**REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE**

Ministère de l'enseignement supérieur et de la recherche scientifique

Université Abderrahmane Mira de Bejaïa

**FACULTE DES SCIENCES ET DES SCIENCES DE L'INGENIEUR**

Département d'informatique
École doctorale réseaux et systèmes distribués


# Mémoire de Magister en Informatique

Option
**Réseaux et Systèmes Distribués**

Thème :

# UTILISATION DES METHODES SUPPORT VECTEUR MACHINE (SVM) DANS L'ANALYSE DES BASES DE DONNEES

Directeur du mémoire : Pr. TALEB-AHMED Abdelmalik

Présenté par : **DJADEL Hacene**

Devant le jury composé de :

| | |
|---|---|
| Président | DAHMANI Abdennasser, Professeur, Université Abderrahmane Mira, Béjaïa, Algérie. |
| Rapporteur | TALEB-AHMED Abdelmalik, Professeur, Université de Valenciennes et du Hainaut Cambrésis, Valenciennes, France. |
| Examinateur | CHIKH Azzeddine, Maître de Conférences, Université Abou Bekr Belkaid, Tlemcen, Algérie. |
| Examinateur | BENAMRANE Nacera, Maître de Conférences, Université des Sciences et de la Technologie d'Oran, Algérie. |
| Invité | TARI Abdelkamel, Chargé de cours, Université Abderrahmane Mira, Béjaïa, Algérie. |

Promotion 2005/2006

# *Dédicaces*

*A ma mère et à mon père*
*A mes grands parents*
*A mes frères*
*A toute ma famille*
*A tous mes amis*

# *Remerciements*

*Je tiens à remercier en premier lieu, mon promoteur, le professeur TALEB-AHMED Abdelmalik de l'Université de Valenciennes et du Hainaut Cambrésis (France) pour m'avoir ouvert les portes de la recherche scientifique. Je le remercie également pour sa patience, sa disponibilité, ses conseils et ses encouragements.*

*Mes remerciements vont également aux membres du jury qui me font l'honneur de juger mon travail. Je les remercie d'avances pour leurs critiques et suggestions.*

*Je remercie monsieur TARI Abdelkamel, chef du département d'informatique, pour son dévouement quotidien à l'école doctorale.*

*Je remercie également tous les responsables et enseignants qui ont œuvré pour la réussite de l'école doctorale.*

*Mes remerciements vont aussi à tous mes collègues et amis avec lesquels j'ai passé des moments inoubliables. Je les remercie pour tous les échanges scientifiques, culturels et amicaux qu'on a pu partager.*

**TABLE DES MATIERES**

# Introduction générale

Le data mining est une discipline qui a déjà fait ses preuves. De nos jours le besoin en informations des entreprises ne cesse de s'accroître. L'entreprise moderne doit faire face à des clients peu fortunés voir pauvres mais aussi très exigeants et très capricieux à la fois. Le client de l'entreprise n'est plus dans la ville voisine à quelques kilomètres mais il est à des milliers de kilomètres dans un autre continent. Comme ça a toujours était le cas, l'entreprise doit connaître ses clients mais là, les métriques ont changé. Les méthodes traditionnelles statistiques d'étude de marchés ne sont plus adéquates. L'entreprise doit savoir cibler ses clients, proposer les bons produits au bon client. A cause de l'ouverture des marchés et de la concurrence mondiale, l'entreprise doit toujours chercher le meilleur rapport qualité/prix. Pour cela l'entreprise n'a souvent pas d'autre choix que de faire des bénéfices en augmentant la quantité de produits vendus.

Face à toutes ces contraintes, le chef d'entreprise doit pouvoir prendre les meilleures décisions dans les meilleurs délais. Pour cela, le data mining constitue un très bon allié du chef d'entreprise. En effet cet outil donne souvent des indications et des estimations décisives. Le data mining a déjà fait ses preuves dans beaucoup de domaines. Les assurances, les banques, détection de fraudes,…etc.

Le data mining ne se limite pas qu'au domaine financier et de l'entreprise. Car ce dernier n'est qu'un exemple des champs d'application de cet outil. D'autres domaines aussi sont soumis à des contraintes similaires. On peut citer l'industrie, la prospection pétrolière et minière, le médicale, la génomique, ….etc. En somme, le data mining peut être profitable dans n'importe quel domaine pourvue qu'on dispose de données relatives à ce domaine.

Dans ce mémoire, nous allons essayer de présenter quelques notions du data mining. Dans le premier chapitre, nous allons voir quelques notions fondamentales sur l'apprentissage automatique. Dans le deuxième chapitre, nous allons présenter quelques grandes classes de data mining à savoir la classification, le clustering et en fin les règles d'association. Dans le troisième chapitre, nous allons présenter une des dernière approches de l'apprentissage automatique qui est : SVM, abréviation de Support Vector Machine, traduit en français par machine à vecteurs de support ou encore, machine à vaste marge. L'origine de cette méthode d'apprentissage automatique remonte à 1979 mais ce n'est que vers 1995 que le monde scientifique a commencé à s'intéresser à elle. Cette méthode a la particularité de rompre avec les méthodes classiques d'apprentissage (minimisation du risque empirique, théorie de la régularisation) et d'utiliser une nouvelle approche née avec elle qui est l'approche de la minimisation du risque structurel. Dans le quatrième chapitre, nous allons présenter quelques algorithmes d'apprentissage destinés au SVM. En fin dans le dernier chapitre, nous allons présenter une solution pour un nouvel algorithme pour les SVM adaptés au data mining.

# Chapitre 1 : Présentation Générale du Data Mining

## I-1) Introduction

Le Data Mining, forage de données en français, est né du croisement de l'IA (Intelligence Artificielle) et de la statistique. Il est difficile de dire à quel époque on a commencé à faire du data mining (1970). Ce n'est que vers les années 1990 que les concepts on été formalisés et le terme data mining a été adopté.

Beaucoup d'institutions possèdent des quantités de données colossales (les banques, les entreprises,…). Il était clair que ces données renfermaient beaucoup d'informations mais comment trouver ces informations. La réponse fut le data mining avec un slogan très attractif: « trouver du diamant dans une mine de charbon ». En effet le data mining permet de trouver de l'information utile et utilisable dans des amas de données inexploitables à cause de leurs tailles. De plus le data mining découvre, proprement dit, l'information et ne se limite pas à de simples requêtes de recherche d'informations.

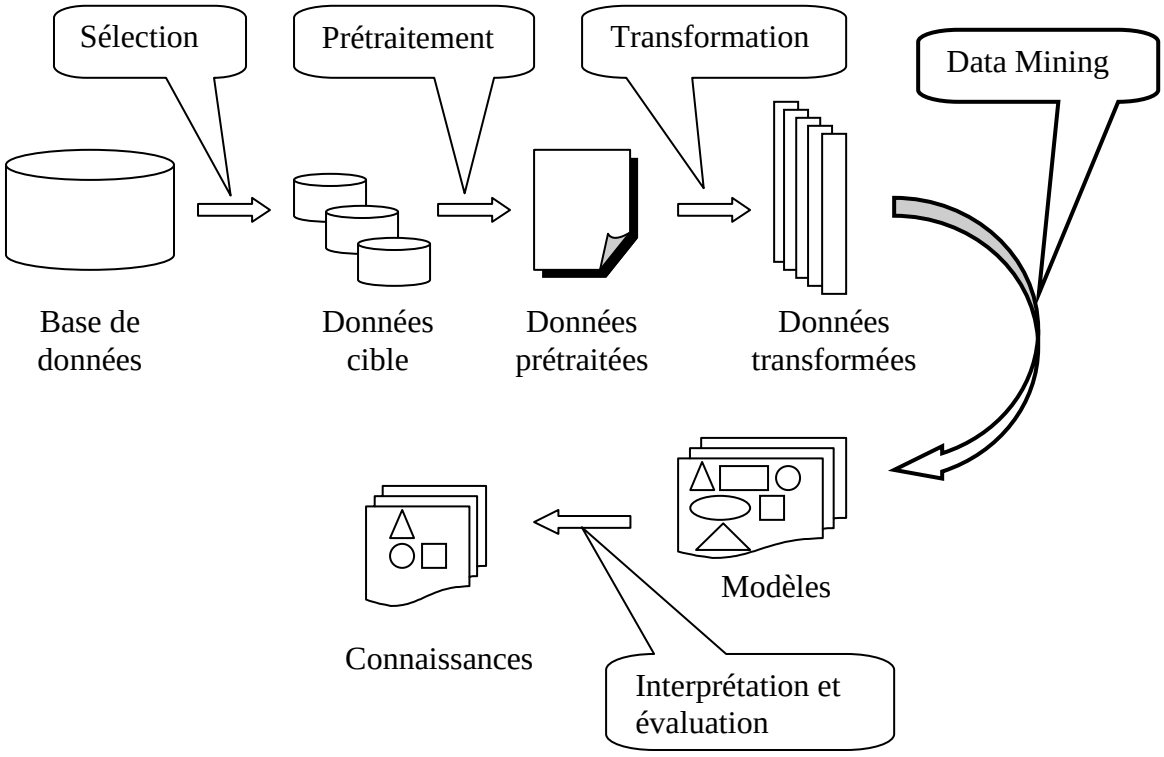



**fig. 1.1:** Les étapes du KDD.

Le data mining est une phase faisant partie d'un processus plus général appelé KDD (Knowlegde Discovery in Databases), appelé en français, ECD (Extraction de Connaissances à partir des Données). La figure 1.1 montre les principales phases du KDD **[1]**. Le point de départ du KDD est la base de données. Une première phase, dite de sélection, permet d'extraire les données nécessaires au processus. Vient en suite, la phase de prétraitement. Cette phase permet d'éliminer le bruit (gérer les données manquantes, erronées, inexactes, imprécises, conflictuelles et exceptionnelles ; résoudre l'ambiguïté). Dans la phase suivante, les données sont converties en une forme adéquate à l'analyse. Juste après, vient l'étape de data mining. Les sorties de cette phase sont des modèles (classificateur, régresseur,…). Tous les modèles obtenus ne sont pas tous opportuns. Pour ne garder que les modèles intéressants, une phase d'interprétation et d'évaluation termine tous le processus de KDD.

Les domaines d'application du data mining sont divers. Pratiquement tous les domaines où l'on dispose de grandes quantités d'informations sont atteints. Le tableau ci dessous site quelque uns des domaines et les problèmes traités dans ces domaines par le data mining.

| Domaines | problèmes |
|---|---|
| Médecine | découvrir les effets secondaires de médicament, identification de séquences génétiques,… |
| Finance | accord de crédit, prédiction de faillite, analyse de stock,… |
| Sciences sociales | analyse de données démographiques, prédiction de résultats d'élections, prédiction des tendances politiques,… |
| Astronomie | analyse d'images satellitaires, |
| Justice | détection de fraudes, identification de véhicules volés, reconnaissance d'empruntes,… |
| Marketing | prédiction de ventes, identification de consommateurs et groupes de produits, proposition d'offres spéciales |
| Science de l'ingénieur | découverte de modèles (pattern) dans les circuit VLSI, prédiction de défaillance de composant,… |
| Assurances | détection de fraudes à l'assurance, détection de profiles à haut risques pour tarification des assurés, |
| Agriculture | maîtrise de l'influence du micro climat et du sol sur les plantes |
| Publication | découverte de profiles de lecteurs pour des éditions spéciales de journaux ou de magazines |

Dans la suite, nous allons présenter des notions sur l'apprentissage automatique puis les grandes classes de méthodes de data mining.

