour le cas du noyau polynomial par rapport au noyau RBF selon tous les critères qui sont la complexité calculatoire, nombre de vecteurs supports et performance en généralisation.

Les algorithmes LIBSVM et SMO montrent leurs limites pour les grandes bases de données, exemple (base MNIST), alors que les algorithmes incrémentaux sont avantageux car ils peuvent mettre à jour le modèle appris à n'importe quel moment et suivre l'apprentissage.

L'algorithme proposé, exécuté en mode incrémental atteint la même performance en généralisation et de complexité moindre qu'en mode hors ligne. Donc l'avantage du mode incrémental est qu'il permet de réaliser l'apprentissage pour des bases de données volumineuses, en exécutant plusieurs phases d'apprentissage, dont seulement la première phase qui commence à zéro mais chacune des autres phases représente une mise à jour de la solution obtenue à la phase précédente. La solution SVM trouvée à la phase i doit être enregistrée pour pouvoir être utilisée à la phase i+1.

Conclusion générale

La réalisation d'un programme d'apprentissage par SVM se ramène à résoudre un problème d'optimisation impliquant un système de résolution dans un espace de dimension conséquente. L'utilisation de ces programmes revient surtout à sélectionner une bonne famille de fonctions noyau et à régler les paramètres de ces fonctions. Ces choix sont le plus souvent faits par une technique de validation croisée, dans laquelle on estime la performance du Système en la mesurant sur des exemples n'ayant pas été utilisés en cours d'apprentissage.

Dans ce mémoire, nous avons proposé et implémenté un nouvel algorithme basé sur l'algorithme de cauwenbergh appliqué aux concepts statiques. Du moment que les algorithmes incrémentaux sont des alternatives des algorithmes hors ligne c'est pour cette raison que nous avons commencé à présenter les algorithmes hors ligne dont la base d'apprentissage est connue d'avance c'est-à-dire avant la résolution du problème d'optimisation ensuite nous nous sommes intéressés aux algorithmes incrémentaux dont la base d'apprentissage est enrichie au fur et à mesure de la résolution du problème

Différents algorithmes (hors ligne) d'optimisation dédiés aux SVM sont disponibles parmi eux l'algorithme SMO (optimisation minimale et séquentielle du risque) qui est un algorithme simple et rapide proposé pour résoudre le problème de programmation quadratique des SVMs sans la nécessité de stocker une grande matrice en mémoire et sans une routine numérique itérative pour chaque sous-problème. Un deuxième algorithme important est celui de Joachims, nous avons choisi d'étudier et de simuler ces deux algorithmes puisqu'ils ont donné de bons résultats et sont les plus populaires, et afin de pouvoir comparer leurs performances en généralisation avec celle du nouvel algorithme incrémental proposé.

Pour la première simulation concernant une classification binaire sur différentes bases de données standards de différentes dimensions, nous pouvons dire que les deux algorithmes hors ligne ont donné de bonnes performances en généralisation avec une moindre complexité. Dans la deuxième simulation concernant la classification multi classe sur deux bases de données réelles des chiffres manuscrits, la première base (USPS) contient 7291 chiffres pour l'apprentissage et 2007 pour le test et la deuxième base (MNIST) contient 60000 chiffres pour l'apprentissage et 10000 chiffres pour le test.

Les résultats obtenus sont satisfaisants pour la première base (USPS), que ce soit du point de vue performance ou complexité, mais les deux algorithmes montrent leurs limites en les testant à la base de données (MNIST). Concernant la dernière simulation effectuée sur des paires de chiffres manuscrits les plus confondus, les deux algorithmes ont donné des résultats meilleurs que pour ceux de multi classe.

Pour les algorithmes incrémentaux, nous avons comparé les résultats entre les deux algorithmes (C & P avec le nouvel algorithme). Ils ont été testés sur une base de données artificielle de dimension 2, dont les échantillons sont tirés d'une manière aléatoire selon une distribution gaussienne. la base d'apprentissage est de 30 échantillons, et la base de test contient 200 échantillons. nous avons choisi que 30 échantillons dont l'objectif est d'avoir des figures non encombrées afin de faciliter l'interprétation des résultats.