## I- 2) Apprentissage automatique :

Apprentissage automatique ou artificiel désigne le processus qui permet à une entité artificielle d'acquérir un comportement souhaité, face à certain stimuli de manière

indépendante (autonome) ou presque. L'avantage d'un tel processus est qu'il permet de construire des entités adaptées à des problèmes particuliers de manière automatique et donc sans aucun effort humain à fournir. Deux acteurs majeurs entrent dans ce processus, le système apprenant et la méthode d'apprentissage. Ces deux parties sont très dépendantes l'une de l'autre. Le processus d'apprentissage artificiel est largement inspiré de la nature. Le processus suit un cycle apprendre essayer, jusqu'à ce que l'entité apprenante (ou élève) ait de bon résultats. L'apprentissage se fait avec un ensemble de cas ou de situation (stimulus) faisant partie de l'environnement dans lequel le système élève devrait évoluer (exister). Dans le jargon on appelle les stimuli utilisés dans le processus d'apprentissage, exemples d'apprentissage.

Si on s'intéresse à la forme des exemples d'apprentissage, on peut distinguer trois types d'apprentissage automatique.

## I-2-a) Apprentissage supervisé

Dans ce type d'apprentissage, on fournit au système un ensemble d'exemples. Chaque exemple est décrit par un vecteur **v** (au sens mathématique). Chaque coordonné du vecteur décrit une des caractéristiques de l'exemple. En plus du vecteur, chaque exemple est étiqueté par un autre attribut **y**. Le processus d'apprentissage consiste à trouver le lien qu'il y a entre le vecteur **v** et l'étiquette **y**. Une fois l'apprentissage terminé, le système est censé déduire l'étiquette **y** uniquement à partir de la valeur du vecteur **v** et ce pour n'importe quel exemple, et même pour des exemples qui n'ont pas fait partie de l'ensemble d'apprentissage. On trouve dans cette catégorie, la classification **[38] [41]**, la régression **[38] [41]**,…

## I-2-b) Apprentissage non supervisé

Ici aussi, on fournit au système un ensemble d'exemples. Mais là, chaque exemple n'est décrit que par le vecteur **v**. Lors du processus d'apprentissage, le système est censé détecter les similitudes qu'il y a entre les exemples **[38][41]**.

## I-2-c) Apprentissage semi supervisé

Ce type d'apprentissage est un mélange des deux précédents. De la même manière que ces derniers, on fournit au système un ensemble d'exemples. Mais ici, une grande partie des données fournies ne sont décrites que par leur vecteur **v.** Seule une petite partie des données fournies sont décrites par le vecteur **v** et **y**. L'apprentissage ici se fait donc en grande partie grâce aux données non étiquetées. Les données étiquetées servent un peu comme un guide général qui se contente de donner des indications assez vagues mais très utiles **[6] [38]**.

On peut aussi classer l'apprentissage en :

## I-2-d) Apprentissage paramétrique vs apprentissage non paramétrique

Ici aussi, la différence est due au type d'informations présentées au système lors de l'apprentissage. Une méthode d'apprentissage non paramétrique est une méthode qui ne nécessite que les données (exemples) d'apprentissage. Contrairement à cela, une méthode d'apprentissage paramétrique est une méthode qui nécessite; comme son nom l'indique;

d'être paramétrée. Les paramètres permettent à la méthode de limiter son champ de recherche **[41]**. Sans cette limitation, la recherche se fera dans un espace infini et donc la méthode risque de perdre beaucoup de temps à chercher la solution là où elle n'y est pas. L'inconvénient des méthodes paramétriques est que souvent les bons paramètres ne sont obtenus que par tâtonnement.

On peut aussi classer l'apprentissage selon une autre classification. En se basant sur la nature de la tâche apprise par l'entité dite élève, on peut citer :

## I-2-e) La classification

La classification est un apprentissage de type supervisé **[18] [16]**. Lors de l'apprentissage, le système apprend à distinguer entre deux catégories d'entités ou plus. Par exemple, on fournit au système les valeurs du tableau suivant et on lui demande d'apprendre à distinguer entre les véhicules utilitaires commerciaux et les véhicules touristiques. Lorsque l'apprentissage sera terminé, il suffit de présenter au système en entrée un vecteur d'entrée (puissance en chevaux DIN, vitesse de pointe) pour qu'il nous renseigne sur la catégorie du véhicule.

| Véhicule | Type | Puissance | Vitesse de pointe |
|---|---|---|---|
| 1 | Touristique | 60 chv | 100 Km/h |
| 2 | Commercial | 120 chv | 100 Km/h |
| 3 | Touristique | 120 chv | 180 Km/h |
| 4 | Commercial | 120 chv | 100 Km/h |
| 5 | Touristique | 40 chv | 80 Km/h |

**Classificateur binaire vs Classificateur n-aire**

L'exemple qu'on a donné ci haut est un classificateur qui classifie en deux classes, on dit que c'est un classificateur binaire. On trouve aussi des classificateurs qui classifient en *n* classes, on dit que c'est un classificateur n-aire. Généralement les classificateurs n-aire sont construits à base de plusieurs classificateurs binaires.

**Classificateur correct vs classificateur complet**

Un classificateur correct est un classificateur qui garantie que sa réponse est exacte. Il est très difficile de construire de tels classificateurs pour un problème réel. Dans la majorité des applications réelles, on se contente de classificateur dont la fiabilité est quantifiée par une probabilité. Un classificateur complet est un classificateur qui peut classifier tous les objets ou entités qu'on lui présente. Pour qu'un classificateur soit 'parfait', il faut qu'il soit à la fois correct et complet.

**Exemples :**

- Classificateur A : 'tous ce qui marche est un animal le reste je ne sait pas' ce classificateur ne va pas commettre d'erreur, c'est un classificateur correct. Mais que va-t-il répondre lorsque on lui pressente un poisson ? Ce classificateur ne saura pas répondre, c'est un classificateur

non complet. Il faut remarquer que la réponse fournie dans ce dernier cas (le poisson) n'est pas fausse.
- Classificateur B : 'tous ce qui a des yeux est un animal le reste non' ce classificateur a toujours une réponse, il prend toujours une décision, il est complet. Par contre sa réponse n'est pas toujours correcte, il suffit de prendre pour exemple la taupe.

### I-2-f) Le clustering (ou segmentation)

Le clustering est un apprentissage de type non supervisé **[42]**. Ici l'objectif du système lors de l'apprentissage est de découvrir les relations de ressemblance et de dissemblance qu'il y'a entre les différentes entités du domaine d'application. Le système crée des groupes d'entités homogènes et détermine les conditions d'appartenance à chacun de ces groupes.
Pour qu'un système de classification puisse apprendre, on doit lui énumérer toutes les classes existantes avant le lancement du processus d'apprentissage. Par contre, un système de clustering n'exige pas de telles informations. Donc le clustering est une méthode moins contraignante, notamment lorsque le domaine d'application est mal défini ou inconnu. Par exemple, en utilisant le tableau précédant, on peut lancer une méthode de clustering et omettre la colonne type de véhicule. Avec un peu de chance, le système pourra créer deux catégories de véhicules, l'une contiendra les véhicules commerciaux et l'autre, les véhicules touristiques.

### I-2-g) La régression

La régression est un apprentissage de type supervisé **[13][16]**. Comme pour la classification, l'objectif de la régression est de trouver le lien entre le vecteur d'entrée **v** et la sortie **y**. Sauf qu'ici la sortie **y** est une valeur ordonnée (continue ou discrète). En reprenant le tableau précédant on peut essayer de trouver la fonction f : Type $*$ Puissance $\rightarrow$ Vitesse de pointe.
Dans la régression on distingue :

L'estimation : indépendante du temps.

La prédiction : prédit l'évolution de la sortie **y** en fonction du temps.

### I-2-h) les règles d'associations

L'objectif est de déterminer les relations d'association qui existent entre les éléments d'un ensemble donné. L'utilisation la plus connue de cette technique est faite dans le cadre de l'analyse du panier de la ménagère **[7]**.

## I-3) Le Risque

### I-3-1) le risque réel

Plus un enfant apprend, moins il commet d'erreurs. Ce principe est aussi valable dans l'apprentissage artificiel. Pour dire qu'un système a appris il faut qu'il ne fasse pas d'erreur ou qu'il en fasse le moins possible. Pour cela, lors de la phase d'apprentissage, on cherche à

minimiser le risque réel d'erreur. Ce risque est exprimé par une fonction de coût. Une fonction de coût est une fonction qui quantifie l'erreur, généralement, en mesurant la distance entre la sortie ou résultat désirée et la sortie effective du système. Pour être précis, le risque doit être calculé sur la totalité des entrées possibles du système **[13][15]**.

$$R_{reel} = \int L(f(x), y)\, dP(x,y) \qquad (1.1)$$

Où
- x est la variable d'entrée du système
- $f(x_i)$ est la sortie effective
- y la sortie désirée
- $P(x,y)$ est la probabilité conjointe de x et y
- $L$ est la fonction de coût

La fonction de coût utilisée dépend de la nature du problème en régression. Par exemple, on peut trouver

$L(a,b) = L^2(a,b) = (a - b)^2$
$L(a,b) = L^1(a,b) = |a - b|$

On peut aussi avoir en classification

$L(a,b) = 1$ si $a = b$
$0$ si $a \neq b$

## I-3-2) le risque empirique

Le problème avec le risque réel $R_{reel}$, est qu'il est souvent impossible de le calculer **[13][15][18]**. Ceci peut être dû au faite que le nombre de valeur possible de x est très grand (exemple nombre d'étoiles dans le ciel), au faite qu'obtenir une valeur de x coûte chère (état des capteur d'un réacteur nucléaire en surchauffe, catégorie d'une page web,…), ou tous simplement au faite que la variable x est un réel sur un intervalle [a, b].

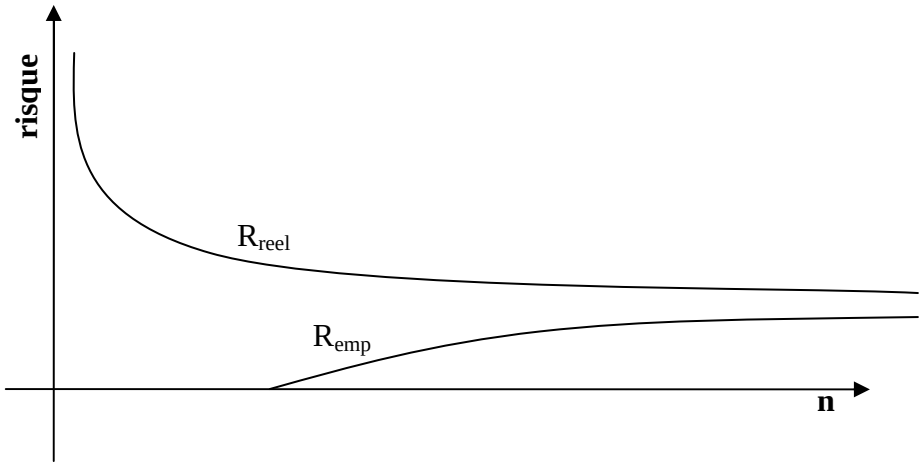



**fig. 1.2** : Risque réel et risque empirique

Pour parer à cet handicape, au lieu d'utiliser $R_{reel}$ , on utilise le risque empirique $R_{emp}$ et on essaye de le minimiser durant le processus d'apprentissage.

$$R_{emp} = \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) \qquad (1.2)$$

Selon la loi des grands nombres $R_{emp}$ se rapproche de $R_{reel}$ lorsque n tend vers l'infini. On voit sur la figure 1.2 l'évolution de $R_{emp}$ et $R_{reel}$ en fonction de *n.* Le risque $R_{reel}$ ne tend pas vers zéro car il est pratiquement impossible de trouver un système capable d'apprendre complètement la spécification d'un problème et qui ne fasse pas du tous d'erreurs. On voit aussi que le risque $R_{emp}$ est inférieur $R_{reel}$ car l'apprentissage se fait en minimisant $R_{emp}$ et non $R_{reel}$ .

Si l'ensemble d'apprentissage est assez représentatif du domaine d'application alors la minimisation du $R_{emp}$ va entraîner la minimisation du $R_{reel}$ . Si l'ensemble n'est pas assez représentatif, on risque d'exposer le système au surapprentissage. Dans ce dernier cas, le système va avoir de bonnes performances sur l'ensemble d'apprentissage mais, de mauvaises sur des exemples qui n'ont pas fait partie de l'ensemble d'apprentissage. On dit aussi que le système ne généralise pas assez. Pour éviter ce problème on utilise des techniques de validation que nous citerons plus loin.

## I-3-3) Le risque structurel

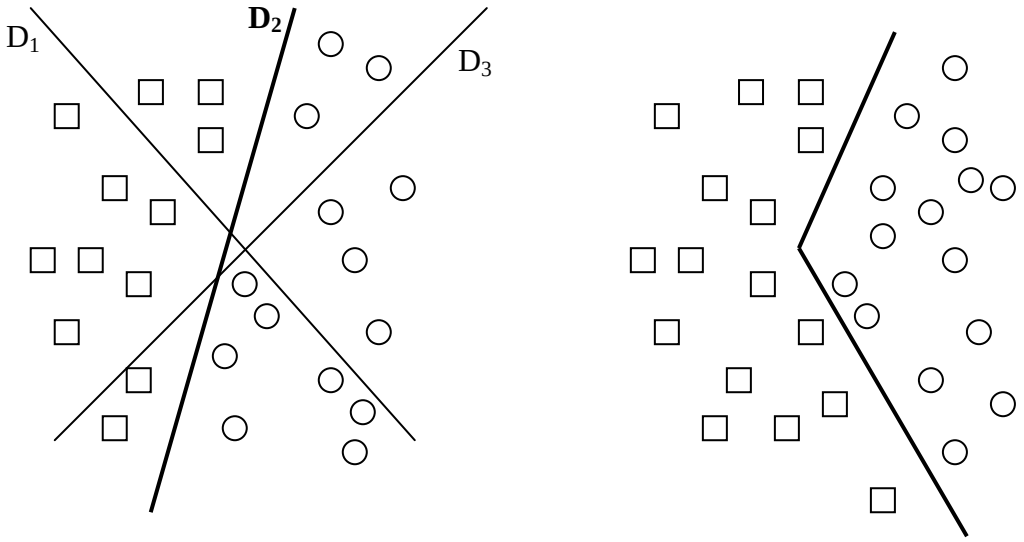
En plus du faite que le data mining et l'intelligence artificielle en général proposent beaucoup de méthodes, la plupart de ces méthodes sont paramétrables. Il arrive souvent qu'une méthode appliquée à un problème, rend de piètres résultats, tandis qu'une autre méthode excelle pour ce même problème. Il arrive encore plus souvent qu'une même méthode améliore ses résultats rien qu'en modifiant ses paramètres. Ce qui fait la différence entre toutes ces méthodes ou entre la même méthode avec des paramètres différents est l'espace d'hypothèses **[13] [15]**. Lorsque on choisit une méthode et on la paramètre, on a par cela, délimité l'espace de recherche d'hypothèses ***H*** dans lequel va se faire la recherche. Ainsi, lors du lancement de l'apprentissage proprement dit, le système recherche la meilleure ou les meilleures hypothèses qui expliquent le problème uniquement dans cette espace d'hypothèses. On comprend alors que si un espace d'hypothèse $\boldsymbol{H_0}$ ne contient pas des hypothèses proches de l'hypothèse $\boldsymbol{h_0}$ qui décrit le problème analysé, alors une recherche dans un tel espace sera en vain. Et de ce faite une méthode qui se limite à cet espace $\boldsymbol{H_0}$ lors de la phase d'apprentissage aboutira forcément à un piètre résultat.

**Exemple**

On prend un ensemble de points de deux catégories dans un espace à deux dimensions ***E***. Et on cherche à créer (faire apprendre à) un système à distinguer entre les deux catégories de points, i.e. que le système sache reconnaître la catégorie d'un point par la seule connaissance de ses coordonnées. Pour cela on fait la supposition que l'on peut séparer les deux catégories par une droite. Donc ici l'espace d'hypothèse $\boldsymbol{H_0}$ est l'ensemble des droites qui appartiennent à l'espace ***E***. La tâche du processus d'apprentissage sera de trouver la droite qui sépare les deux catégories de points (on ne tolérera aucune erreurs).

Sur la figure 1.3 (a) on voit deux droite (D1 et D3) qui séparent mal les deux catégories de points. La dernière droite (D2 ) sépare bien les points. Durant le processus d'apprentissage le système aura appris que la droite D2 sépare les deux catégories et cela seulement au travers les exemples fournis.

Sur la figure 1.3 (b), on a un autre ensemble de points. Ici, il est impossible de trouver une droite qui va séparer parfaitement les deux catégories de points. L'espace d'hypothèses $\boldsymbol{H_0}$ n'est pas adapté pour ce cas. Il faut donc un autre espace d'hypothèses. Si on suppose maintenant que les points sont séparables par une paire de demi-droites (ou une droite pliée) alors on pourra trouver une solution.




**a** : Droite **b** : Deux demi-droites

**fig. 1.3** : Exemple de séparateurs

Ici l'espace d'hypothèse $\boldsymbol{H_1}$ est l'ensemble des droites pliées en deux. On peut aussi avoir des problèmes où l'espace d'hypothèses $\boldsymbol{H_1}$ ne contiendrait pas la solution $\boldsymbol{h_0}$ souhaitée. Et il faudrait utiliser un espace d'hypothèses $\boldsymbol{H_2}$ tel que $\boldsymbol{H_2}$ est l'ensemble des droites pliées en trois (ayant deux points de discontinuité).

A partir de $\boldsymbol{H_0}$ , $\boldsymbol{H_1}$ et $\boldsymbol{H_2}$, on peut donner une forme plus générale pour un espace $\boldsymbol{H_n}$ tel que $\boldsymbol{H_n}$ est l'ensemble des droites pliées en n+1 (ayant n points de discontinuités).
On remarque déjà que le problème sur la figure 1.3 (a) peut très bien être résolu en utilisant l'un des espaces d'hypothèses $\boldsymbol{H_1}$ , $\boldsymbol{H_2}$ ,…, $\boldsymbol{H_n}$ . De même que le problème de la figure 1.3 (b) peut être résolu avec l'un des espaces d'hypothèses $\boldsymbol{H_2}$ , $\boldsymbol{H_3}$ ,…, $\boldsymbol{H_n}$. On peut conclure que si un problème accepte une solution dans un espace $\boldsymbol{H_i}$ alors il peut très bien être résolu dans un espace $\boldsymbol{H_j}$ ( $\boldsymbol{j} > \boldsymbol{i}$ ). Cette conclusion hâtive est due au faite que les espaces d'hypothèses $\boldsymbol{H_0}$ , $\boldsymbol{H_1}$ ,…, $\boldsymbol{H_n}$ sont imbriqués les uns dans les autres comme suit $\boldsymbol{H_0} \subset \boldsymbol{H_1} \subset \boldsymbol{H_2} \subset \ldots \subset \boldsymbol{H_{n-1}} \subset \boldsymbol{H_n}$. On pourrait même croire qu il suffit de prendre un espace $\boldsymbol{H_k}$ avec $\boldsymbol{k}$ le plus grand possible, et de cette manière on résoudra facilement tous les problèmes de séparation entre deux classes dans un plan à deux dimensions. Malheureusement, ceci n'est guère vrai, une telle approche va certe aboutir à des résultats rapides est corrects à cent pour cent sur les

données d'apprentissage mais dès que le jeu de données change, la qualité des résultats chute vertigineusement. Ceci est d'autant plus vrai lorsque la valeur de ***k*** (de $H_k$ ) est très grande. Ici aussi, le problème est dû au surapprentissage. En effet, plus le ***k*** est grand et donc plus l'espace d'hypothèses est grand, plus il faudrait d'exemples (points) pour avoir une bonne représentativité du problème traité. Si ***k*** tend vers l'infini alors il faudrait avoir un listing complet de tous les points du problème. Comme on l'a déjà vu, cela est impossible pour des problèmes réels dignes de ce nom. Donc pour qu'une solution ***h*** généralise bien, il faut la prendre d'un espace d'hypothèses $H_i$ dont le ***i*** est le plus petit possible. Et donc, la prendre de l'espace d'hypothèses le plus simple possible. Ce principe porte le nom de rasoir d'Ockham. En gros, ce principe dit qu'on doit se contenter de l'explication la plus simple tant qu'elle est juste. Ce principe a été concrétisé dans la théorie de la régularisation **[13][39][41]** et notamment dans les SVM avec le principe de la minimisation du risque structurel.

Sur la figure 1.4 on a un autre exemple de l'application du principe du rasoir d'Ockham, cette fois appliqué au cas de la régression. Sur cette figure, on a un ensemble de points et on cherche à tracer la courbe qui va passer par tous ces points. On voit que l'on peut avoir une infinité de tracés (1,2,3,…). En appliquant le principe du rasoir d'Ockham, on aboutira à la courbe la plus simple qui est la courbe numéro 1.