Une deuxième expérimentation est faite sur les chiffres manuscrits pour trois paires de classes extraites à partir de la base de données USPS. Le noyau qui a donné de bons résultats en performance de généralisation c'est le noyau polynomial.

A travers ces expérimentations, Nous pouvons dire que notre algorithme incrémental a donné de bonnes performances en généralisation aussi bien que l'algorithme incrémental de référence (C&P), et une complexité calculatoire moindre.

Si nous le comparons avec un algorithme de référence hors ligne tel que SMO de LIBSVM, nous pouvons dire que les performances en généralisation sont très proches, mais l'algorithme SMO a la meilleure complexité calculatoire, que nous la voyons logique puisque tous les échantillons de l'apprentissage sont totalement chargés en mémoire.

Nous pouvons mentionner aussi à travers ces expérimentations que la phase d'entrainement du classificateur a pris beaucoup de temps dû à la sélection manuelle des paramètres. Le paramètre de régularisation C n'a aucun effet si les données sont linéairement séparables, c'est-à-dire il ya absence totale des exemples bruités. Dans le cas contraire le test de différentes valeurs de C est envisagé.

Pour les hyper paramètres des fonctions noyaux tel que le degré pour un noyau polynomial et sigma pour le noyau gaussien (RBF), nous devons ajuster à chaque fois un couple de paramètres (C et d) pour le noyau polynomial et (C et sigma) pour le noyau RBF.

Finalement , on peut dire que le principal inconvénient des systèmes classiques (hors ligne) est qu'ils sont a priori appris sur un groupe de données spécifiques et mis en œuvre pour fonctionner sans modifier leurs structures lors de l'utilisation. Contrairement aux algorithmes incrémentaux qui peuvent être appliqués pour les systèmes Hors ligne ou adaptatifs, comme ils peuvent être appliqués aux systèmes évolutifs qui peuvent être mis à jour au cours de leurs utilisations.

Les différentes pistes explorées pendant ce travail nous ont amenées à envisager de nombreuses perspectives. Nous présentons ici celles qui nous paraissent les plus prometteuses

1/ La sélection automatique des paramètres des noyaux est envisagée. Le paramètre de régularisation C pour un noyau linéaire, et le couple (C, sigma) pour un noyau RBF et le couple (C, d) pour un noyau polynomial.

2/ L'un des axes de recherche actuel est de parvenir à coder des connaissances *a priori* dans ces systèmes, c'est-à-dire à mieux comprendre le rôle des fonctions noyau.

3/ L'extension la plus immédiate de notre méthode d'apprentissage incrémental pour les machines à vecteurs supports est celle lui permettant d'aborder des problèmes de modélisation de concepts évoluant dans le temps. L'implémentation d'un processus d'oubli fera ainsi l'objet de prochains développements.

La notion de marge a renouvelé la vision des méthodes inductives en général, et a stimulé tout un courant de recherche dont on peut espérer qu'il débouchera sur de nouvelles classes de méthodes, encore mieux comprises et encore plus adaptables à nos problèmes.

Finalement, on espère que ce modeste travail a donné un aperçu sur ce domaine de recherche très actif, intéressant, et défiant.

Bibliographie

[1] S.Khellat Kihel, « Les séparateurs à vaste marge Bi- classe »,Université des sciences et de technologie d'Oran, exposé de Master2, 2012

[2] E. Osuna, R. Freund, and F. Girosi. " A Improved Training Algorithm For Support Vector Machines" ,Proc. 1997 IEEE Workshop on neural network for signal Proceesing , pp 276-285, 1997

[3] J.C. Platt. " Fast Training of Support Vector Machines Using Sequential Minimal Optimization ". In B. Scholkopf, C. Burges, and A. Smola, editors, Advances in Kernel Methods - Support Vector Learning, Cambridge MA: MIT Press ,1998 pp 185-208.

[4] J. Callut, «Implémentation efficace des Support vector Machines pour la classification » Mémoire présenté en vue de l'obtention du grade de Maître en informatique. Université libre de Bruxelles . département informatique, 2003

[5] A. Cornuéjols, « Une nouvelle méthode d'apprentissage : Les SVM. Séparateurs à Vaste marge », bulletin de l'AFIA, N° 51, Université de Paris-Sud, Orsay, France, Juin 2002.

[6] P.Vincent, « Modèles à noyaux à structure locale », Thèse de Phd en informatique , Université de Montréal,2003

[7] E. Osuna, R. Freund, F. Girosi. "Training support vector machines : an application To face detection", In proceedings of CVPR , june 17-19 ,1997

[8] C.C. Chang , C.J Lin. "LIBSVM : A library for support vector machines". ACM Transactions on Intelligent Systems and Technology, 2 :27 :1–27 :27, 2011. Software Disponible à http://www.csie.ntu.edu.tw/~cjlin/libsvm

[9] G.Cauwenbergh , T.Poggio , " Incremental and decremental Support Vector machine Learning", In Adv neural information processing ,volume 13.MIT press, 2001

[10] G. Fung , O.L. Mangasarian , "Incremental Support Vector Machine Classification» In Proceedings of the second SIAM international conference on data mining, pages 247– 260.Citeseer, 2002.

[11] Y. Guermeur, « Svm multi classes, théorie et applications ». Mémoire présenté pour l'obtention de l'habilitation à diriger des recherches, Université Nancy I,France,2007.

[12] T. Joachims , Svm light. Software disponible à http://svmlight.joachims.org/.

[13] T. Joachims. "Text categorization with support vector machines : Learning with many relevant features". Technical Report LS VIII, University Dortmund, Germany, 1997.

[14] G. Loosli, S. Canu, L. Bottou. , « Svm et apprentissage des très grandes bases de données » . Conférence sur l'apprentissage, Tregastel, France, 22-24 May 2006.

[15] J. Shawe-Taylor, N. Cristianini. "Kernel methods for pattern analysis". Cambridge University Press, 2004.

[16] A. Shigeo. "Support Vector Machines for Pattern Classification". Springer-Verlag London Limited, 2005.

[17] A. Almaksour ,E.Anquetil, « Apprentissage incrémental en-ligne pour des problèmes de classification évolutifs" , 2011

[18] M.Maloof, S.Michalski, " Incremental learning with partial instance memory" Artif. Intell., 154(1-2) :95–126,2004

[19] G.dreyfuset, J.Martinez, M.Samuelide, M.B .cordon, F.Badran, S.Thiria , « Apprentissage statistique, Réseaux de neurones, cartes topologiques, Machines à vecteurs supports », Edition Eyrolles,2008

[20] N. Ayat, « Sélection de modèle automatique des machines à vecteurs de support , application à la reconnaissance d'images de chiffres manuscrits ». Thèse présentée à l'école de technologie supérieure, Montréal, Canada, 2004

[21] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin , "A Practical Guide to Support Vector Classification , Rapport technique, National Taiwan University 2009

[22] A. Cornuéjols, L. Miclet, Y.Kodratoff, « Apprentissage Artificiel, Concepts et algorithmes » ISBN 2-212-11020-0 , 2002.

[23] G. Loosli, S. Canu, S.V.N. Vishwanathan, A. Smola, M. Chattopadhyay, « Boîte à outils svm simple et rapide ». Revue d'intelligence artificielle, 19, 2005.

[24] R. Fernandez, " Machines de Support pour la reconnaissance de formes : propriétés et applications " , Thèse de l'Université Paris 13, Juin 1999.

[25] O. L. Mangasarian and David R. Musicant , "Lagrangian support vector machines". Journal of Machine Learning Research, 1 :pages 161–177, 2001.