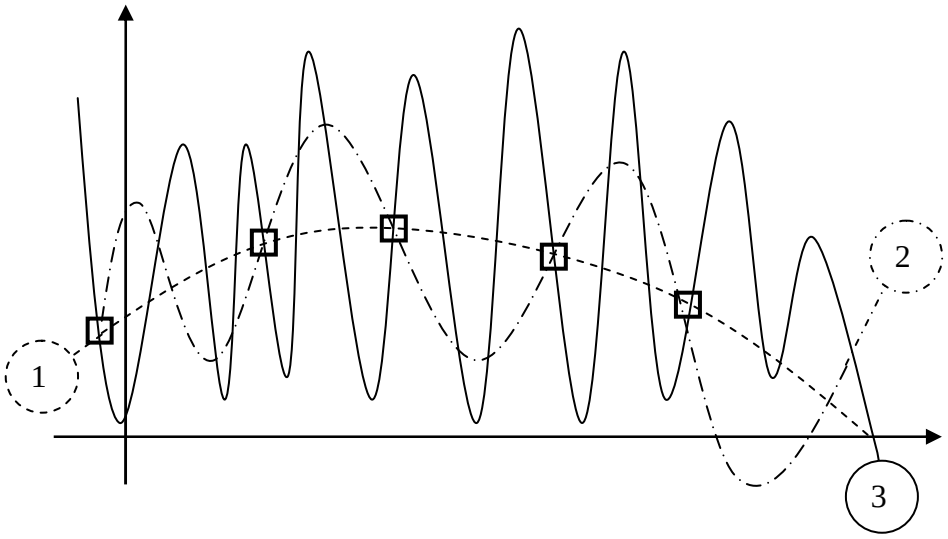



**fig. 1.4** : Rasoir d'Ockham appliqué à la régression

## I-4) Méthodes de validation :

Comme on l'a déjà vu, le problème majeur dans l'apprentissage artificiel est le surapprentissage. En effet, lorsque on lance un processus d'apprentissage, on n'a aucun moyen de savoir si l'espace d'hypothèse $H_0$ choisi est adéquat ou non. Pour répondre à cette question on utilise les techniques de validation **[39] [41]**.

On peut classer les techniques de validation en deux catégories, à savoir la validation par expertise et la validation par les statistiques.

## I-4-1) Validation statistique :

Comme son nom l'indique, cette méthode utilise les statistiques. La validation ici est simplement une phase où l'on teste la réponse du système (apprenant) sur un autre ensemble de données autre que celui utilisé pour l'apprentissage. Pour cela on doit disposer de deux ensembles de données. Un ensemble est utilisé lors de la phase d'apprentissage, et l'autre ensemble est utilisé pour la validation. Ainsi, si le système généralise bien, ses performances sur l'ensemble de validation ne devront pas varier drastiquement par rapport aux performances obtenues sur l'ensemble d'apprentissage. Généralement on construit trois ensembles au lieu de deux. A savoir **[41]** :

- ensemble d'apprentissages proprement dit ;
- ensemble de tests ;
- ensemble de validation.

Les deux premiers ensembles servent à l'apprentissage. On fait tourner l'algorithme sur le premier ensemble et on modifie les paramètres de l'algorithme jusqu'à ce qu'on obtienne un résultat satisfaisant. Ensuite, on teste le modèle sur le deuxième ensemble. Si à ce moment là, le résultat est mauvais on revient à la première phase, et on recommence le procédé. Lorsque on aura obtenu de bons résultats sur l'ensemble de tests, on fait une sorte de test ou d'examen final, qui donne une sorte de note définitive. Cette note permet notamment de comparer le modèle en cours avec d'autres modèles obtenus avec d'autres méthodes ou algorithmes.

Si on pouvait avoir un ensemble de données assez large pour être scindé en deux ou trois à chaque fois qu'on veut utiliser une méthode d'IA, alors la seule technique de validation statistique qui existerai serai celle qu'on vient de citer. Mais comme cela n'est pas le cas, les chercheurs ont développé d'autres techniques de validation statistiques qui permettent de surpasser cet handicape.

**Cross validation (validation croisée):**

Comme les données ne sont pas en quantité suffisantes, on partitionne l'ensemble des données en ***m*** partitions de tailles égales. On fait l'apprentissage avec l'échantillon constitué de (***m*** – 1) partitions, la partition restante sert d'échantillon de test. On répète cette opération ***m*** fois, en veillant à utiliser une partition différente comme échantillon de test à chaque fois. La moyenne des ***m*** tests constitue le résultat final du test **[38] [40][41]**.

**Le bootstrap :**

Cette méthode aussi est utilisée lorsque les données sont rares **[38]**. Ayant un ensemble de données de taille ***n***, l'ensemble d'apprentissage est constitué en tirant ***n*** exemples de manière aléatoire et avec remise. Ici, un exemple peut apparaître plus d'une fois dans l'ensemble d'apprentissage. L'ensemble de test est constitué des données restantes.
L'estimation de l'erreur est soit donnée par le résultat sur l'ensemble de test soit par d'autres formulations tel que :

$$\frac{1}{k}\sum_{i=1}^{k}[0.632 \times \text{test error}_i + 0.368 \times \text{repl error}_i].$$

Où $k$ est le nombre de fois que l'on répète l'échantillonnage bootstrap.

$\text{test error}_i$ est l'erreur sur le i-emme ensemble de test obtenu avec le i-emme prédicateur (classificateur, regresseur,…).

$\text{repl error}_i$ est l'erreur sur tout l'ensemble d'apprentissage obtenu avec le i-emme prédicateur.

**Leave one out :**

Cette méthode est un cas extrême de la validation croisée **[40]**. Ici on partitionne aussi les données sauf que chaque partition ne contient qu'un seul exemple. On fait donc l'apprentissage avec l'ensemble des données auquel on soustrait un exemple. Avec ce dernier on fait le test. On réitère cette opération jusqu'à ce qu'on ait testé avec tous les exemples. Le test ici consiste seulement à voire si le système n'a pas fait d'erreur sur l'exemple de test. A la fin, on calcule la moyenne de tous les tests. Cette dernière est le résultat du test. On voit ici que le coup de la validation en temps de calcul est très prohibitif mais en contrepartie on a l'avantage d'un résultat bien plus précis.

### I-4-2) Validation par expertise :

Il arrive que le domaine d'application soit trop sensible pour se fier uniquement à la validation statistique. Par exemple, imaginant un système de détection de mal formations graves chez un fœtus. Quand on sait que de telles anomalies chez les fœtus peuvent entraîner de graves décisions, on ne peut pas se permettre de valider un tel système de détection sans la consultation d'experts. L'expert analyse les rouages internes du modèle et pour qu'il puisse le faire, le modèle choisi doit obligatoirement lui être compréhensible. Il est donc primordial de choisir des modèles simples pour ce genre d'applications.

## I-5) Conclusion :

Dans ce chapitre nous avons vu des notions liées à l'apprentissage automatique. Bien évidemment, nous n'avons pas traité tous les modèles d'apprentissage automatique mais nous avons essayé de donner les rudiments nécessaires pour que le lecteur puisse bien comprendre les prochains chapitres.

# Chapitre 2 : Tour d'horizon du data mining

## Introduction

Le data mining offre beaucoup de méthodes. Il s'inspire beaucoup des acquis de l'IA et de la statistique. Dans ce chapitre nous allons essayer de présenter quelques méthodes classiques. Nous allons nous intéresser aux méthodes de classification, de clustering et aux règles d'associations. Ce chapitre ne va pas inclure toutes les méthodes car rien que l'énumération des méthodes existantes nécessitera des pages. Pour être compréhensible et utile nous allons présenter des méthodes de bases, souvent à l'origine de beaucoup de variantes.

## II-1) Méthodes de Classification :

Dans ce paragraphe nous allons voir quelques méthodes de classification. Nous allons surtout présenter brièvement le classificateur naïf de Bayes et les méthodes de plus proches voisins puis nous allons nous attarder sur les réseau de neurones et les arbres de décision.

### II-1-1) Classificateur Naïf de Bayes (Naïves Bayes Classifier)

Cette méthode utilise principalement la règle de Bayes pour classer les différents exemples. L'appellation de la méthode est due à l'utilisation d'une supposition naïve dans cette méthode que nous allons voir dans la suite. Nous rappelons ici la règle de Bayes:

$$P(Y = y_i / X = x) = \frac{P(X = x / Y = y_i) \cdot P(Y = y_i)}{\sum_j P(X = x / Y = y_j) \cdot P(Y = y_j)} \qquad (2.1)$$

On calcule cette probabilité pour tous les $y_i$ et on considère le $y_i$ qui la maximise comme étant la classe recherchée. On voit qu'il suffit de chercher le $y_i$ qui maximise

$$y = \underset{y_i}{\text{Arg Max}} \left( P(X = x / Y = y_i) \cdot P(Y = y_i) \right) \qquad (2.2)$$

Le calcul des différents $P(Y = y_i / X = x)$ exige la connaissance des termes suivant :

$$P(Y = y_i)$$
$$P(X = x / Y = y_i)$$

Pour la première probabilité, on peut l'estimer à partir des exemples d'apprentissage comme suit :

$$P(Y = y_i) = \frac{\| (x, y) \in Data / y = y_i \|}{\| Data \|} \qquad (2.3)$$

Où || E || dénote le cardinale de l'ensemble E

Pour la deuxième probabilité aussi, nous allons utiliser les exemples d'apprentissage. Seulement là, on peut avoir des problèmes. Si l'espace des exemples est très grand, il sera difficile d'avoir toutes les valeurs possibles de $X$ dans l'ensemble d'apprentissage. Et si on suppose que cela peut se faire, on aura un autre problème ; celui de stocker toutes les probabilités $P(X = x \mid Y = y_i)$ associées à chacune de ces valeurs.