[26] M.Ludovic , « Les machines à vecteurs support pour la classification en imagerie Hyperspectrale , implémentation et mise en œuvre », rapport d'épreuve test du conservatoire national des arts et métiers, Grenoble , 2010.

[27] C.Domeniconi , D.Gunopulos , « Incremental support vector machine construction » Proceedings in ICDM , international conference on data mining, university of california, 2001

[28] B. Scholkopf, C. Burges, and A. Smola , "Introduction to support vector learning" In B. Scholkopf, C. Burges, and A. Smola, editors, Advances in Kernel Methods - Support Vector Learning, chapter 1. 1999.

[29] Thomas G. Dietterich and G.Bakiri. “Solving multiclass learning problems via error-correcting output codes”. Journal of Artificial Intelligence Research, 2 :pages 263–286, 1995.

[30] V.N. Vapnik , “Estimation of Dependences Based on Empirical Data”. springer Verlag, Berlin, 1982.

[31] V. Vapnik, “The Nature of Statistical Learning Theory”. Springer Verlag, New York, USA, 1995.

[32] V. Vapnik , “Statistical Learning Theory”. New York , USA, 1998.

[33] B. Scholkopf and A. Smola, “Leaning with Kernels”. MIT Press, 2001.

[34] N. Cristianini and J. Shawe-Taylor, “Introduction to Support Vector Machines”. Cambridge Univeristy Press, 2000.

[35] C.-W. Hsu and C.-J. Lin , “A comparison of methods for multi-class support vector Machines ». IEEE Transactions on Neural Networks, 13:415-425, 2002.

[36] J. Weston and C. Watkins , “MultiClass Support Vector Machines”. In M. Verleysen, editor Proceedings of ESANN99, Brussels 1999. D. Facto Press.

[37] T. Joachims. “Making large-scale SVM learning practical”. In Advances in Kernel Methods – Support Vector Learning, pages 169-184. MIT Press, 1999.

[38] E. Osuna, R. Freund, and F. Girosi, “Support vector machines : Training and Applications”, Technical Report AI-Memo 1602, M.I.T Artificial Intelligence Laboratory, March 1997.

[39] Gill, Murray, and Wright. “Practical optimization”, 1981

[40] Ginny Mak , “The implementation of support vector machines using the sequential minimal optimization algorithm », A master’s project submitted in partial fulfillment of requirements for the master of science degree, Mc Gill university, Montreal , Canada,2000

[41] S. Knerr, L. Personnaz, J. Dreyfus, “single-layer learning revisited: a stepwise procedure for building and training a neural network”, Neurocomputing NATO ASI series , Vol 68 , pages 41-50 , 1990.

[42] H. Majdoulayne , « Extraction de caractéristiques de texture pour la classification d’images satellites » THÈSE En vue de l’obtention du doctorat de l’université de Toulouse , 2009

[43] M.Jonathan , « Contribution à l'intégration des machines à vecteurs de supports au sein des systèmes de reconnaissance de formes : Application à la lecture automatique de l'écriture manuscrite » , Thèse de doctorat Montréal Canada 2007

[44] F. Lauer , « Machines à Vecteurs de Support et Identification de Systèmes Hybrides » , Thèse de Doctorat de l'Université Henri Poincaré – Nancy 1, 2008

[45] J. Kharroubi , « Etude de Techniques de Classement , Machines à Vecteurs Supports pour la Vérification Automatique du Locuteur », Thèse Présentée pour obtenir le grade de docteur de l'Ecole nationale Supérieure des télécommunications, 2002

[46] P. Mahé , « Noyaux pour graphes et Support Vector Machines pour le criblage virtuel de molécules » , rapport de stage DEA MVA 2002/2003,
URL : cbio.ensmp.fr/~pmahe/doc/rapportDEA-PM.pdf

[47] G. Lebrun **, «** Sélection de modèles pour la classification supervisée avec des SVM(Séparateurs à vaste marge).Application en traitement et analyse d'images », Thèse de doctorat de l'université de CAEN , 2006

[48] G. Zoutendjik. "Methods of feasible directions : a study in linear and non-linear Programming", 1970

[49] B.A. Murtagh and M.A. Saunders, "Large-scale linearly constrained optimization. Mathematical Programming", 14(1) :41–72, 1978

[50] R.J. Vanderbei. Loqo , "An interior point code for quadratic programming. Optimization methods and software, 11(1) :451–484, 1999.

[51] R. JALAM , « Apprentissage automatique et catégorisation de textes multilingues » thèse pour obtenir le grade de Docteur en Informatique univ Lumière LYON2 , 2003

[52] M. Nathalie , « Eléments d'apprentissage en statistique fonctionnelle, Classification et Régression fonctionnelles par réseaux de neurones et Support Vector Machine » Thèse pour obtenir le grade de docteur de l'Université Toulouse II,

[53] L. Ralaivola , « Modélisation et apprentissage de systèmes et de concepts Dynamiques ». Thèse de doctorat de l'université de Paris 6. 2003

[54] G. Loosli , « Méthodes à noyaux pour la détection de contexte » , Thèse de doctorat pour obtenir le grade de Docteur de l'Institut National des Sciences Appliquées de Rouen , 2005

[55] C. Burges, « A tutorial On support Vector machine for pattern recognition ». Data mining and knowledge discovery,2(2):955-974,1998

[56] H.Mohamadally et B.Fomani , " SVM : Machines à Vecteurs de Support ou Séparateurs à Vastes Marges" Versailles St Quentin, France . janvier 2006.

[57] A.Djeffal , « Utilisation des méthodes Support Vector Machine (SVM) dans l'analyse Des bases de données » , thèse de doctorat en science université de Biskra,2010

[58] B. Boser, I. Guyon, and V. Vapnik. “A Training Algorithm For Optimal Margin Classifiers” ,In Fifth Annual Workshop on Computational Learning Theory, Pittsburg,1992.

[59] A.Velachos , “Active learning with support vector machine”, Master of science school Of informatics university of Edinburgh , 2004.

[60] A. Ben Ishak, « Sélection de variables par les machines à vecteurs supports pour la discrimination binaire et multi classe en grande dimension », Thèse PhD, Université de Tunis, 2007.

[61] A. Bordes , "New Algorithms for large scale support vector Machines » Thèse de doctorat de l’université Paris VI, Pierre et Marie Curie, 2010.

[62] L.Wang (Ed.). “Support Vector Machines : Theory and Applications”. Springer-Verlag Berlin Heidelberg, 2005.

[63] A.J. Smola, P.L. Bartlett, B.Scholkopf, and D.Schuurman , “Advances in Large Margin Classifiers” , MIT Press, 2000.

[64] I. Steinwart and A. Christmann , “Support Vector Machines”, Springer, 2008.

[65] C . Cortes and V. Vapnik, “Support-vector networks. Machine Learning”, 1995.

[66] T. Friess et al, “ The Kernel Adatran Algorithm,a fast and simple learning Procedure for support vector machine “, in proceedings of the 15th international Conference on machine learning (ICML 98),Morgan Kaufmann, 1998.

[67] L.Ralaivola , F.d’Alché-Buc, « Apprentissage incrémental pour les SVM : une approche locale », Actes de la conférence francophone, Saint-Etienne , 2001

[68] N.A. Syed, H.Liu and K.K.Sung, “Incremental learning with support vector machines” In proc. Int. Joint Conf. on Artificial intelligence (IJCAI-99), 1999

[69] J. Friedman,Trevor Hastie, Robert Tibshirani , “The Elements of Statistical Learning , Data Mining, Inference and Prediction”, Preface to the Second Edition September 30, 2008

[70] G. Ratzer, H. Vangheluwe , “The Implementation of Support Vector Machines Using The Sequential Minimal Optimization Algorithm”, School of Computer Science, McGill University, Montreal, Canada, April 2006.