Pour remédier à ce problème, la solution adoptée est la suivante :

$$\begin{aligned} P(X = x \mid Y = y_i) &= P(X = ((x^0, x^1, x^2, \ldots, x^j, \ldots, x^{n-1}, x^n) \mid Y = y_i) \\ &= P(x^0, x^1, x^2, \ldots, x^j, \ldots, x^{n-1}, x^n \mid Y = y_i) \\ &= P(x^0 \mid x^1, x^2, \ldots, x^j, \ldots, x^{n-1}, x^n, Y = y_i) \cdot \\ &\quad P(x^1 \mid x^2, \ldots, x^j, \ldots, x^{n-1}, x^n, Y = y_i) \cdot \ldots \\ &\quad P(x^{j-1} \mid x^j, \ldots, x^{n-1}, x^n, Y = y_i) \cdot \ldots \\ &\quad P(x^n \mid Y = y_i) \end{aligned}$$

Puis là, on suppose naïvement que les $x^j$ sont indépendants **[2]** et on obtient :

$$\begin{aligned} P(X = x \mid Y = y_i) &= P(x^0 \mid Y = y_i) \cdot \\ &\quad P(x^1 \mid Y = y_i) \cdot \ldots \cdot \\ &\quad P(x^{j-1} \mid Y = y_i) \cdot \ldots \cdot \\ &\quad P(x^n \mid Y = y_i) \end{aligned} \qquad (2.4)$$

A partir de là, on peut donner un algorithme d'apprentissage simplifié du classificateur naïf de bayes

**Algorithme apprentissage de Naïf Bayes**
**Debut**
Pour chaque $(x_j,y_i) \in$ Data faire
$N_i = N_i + 1$ ;
Soit $x_j = (x_j^{k0}, x_j^{k1}, x_j^{k2},\ldots, x_j^{kM-1}, x_j^{kM})$
Pour chaque k = k0 à km
$l_{ik} = l_{ik} + 1$ ;
fin
fin
Pour tous les Ni
$P(Y = y_i) = N_i / \| Data \|$ ;

Pour m=0 a M
Pour k = 0 a Km
$P(x^k \mid Y = y_i) = l_{ik} / \| N_i \|$
fin
fin
fin
**Fin**

## II-1-2) K plus proches voisins (KNN K Nearest Neighbours)

Cet algorithme présenté dans **[3]** appartient à la famille dite apprentissage basé mémoire. Son principe est très simple. Il n'y a pas de phase d'apprentissage proprement dit. Car l'algorithme ne fait que mémoriser les exemples qu'on lui fourni (exemples et leurs classes correspondantes). Dans la phase d'exploitation, lorsque on lui présente un exemple $X$ à classifier, l'algorithme cherche dans les exemples qu'il a mémorisés, les k exemples qui lui sont les plus proches. Puis parmi ces derniers, l'algorithme cherche la classe majoritaire et conclue que c'est la classe à attribuer à l'exemple $X$.
Pour la valeur de $k$, on peut la fixer au préalable. Cependant une valeur très faible de $k$, rend le classificateur très sensible au bruit et une valeur trop élevée rend le modèle moins précis. Une technique simple pour fixer $k$ est de l'obtenir en testant toutes ses valeurs possibles et en prenant celle qui minimise l'erreur.
On a dit aussi qu'un exemple $X$ est classifié avec la classe de ses $k$ plus proches voisins. Dans **[4]** l'auteur propose de pondérer les voisins avec des poids inversement proportionnels à leur distance de l'exemple $X$. Un autre paramètre important aussi de cet algorithme est le choix de la distance. Ce paramètre aussi peut être fixé par tâtonnement.
On voit sur la figure (2.1) l'importance du paramètre $k$. Si $k = 1$ alors le nouvel exemple, symbolisé par une croix, va être classé dans la classe des ronds. Si $k = 2$ alors le nouvel exemple va être classé dans la classe des étoiles. Et pour $k = 3$, le nouvel exemple va être classé dans la famille des carrés.

Un des problèmes majeurs de l'algorithme KNN est qu'il exige un temps de calcul considérable durant la phase d'exploitation. En effet, parcourir tous les exemples pour rechercher les plus proches à chaque interrogation, n'est pas très attractif. Pour y remédier, des solutions ont été proposées pour agréger les exemples soit en ne gardant que les exemples opportuns soit en reconstruisant un nouvel ensemble modèle **[5]**, à partir des exemples d'apprentissage.

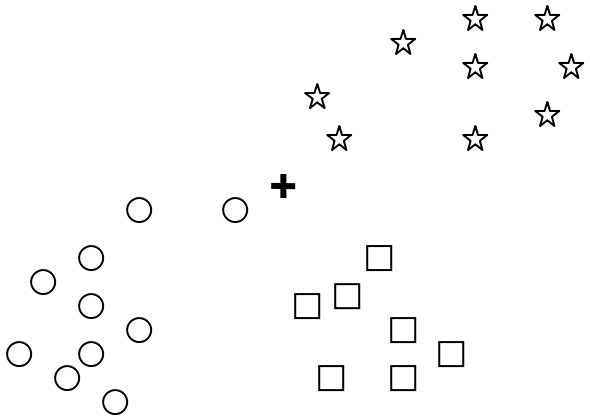


**fig. 2.1**: Influence du paramètre k dans KNN

## II-1-3) Les Réseaux de neurones :

Les réseaux de neurones artificiels ont fait leur apparition dans les années 1960 **[38]**. Ils ont été inspirés des réseaux de neurones biologiques

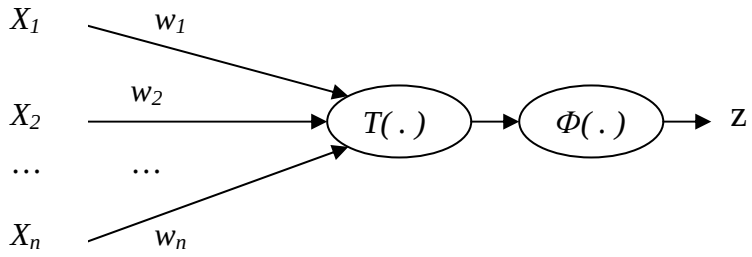



**fig. 2.2**: Schéma d'un neurone artificiel

Un neurone artificiel est caractérisé par ses poids synaptiques $w_i$ , par sa fonction de transfert $T(\ )$ et par sa fonction d'activation $\Phi(\ )$ appliquée au résultat de la fonction de transfert. Les entrées du neurone $x_1, x_2, \ldots, x_n$ forment un vecteur d'entrée $x$ qui stimule le neurone, après calcul, le neurone fourni en sortie une valeur $z = \Phi(T(w, x))$

On distingue généralement deux grandes familles de neurones, les neurones à fonction d'activation logistique et les neurones à fonction d'activation radiale.

Pour la première famille, on utilise une fonction de transfert qui calcule la somme pondérée sur les entrées du neurone

$$s = T(x) = \sum_i w_i \cdot x_i$$

Comme fonction d'activation, on utilise dans ce cas des fonctions en forme de S tel que :

- La fonction sigmoïde

$$\Phi(s) = \frac{1}{1+e^{-\lambda S}}$$

- La fonction échelon

$$\Phi(s) = \begin{array}{ll} 1 & \text{si } s \geq 0 \\ 0 & \text{si } s < 0 \end{array}$$

- La fonction tangente hyperbolique

$$\Phi(s) = TanHyp(s) = \frac{e^{s} - e^{-s}}{e^{s} + e^{-s}}$$

Pour la seconde famille, on utilise une fonction de transfert qui calcule une distance par rapport à un vecteur centre de référence C,

$$s = T(x) = (c-x)^{t} \cdot \frac{1}{\Sigma} \cdot (c-x)$$

Où $\Sigma$ est la matrice variance covariance des $x$

La sortie S est nulle lorsque $c$ et $x$ sont superposés et s'accroît avec l'accroissement de la distance. Comme fonction d'activation, on utilise ici des fonctions en forme de cloche tel que :

- La fonction gaussien : $\Phi(s) = e^{-|s|}$
- La fonction spline : $\Phi(s) = |s|$
- La fonction multiquadratique $\Phi(s) = (1+s)^{1/2}$
- La fonction inverse multiquadratique $\Phi(s) = (1+s)^{-1/2}$
- La fonction de Cauchy $\Phi(s) = (1+s)^{-1}$

Le plus souvent on utilise des fonctions qui permettent d'obtenir une non linéarité à la sortie du neurone, mais il n'est pas impossible de trouver des neurones à fonction d'activation linéaire $\Phi(s) = s$.

### II-1-3-1) Le Perceptron

Le perceptron a été proposé par Rosenblatt en 1957 **[38]**. Il est constitué d'un seul neurone avec n+1 entrées, l'une de ses entrées est toujours à 1, elle est appelée le biais. Le perceptron reçoit en entrées un vecteur de $n$ dimensions $X = (x_1, x_1, ..., x_n)$ et calcule la somme :

$$s = T(w,x) = \sum_i w_i \cdot x_i + w_0$$

La sortie du perceptron est :

$$\Phi(T(w,x)) = \begin{cases} 1 & \text{si } s > 0 \\ -1 & \text{si non.} \end{cases}$$

Le perceptron permet de construire un hyperplan qui sépare entre deux classes. L'équation de ce séparateur est

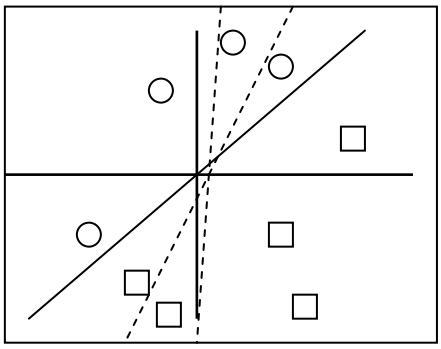
$$\sum_i w_i \cdot x_i + w_0 = 0 \qquad (2.5)$$

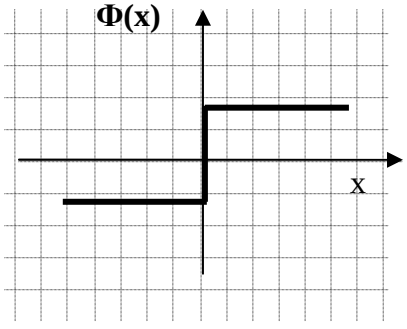
L'apprentissage du perceptron se fait avec un algorithme qui modifie les poids synaptiques jusqu'à satisfaction d'un certain critère. Sur la figure, on voit un exemple simple où l'espace d'entrées est de dimension deux. L'algorithme fait varier et l'inclinaison et la position de l'hyperplan séparateur (ici une droite).
Sur la figure on voit deux classes de points schématisées par des cercles et des carrés. L'algorithme permet de faire apprendre au perceptron à distinguer entre les deux classes, il faut noter que l'apprentissage ici est supervisé.

$\Delta = (\Delta_0, \Delta_1, \Delta_2, \ldots, \Delta_n)$
$W = (w_0, w_1, w_2, \ldots, w_n)$
$X^i = (x^i_1, x^i_2, x^i_3, \ldots, x^i_n)$
$X^i$ vecteur d'entrée
$V^i$ sortie voulue (désirée) pour $X^i$
$Y^i$ la sortie calculée par le perceptron pour $X^i$

Initialisation aléatoire de W
Tanque apprentissage non terminé
  Pour i= 1 à K
    Selectionner $X^i$
    $Y^i = \Phi(\mathbf{T}(W_t, X^i))$
    Si $V^i \neq Y^i$
      Pour j = 1 à n
        $\Delta_j = (V^i - Y^i) * x^i_j$
      $\Delta_0 = (V^i - Y^i) * 1$ ;

      $W_{t+1} = W_t + \eta * \Delta$






**La fonction Φ**

**fig. 2.3** : Le perceptron

Rosenblatt a démontré que si le problème est linéairement séparable alors l'algorithme converge. $\eta$ est un cœfficient qui permet de régler la vitesse de convergence de l'algorithme. S'il est trop grand, on risque de passer à coté de la solution et donc on risque de ne jamais trouver les bons poids. S'il est très petit, la convergence risque de prendre beaucoup de temps.

### II-1-3-2) ADALINE

ADALINE (pour ADAptive LINear Element) a été proposé par Windrow et Hoff en 1960 **[38]**. Ce modèle est une amélioration du perceptron de Rosenblatt. L'algorithme d'apprentissage suit toujours le même principe, sauf qu'ici la fonction d'activation est continue. Grâce à cela, l'erreur commise par le neurone est quantifiée et la correction apportée au poids est proportionnelle à l'ampleur de l'erreur. Cette nouvelle méthode est appelée la règle de LMS (Least Mean Square), on l'appel aussi la règle du delta ou la règle de Windrow et Hoff. L'idée est la suivante :

Si on veut construire un séparateur qui sépare au mieux les deux classes, au moins pour l'échantillon d'apprentissage, alors, il faut minimiser l'erreur globale :

$$E = \frac{1}{\mathrm{n}} \sum_i E^i \qquad (2.6)$$

$E^i = 0$ lorsque $X^i$ est bien classé,
$= (V^i - Y^i)^2$ ; si non

Où $E^i$ est l'erreur commise par le perceptron sur l'exemple $X^i$,
$Y^i = \Phi(T(W, X^i))$,
$V^i$ la sortie voulue (souhaitée) pour $X^i$

Pour cela, on calcule le gradient de $E^i$ par rapport à $W$. À chaque itération on fait une descente du gradient comme suit

$$W_{t+1} = W_t - \eta \cdot \frac{\partial E^i}{\partial W} \qquad (2.7)$$

On a:

$$\frac{\partial E^i}{\partial W} = \frac{\partial E^i}{\partial Y^i} \cdot \frac{\partial Y^i}{\partial W}$$

$$\frac{\partial E^i}{\partial Y^i} = 2 \cdot (V^i - Y^i)$$

Si on prend

$$Y^i = \Phi(T(W, X^i)) = W \cdot X^i ; \qquad (\Phi(.) \text{ fonction identité})$$

Alors

$$\frac{\partial Y^i}{\partial W} = X^i$$

Finalement on a :

$$\frac{\partial E^i}{\partial W} = 2 \cdot (V^i - Y^i) \cdot X^i \qquad (2.8)$$

Et donc, la seule modification à apporter à l'algorithme précédant, est dans le calcul du $\Delta_j$

$$\Delta_j = (V^i - Y^i) \cdot x_j^i \qquad (2.9)$$

Le perceptron de Rosenblatt résous bien les problèmes qui sont linéairement séparables. Celui de Windrow et Hoff, peut même s'étendre à d'autres problèmes en utilisant des fonctions non linéaires. Mais dés qu'on passe à des problèmes plus complexes tel que le XOR (ou exclusif) on se rend compte des limites du perceptron. Pour remédier à cela, on utilise plusieurs de ces unités pour constituer un réseau dont le pouvoir expressif est beaucoup plus grand.

### II-1-3-3) Réseau de neurones

Plus un réseau comporte de neurones, plus son pouvoir expressif augmente. Pour donner un ordre de grandeur: le cerveau d'une fourmi comporte quelques milliers de neurones, celui d'un chien quelques millions et celui d'un homme quelques milliers de milliards. Sachant cela il est tentant d'utiliser un grand nombre de neurones pour construire un équivalent au cerveau humain ou pourquoi pas plus! Malheureusement disposer d'un nombre élevé de neurones ne constitue pas pour autant une entité intelligente. Car tous ces neurones doivent être interconnectés correctement pour fonctionner en harmonie. Ceci se fait durant la phase d'apprentissage, cette phase est d'autant plus lente que le nombre de neurones est grand.

Si un neurone est caractérisé par ses poids synaptiques, sa fonction de transfert et sa fonction d'activation, un réseau de neurones lui est caractérisé par le type de ses neurones, leurs nombres et le type de maillage qui existe entre eux. On peut avoir un réseau complètement connecté où tous les neurones sont connectés les uns au autres ou bien un réseau partiellement connecté. On peut aussi avoir un réseau organisé en couches, un réseau avec ou sans rétroaction, un réseau organisé en modules. Il faut aussi savoir qu'un réseau de neurones est aussi caractérisé par son algorithme d'apprentissage.

### II-1-3-4) Le Perceptron Multi Couches (PMC) :

Depuis les années 1960 on savait qu'un réseau de neurones avait un pouvoir d'expression beaucoup plus grand que celui d'un seul neurone. L'obstacle à leur utilisation était l'inexistence de méthodes d'apprentissages adaptées.

Le perceptron multi couche est un réseau neuronal organisé en couches sans rétroaction. Il a été présenté indépendamment par Rumelhart et McClelland **[38]** , Hinton et Williams en 1986. L'algorithme d'apprentissage utilisé ici est une généralisation de l'algorithme de Windrow et Hoff précédant. Il calcule la correction à apporter au poids des neurones de la dernière couche, puis la correction de la couche d'avant et ainsi de suite jusqu'à atteindre la première couche d'où le nom de l'algorithme rétro propagation du gradient de l'erreur.
Les sorties des neurones d'un PMC ayant $C$ couches, et $N_c$ neurones dans chaque couche peuvent être formulées mathématiquement comme suit :

$$Z_i^c = \begin{cases} X_i & i = 1 \text{ à } N_0 \\ \Phi\left(\sum_{j=1}^{Nc-1} w_{i,j}^{(c)} * Z_j^{(c-1)} + w_{i,0}^{(c)}\right) & i = 1 \text{ à } Nc \end{cases} \qquad (2.10)$$

Où $w_{ij}^{(c)}$ est le poids reliant la sortie du neurone $j$ de la couche $c$-1 à l'entrée du neurone $i$ de la couche $c$.

$w_{i,0}$ est le biais du neurone $i$ de la couche $c$.

On peut réécrire la formule précédente comme suit :

$$Z_i^{(c)} = \begin{cases} X_i & i = 1 \text{ à } N_0 \\ \Phi\left(\sum_{j=0}^{Nc-1} w_{i,j}^{(c)} \cdot Z_j^{(c-1)}\right) & i = 1 \text{ à } Nc \end{cases}$$

Sachant que $Z_0^{(c)} = 1$ pour $c = 0$ à $(C - 1)$ ;

Après calcul on obtient les formules suivantes :

- $\dfrac{\partial R_{emp}}{\partial w_{ij}^{(c)}} = \delta_i^{(c)} \cdot Z_j^{(c-1)}$
- $\delta_i^{(c)} = \dfrac{\partial R_{emp}}{\partial Z_j^{(c)}} \cdot \Phi'(S_i^{(c)})$ pour $c = C$

$$= \Phi'(S_i^{(c)}) \cdot \sum_{k=0}^{Nc+1} \delta_k^{(c+1)} \cdot w_{k,i}^{(c+1)} \quad \text{pour} \quad c = 1 \text{ à } C$$

Tel que :

- $S_i^{(c)} = \sum w_{i,j}^{(c)} \cdot Z_j^{(c-1)}$ et
- $\Phi'(S) = \dfrac{\partial \Phi(S)}{\partial S}$

si $\Phi'(S) = \frac{1}{1+e^{-\lambda S}}$ alors $\Phi'(S) = \Phi(S) \cdot (1 - \Phi(S))$

si $\Phi(S) = TanHyp(S)$ alors $\Phi'(S) = 1 - (\Phi(S))^2$

Une fois cela fait on exécute un algorithme semblable à celui du perceptron et on modifie chaque poids $w_{i,j}^{(c)}$ comme suit

$$w_{ij}^{(c)} = w_{ij}^{(c)} - \eta \cdot \frac{\partial R_{emp}}{\partial w_{ij}^{(c)}} \qquad 2.11$$

**Complexité d'un PMC**

La complexité d'un PMC est déterminée par le nombre de ses neurones. Si ce nombre est trop faible, le réseau risque de ne pas être assez expressif pour apprendre la fonction désirée. De même, si ce nombre est trop grand, le réseau risque le surapprentissage. Un choix judicieux de ce nombre s'impose, chose qui n'est pas évidente vu que la fonction à apprendre est inconnue. Une des techniques utilisées pour résoudre ce problème est de prendre un réseau de grande taille et d'ajouter une contrainte dans le processus d'apprentissage :

$$\sum w_{i,j} < Constante \qquad 2.12$$

Cette contrainte a pour rôle de limiter le nombre de neurones qui seront réellement utilisés par le réseau (technique de régularisation **[41]**). Car un neurone qui aura tous ses poids synaptiques nuls ou presque peut être enlevé du réseau sans influencer les performances de ce dernier.

## II-1-3-5) Les Réseaux à Fonction de base Radial (RBF):

Les réseaux à fonction de base radiale ont été proposés dès les années 70 par Hardy **[38]**. Le réseau est organisé en couches. La première couche (couche cachée) est constitué de neurones à sorties en forme de cloche et la deuxième couche (couche de sorties) est constitué de neurones à sorties en forme de S.

L'expression mathématique des sorties de ce réseau est :

$$\psi_i(x) = \sum w_{kj} \cdot \Phi(x) = \sum w_{kj} \cdot G(T(x)) \qquad 2.12$$

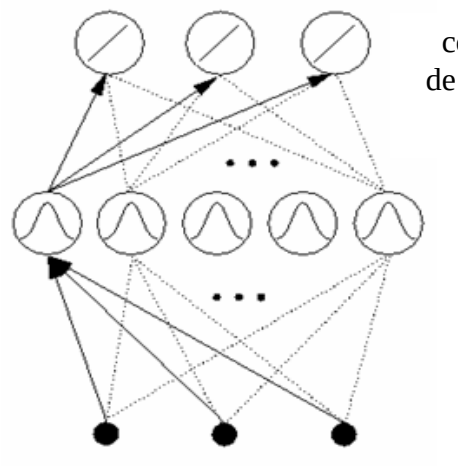
Où

*G ( )* une fonction en forme de cloche

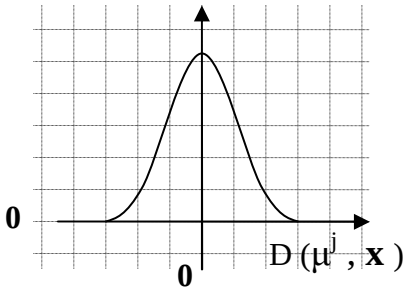
$$T(x) = (\mu^j - x)^t \cdot \frac{1}{\Sigma^j} \cdot (\mu^j - x)$$

$(\mu^j - x)^t$ la transposé de $(\mu^j - x)$

$\Sigma^j$ est la matrice variance covariance des $x$ dans le voisinage de $\mu^j$




Reseau de neurone de J. Moody and C. Darken




Fonction d'activation

**fig 2.4** : Réseau RBF

Pour simplifier le calcul de $\frac{1}{\Sigma^j}$

Généralement on prend

$$\Sigma^j = (\sigma^j)^2 \cdot I$$ $I$ est la matrice unitaire

Pour avoir

$$\frac{1}{\Sigma^j} = \frac{1}{(\sigma^j)^2} \cdot I$$

On va même jusqu'à prendre

$$\Sigma = (\sigma)^2 \cdot I$$

et dans ce cas on aura

$$\frac{1}{\Sigma^j} = \frac{1}{(\sigma)^2} \cdot I$$

Lorsque $\sigma^j$ est grand, l'espace d'entrée couvert par le neurone $j$ de la couche cachée est grand. Lorsque $\sigma^j$ est petit l'espace d'entrée couvert par ce même neurone, rétréci. Par espace d'entrée couvert on veut dire l'ensemble (éventuellement infini) de tous les points

(vecteurs) d'entrée pour lesquels le neurone s'active. On appel souvent $\sigma^j$ largeur de gaussienne.