[71] C. Campbell, “An Introduction to Kernel methods”, Departement of Engineering Mathematics, Bristol University, Bristol BS8 1TR, United Kingdom.

[72] MIRTA B. GORDON , « Mémoire et apprentissage dans les systèmes artificiels » Master de Sciences Cognitives de Grenoble 2003-2004

[73] B. PHILIPPE , « Apprentissage Statistique & Data mining », Institut national des sciences appliquées (INSA) de toulouse,Version Octobre 2006

[74] J. Friedman, T. Hastie , R .Tibshirani , "The Elements of Statistical Learning , Datamining , inference and prediction ", preface to the second edition,2008

[75] Z.Zidelmal, « Application des SVMs basés sur l'algorithme SMO pour la détection d'anomalies cardiaques » , thèse de doctorat , université Mouloud Mammeri Tiziouzou Algérie, 2007

[76] N. Saunier , « Apprentissage incrémental par sélection de données dans un flux pour une application de sécurité routière », CAP, 2004

[77] S.Boyd , L.Vanderbergh , "Convex Optimization" , Cambridge university press March 2004

[78] O. Bousquet , « Introduction aux Support Vector Machines (SVM) », Centre de Mathématiques Appliquées, Ecole Polytechnique, Palaisea Orsay, 15 Novembre 2001

[79] http://docs.happycoders.org/orgadoc/artificial_intelligence/classification_documents /Classification.pdf

[80] N. Cristianini , « Support Vector and Kernel Machines », BIOwulf Technologies, 2001

[81] Christophe G, Nicolas M, Slimane M, Hubert C , « Introduction aux Machines à Vecteurs Support Application à la classification ». Laboratoire d'informatique de l'Université de Tours.

[82] R. O. Duda and P. E. Hart , "Pattern Classification and Scene Analysis". John Wiley & Sons, 1973.

[83] M. Minoux , « Programmation mathématique. Théorie et algorithmes. Tome 1 ». Dunod, 1983

[84] B.Hahn , T.Valentine, "ESSENTIAL MATLAB For Engenieers and Scientists", Third edition , 2007.

[85] Y. Zhan and D. Shen , "Design efficient support vector machine for fast classification.Pattern Recognition », 38(1) :157–161, 2005.

[86] M. Ramona , « Classification automatique de flux radiophoniques par Machines à Vecteurs de Support » , Thèse présentée pour obtenir le grade de docteur de l'Ecole Télécom ParisTech ,2010

[87] J.C. Platt, N. Cristianini, and J. Shawe-Taylor. "Advances in Neural Information Processing Systems", volume 12, chapter Large margin DAGs for multiclass classification, pages 547–553. MIT Press, 2000.

[88] F. Takahashi , S. Abe, "Decision-tree-based multiclass support vector machines. In Neural Information Processing, 2002. ICONIP'02. Proceedings of the 9th International Conference on, volume 3, pages 1418–1422. IEEE, 2002.

[89] L. Hamel , "Knowledge discovery with support vector machines". Wiley Edition, 2009.

[90] M.Feuilloy. « Etude d'algorithmes d'apprentissage artificiel pour la prédiction de la SYNCOPE chez l'homme » , thèse de doctorat, 2009

[91] P. Ciarlet & Hasnaa Zidani , Optimisation quadratique, Cours AO101, Ecole nationale des techniques avancées(ENSTA) , 2006

[92] C.Guéret & N.Monmarché,M.Slimane,H.Cardot. '' Introduction aux Machines à Vecteurs Support '', rapport de stage : Laboratoire d'informatique de l'université de Tours (EA2011)

[93] M.Bouillon : « Apprentissage incrémental et décrémental « , Classification avec un système d'inférence floue évolutif appliquée à la reconnaissance de gestes manuscrits. Mémoire de Master2 , université INSA DE RENNE, 2012