Grâce à la structure de ce réseau, une entrée $x^i$ éloignée de tous les centres de références des neurones de la couche cachée, n'activera aucun neurone de cette couche. Et donc si le problème traité par le réseau n'est pas complètement défini i.e. l'espace des entrées n'est pas complètement connu (espace ouvert), et q'un vecteur d'entrée $x^i$ d'une nouvelle classe inconnue du réseau se présente, ce dernier ne risque pas de mal classer cette entrée $x^i$.

J. Moody and C. Darken **[38]** ont proposé en 1989 un réseau RBF constitué d'une couche d'entrée, d'une couche caché constituée de neurones à fonction radial et d'une couche de sortie à neurones linéaires. Ils ont proposé pour ce réseau un algorithme apprentissage en deux phases désormais devenu classique. La première phase ajuste les paramètres des neurones de la première couche et la deuxième phase ajuste les poids des neurones de la deuxième couche. Pour la première couche (couche cachée), ils ont proposé d'utiliser l'algorithme des *k*-means pour retrouver les centres $\mu^j$ des gaussiennes et une heuristique qui consiste à varier les largeurs $\sigma^j$ pour assurer un certain recouvrement de chaque point de l'espace d'entrée par ses $n$ plus proches voisins. Pour la deuxième phase, on utilise l'algorithme de rétropropagation de l'erreur pour calculer les $w_{kj}$ . Ce que ne nous dit pas l'algorithme, c'est le nombre de neurones à fonction de base radial à utiliser. Des techniques similaires à celle utilisée dans le cas du PMC peuvent être utilisées afin de répondre à cette question.

## II-1-4) Les arbres de décisions (decision trees)

La construction d'un arbre de décision est un apprentissage supervisé **[43]**. Ce type d'apprentissage est très utilisé surtout que le modèle qui fait ressortir n'exige pas beaucoup de calcul contrairement aux autres méthodes. Un autre atout très important de cette méthode est que le modèle obtenu est très compréhensible. Et même si la taille du modèle est très grande, on peut toujours comprendre son fonctionnement. On trouve deux types d'arbre de décision, à savoir les arbres pour la classification et les arbres pour la régression.

### II-1-4-1) Arbre de classification :( Principe des arbres de décision)

L'idée principale est d'arriver à prendre une décision en répondant à une suite de questions. La réponse fournie à la question de l'étape $N$ déterminera la question de l'étape $N + 1$. L'ensemble de tous les cheminements possibles parmi toutes les questions forme un arbre. D'où le nom de cette technique. Il va sans dire qu'un bon arbre de décision est celui qui minimise le nombre de questions car du coup, il minimise le temps de calcul lors de son exploitation. Pour bien comprendre cette technique, nous allons nous appuyer sur un exemple : supposons que nous voulons programmer un robot zoologue. Ce robot évoluera dans un monde peuplé uniquement de : tortues, léopards, aigles, autruches et lièvres. Et Il sera capable de distinguer :

Q1- si un animal a des plumes.
Q2- si un animal vole.
Q3- si un animal est rapide.
Q4- si un animal est herbivore.

Pour distinguer les aigles des autres animaux notre robot pourra suivre le schéma présent dans la figure 2.5. Grâce à ce schéma, notre robot sera bien capable de distinguer les aigles des autres animaux et ce avec un nombre de questions variant de 1 à 3 selon les cas. Mais avec le schéma de la figure 2.6 la décision est prise avec une seule question car la question choisie discrimine bien les aigles des autres animaux (dans le monde du robot).

Un bon arbre de décision pose les bonnes questions dans le bon ordre et chaque question doit être la plus discriminante possible.

Dans notre exemple chaque nœud de l'arbre a deux nœuds fils. Mais on peut très bien avoir des arbres où chaque nœud peut avoir plus de deux nœuds fils, par exemple : quel est la catégorie socioprofessionnel : médecin, informaticien, menuisier, mécanicien,…etc.

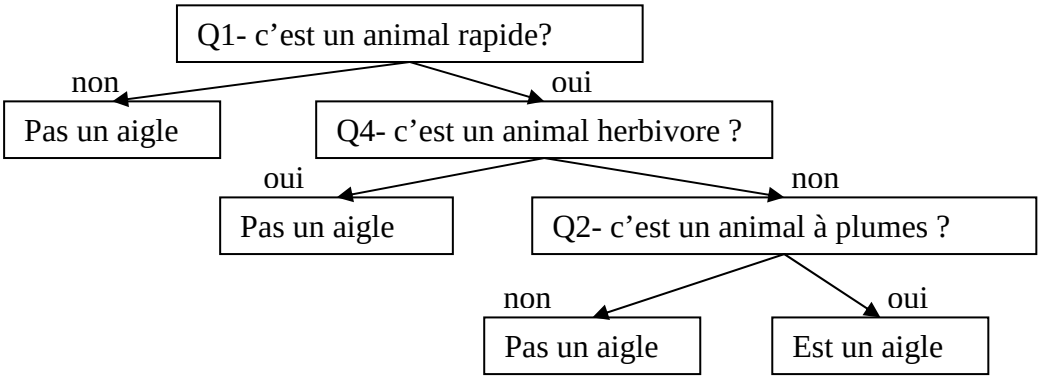



**fig. 2.5 : Arbre de décision long**

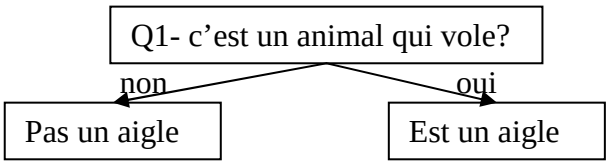



**fig. 2.6 : Arbre de décision court**

### II-1-4-2) Arbre de régression :

Le principe est le même que pour les arbres de classification sauf qu'au lieu de prédire la classe d'un exemple, ce genre d'arbre perdit une valeur numérique discrète ou continue. La figure 2.7 représente un exemple d'arbre de régression

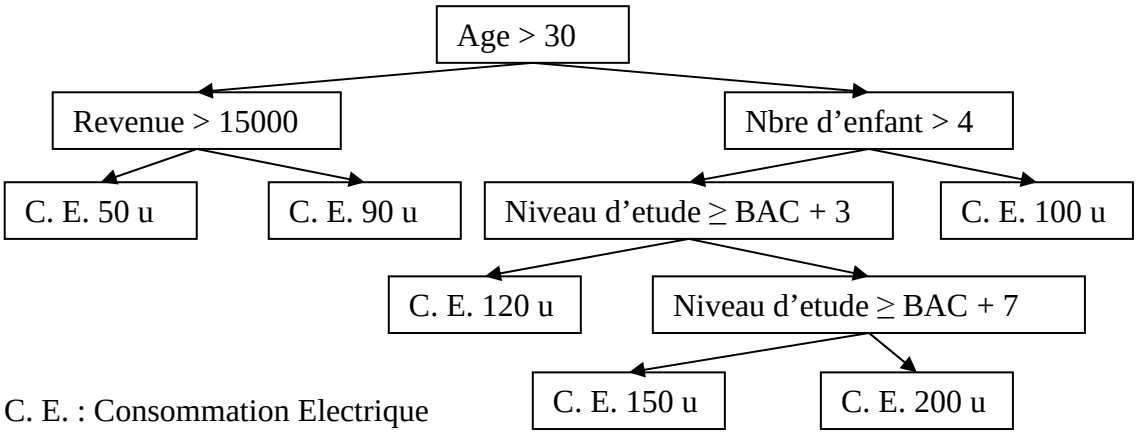



**fig. 2.7 :** Arbre de régression

### II-1-4-3) Construction d'un arbre de décision

Dans le contexte du data mining, il est hors de question de construire un arbre de décision «manuellement». Il faut une méthode automatique, à première vue on pourra croire qu'un algorithme qui va essayer de construire tous les arbres possibles et d'en choisir le meilleur, pourra résoudre le problème. Mais ceci sera impossible à cause de l'explosion combinatoire. Les techniques utilisées se basent toutes sur un critère qui mesure l'homogénéité d'un ensemble de données. Pour chaque nœud, on cherche à trouver la question ou le test qui va créer un partitionnement qui va maximiser ce critère sur l'ensemble des partitions ainsi obtenues. Le partitionnement donne lieu à de nouveaux ensembles de données (les sous-ensembles) pour lesquels on va réitérer le partitionnement. Ce processus est répété jusqu'à ce qu'une condition d'arrêt soit vérifiée. Finalement, chaque chaîne de questions trouvée, constitue les branches de l'arbre. Parmi les critères d'homogénéité ou pureté (ou non homogénéité, impureté) les plus utilisés on trouve **[43]** :

**L'entropie**

Hérité de la théorie de l'information, ce critère atteint son minimum lorsque les éléments d'un ensemble appartiennent tous à la même classe et atteint son maximum dans le cas contraire.

$$Entropie(E) = -\sum f_i \; Log(f_i)$$

Où $f_i$ est la proportion de la classe $i$ dans l'ensemble $E$ ($E$ non vide)

**L'indice de Gini :**

Ce critère aussi atteint son minimum lorsque les éléments d'un ensemble appartiennent tous à la même classe et atteint son maximum dans le cas contraire.

$$Indice\_Gini\ (E) = 1 - \sum f_i^2$$

Où $f_i$ est la proportion de la classe *i* dans l'ensemble *E* (*E* non vide)

**Le critère de Chi 2 :**

C'est le même indice statistique de $\chi_2$

Généralement la construction d'un arbre de décision se fait de manière descendante. La figure 2.8 représente un exemple d'algorithme récursif qui construit un arbre de décision. Bien évidemment l'algorithme de cette figure est simpliste et n'est pas adapté aux situations réelles. Par exemple si nous voulons construire un arbre qui décide si une personne doit faire ou non un test de dépistage pour une certaine maladie et une feuille présente 80% de personnes saintes et 20% susceptibles d'avoir la maladie, notre algorithme tel qu'il est étiquettera cette feuille avec personnes saintes ce qui laissera passer plusieurs porteur de la maladie.

**Algorithme** Construire_Arbre_Avec_Data( E )
**Début**
**Si** ((il n'y a plus d'attribut ou de test possibles sur ce noeud) **OU** (nœud pure) ) **Alors**
- Le nœud courant est une feuille et on l'étiquettera avec le nom de la classe majoritaire dans E

**Si Non**
Sélectionner un attribut $A^*$ et un test $T^*$ sur cet attribut qui maximisent un certain critère qui reflète l'homogénéité globale dans les nouveaux sous ensembles $SE_i$.

Critère(A,T)=$\sum$ (Cardinal ($SE_i$)/ Cardinal (E))* Homogénéité ($SE_i$)

$(A^*,T^*) = (A,T)$ tel que: Critère(A,T) ) = **Max** ( Critère(A,T) )

**Si** (Critère($A^*$,$T^*$) ≥ Critere_Max ) **Alors** ←
le nœud courant devient une feuille et on arrête le partitionnement sur l'ensemble de données E

**Si non**
-Mettre le test T sur l'attribut A dans le nœud courant et attacher au nœud courant des nœuds fils qui contiendront chacun un sous ensemble $SE_i$
-Pour chaque $SE_i$ exécuter Construire_Arbre_Avec_Data ($SE_i$)

**Fin Si**
**Fin Si**
**Fin**

**fig. 2.8:** Algorithme de construction d'arbre de décision.

### II-1-4-4) Le surapprentissage dans les arbres de décision :

Un arbre de décision s'il est construit de façon à s'adapter de manière exacte aux données d'apprentissage, a de grande chance d'avoir de médiocres résultats sur des données autres que celles utilisées en apprentissage. Ce qui est évidemment un problème majeur, puisque cela remet en question toute l'utilité de cet arbre. Donc pour résoudre ce problème on utilise ce que on appel : l'élagage. L'élagage est une technique qui permet d'augmenter considérablement le pouvoir de généralisation de l'arbre. Comme son nom l'indique, cette technique se débarrasse de certaines branches de l'arbre. On a deux types d'élagage :

#### a) Le préélagage :

Dans ce cas il n'y a pas d'élagage proprement dit. Ici on veille à construire un arbre capable de généraliser dés le début de la construction. Pour cela, on introduit dans les conditions d'arrêt de l'algorithme, des paramètres qui vont volontairement empêcher la construction d'arbre ''parfait''. De cette manière, on évite que l'arbre obtenu ne se spécialise dans les données d'apprentissage. L'algorithme de la figure 2.8 est un cas de préélagage. On voit (montré d'une flèche) l'instruction qui arrête prématurément le partitionnement d'un nœud. Ici la condition est sur le degré de pureté des feuilles de l'arbre (seuil sur l'homogénéité). L'atout majeur de ce type d'élagage est que le temps de construction de l'arbre est plus court qu'avec la technique de postélagage.

#### b) Le postélagage :

Dans ce cas il y a une réelle étape d'élagage. Celle-ci aura lieu après la construction de l'arbre sur l'ensemble d'apprentissage. Durant la phase de construction de l'arbre, un nœud n'est transformé en feuille que si le nœud est pure (il ne contient qu'une seule classe) ou s'il ne reste plus de tests possibles capable de diviser ce noeud. Donc à la fin de la première phase, on obtient un arbre très long avec beaucoup de petites feuilles (feuilles qui contiennent peu d'éléments de l'ensemble d'apprentissage). La phase d'élagage se fait avec un ensemble de tests différents de l'ensemble d'apprentissage. L'avantage avec le postélagage, est que si on peut construire un arbre (ou une branche de l'arbre) qui peut être à la fois précis(e) et avec un grand pouvoir de généralisation, cet arbre (ou cette branche) sera vraiment construit(e). Par contre avec la technique de préélagage, ce genre d'arbre (ou branche) sera éliminé(e) d'emblé.

### II-1-4-5) Quelques algorithmes de construction d'arbre de décision :

#### II-1-4-5-1) Algorithme CHAID (CHi-2 Automatic Interaction Detection)

Cet algorithme a été proposé par Kass, G. (1980) **[43]**. Il est le premier à être implémenté dans un logiciel commercial. Cet algorithme produit des arbres n-aires et peut être utilisé pour des données à attributs binaires, multimodales ordonnés ou non et numériques discrètes ou continues. Cet algorithme utilise le teste de Chi 2 pour sélectionner l'attribut sur lequel va se faire le test.

### II-1-4-5-2) Algorithme CART (Classification And Regression Trees) :

Cet algorithme a été proposé par L. Breiman, J. H. Friedman, R. A. Olshen et C. J. Stone en1984 **[43]**. Cet algorithme produit des arbres binaires, c'est-à-dire, chaque nœud parent a uniquement deux nœuds fils. Il peut être utilisé pour tous types de données que se soient des données à attributs binaires, des données à attributs multimodales ou des données à attributs continus. Au niveau de chaque nœud on a un nombre fini de tests possibles. Ce nombre est la somme des tests possibles sur chaque attribut. Le nombre de tests possibles sur un attribut dépend de sa nature et du nombre de ces valeurs possibles. Lorsque l'attribut sur lequel se fait le test est binaire le nombre de tests candidats est un (soit l'exemple porte la première valeur pour cette attribut soit il porte la deuxième valeur). Lorsque l'attribut sur lequel se fait le test est ordonné discret, il y'aura autant de tests candidats que de valeurs capable de séparer les valeurs possibles de l'attribut en deux sous ensembles. Lorsque l'attribut sur lequel se fait le test est continu, on se ramène au cas ordonné discret. L'astuce consiste à ne garder que les valeurs de l'attribut réellement présentes dans les exemples d'apprentissage. Et enfin lorsque l'attribut sur lequel se fait le test est multimodale non ordonné, il y'aura autant de tests candidats que d'éléments dans l'ensemble des parties de l'ensemble des valeurs de l'attribut. Cet algorithme est aussi le premier à implémenter la technique du postélagage.

| type d'attribut | valeurs possibles de l'attribut | tests possibles | nombre de tests candidats ( NT ) |
|---|---|---|---|
| Binaire | E = {$V_1$, $V_2$} | Attribut = $V_1$ | NT = 1 |
| Ordonné discret | E = { $V_1$, $V_2$, … ,$V_n$} | Attribut < V tel que V∈ E – { $V_1$,Vn } | NT = n – 1 |
| Ordonné continue | E = [a,b] (inteval) | Attribut < V tel que V∈ [a,b] et V apparaît comme valeur de l'attribut dans un exemple | NT ≤ m m le nombre d'exemples disponibles |
| Multimodales non ordonné | E = { $V_1$, $V_2$,… $V_n$} | Attribut ∈ S tel que S ∈ $2^E$ (ensemble des parties de E) | NT = $2^{n-1}$ – 1 |

Tableau des tests possibles sur un attribut en fonction de sa nature

**a) Cas de la classification :**

Lors de la recherche du meilleur test à appliquer sur un nœud, le test gagnant est celui qui prend les meilleures valeurs pour le critère. Le critère utilisé ici est l'indice de Gini. Le calcul se fait comme suit :

$$Critere(T) = 2\, P_L \cdot P_R \sum_{j=1}^{nombre\_de\_classes} \left| P(j / t_L) - P(j / t_R) \right| \qquad (2.13)$$

Tel que :

| | |
|---|---|
| $t_L$ $t_R$ | sont respectivement le nœud fils gauche et le nœud fils droit du nœud $t$ obtenus après avoir utilisé le test $T$ sur le nœud $t$. |
| $P_L$ | est le rapport nombre d'exemple dans $t_L$ sur le nombre d'exemple total (dans tous l'arbre). |
| $P_R$ | est le rapport nombre d'exemple dans $t_R$ sur le nombre d'exemple total. |
| $P(j / t_L)$ | est le rapport nombre d'exemples de la classe $j$ dans $t_L$ sur le nombre d'exemple dans $t_L$. |
| $P(j / t_R)$ | est le rapport nombre d'exemples de la classe j dans $t_R$ sur le nombre d'exemple dans $t_R$ |

**b) Cas de la régression :**

L'algorithme se déroule de la même façon que celui de la classification, la seule différence est dans le critère de sélection du test au niveau des nœuds. Ici on utilise la variance résiduelle. Plus ce critère est petit meilleur est le test. Soit $t_L$ $t_R$ respectivement le nœud fils gauche et le nœud fils droit du nœud $t$ obtenus après avoir utilisé le test $T$ sur le nœud $t$. $E$ l'ensemble d'exemples correspondant au nœud $t$ et $E_L$ ( $E_R$ respectivement) l'ensemble d'exemples correspondant au nœud $t_L$ ( $t_R$ respectivement). Alors le critère se calcule comme suit :

$$VR(T) = P_L \cdot Var_L + P_R \cdot Var_R \qquad (2.14)$$

Où :

| | |
|---|---|
| $P_i$ | est la proportion d'exemple de $E$ qui vont dans le sou ensemble $E_i$ / i ∈ { $R$ , $L$ } |
| $Var_i$ | est la variance statistique de l'attribut expliqué (pour lequel on construit l'arbre) dans l'ensemble $E_i$ / i ∈ { $R$ , $L$ } |

Il est facile de remarquer que lorsque la variance $Var_L$ (respectivement $Var_R$) tend vers zéro, cela veut dire que les valeurs de l'attribut expliqué dans l'ensembles $E_L$ (respectivement $E_R$) ne sont pas éloignées de sa moyenne.

**c) L'élagage dans CART**

Comme on l'a dit plus haut, l'algorithme CART utilise une procédure de postélargage appelé élagage à minimisation de complexité. Soit $T$ l'arbre construit, $l(T)$ le nombre de feuilles dans l'arbre $T$ , $R(T)$ l'erreur commise par l'arbre $T$ . En fixant la valeur d'un paramètre α qui sert à régler le compromis entre la complexité de l'arbre (beaucoup de feuillage) et le taux d'erreur, le coût de la complexité de l'arbre $T$ est

$$CC(T) = R(T) + \alpha \cdot l(T) \qquad (2.15)$$

Le taux d'erreur se calcule différemment selon la nature de l'arbre :
Pour un arbre de classification :

$$R(T) = \frac{r}{n} \qquad (2.16)$$

Où $r$ le nombre d'exemples mal classés par l'arbre,

$n$ le nombre d'exemples total

Pour un arbre de régression :

$$R(T) = \sum P_i \cdot Var_i \qquad 2.17$$

Où $P_i$ la proportion d'exemple dans la feuille $i$
$Var_i$ la variance dans la feuille $i$, de l'attribut expliqué

La procédure d'élagage se fait en deux phases. La première phase sélectionne des arbres candidats en se basant sur le $CC$. Pour cela, cette phase devra construire tous les arbres possibles en élaguant l'arbre $T$. Cela n'est pas nécessaire car il a été démontré que pour une valeur fixé de $\alpha$ il y'a un seul arbre $T_i$ qui minimise $CC$. Tous les autres arbres sont soit des arbres dont le $CC$ est supérieur, soit des arbres qui ont un $CC$ équivalent et qui peuvent donner $T_i$ après élagage. Autrement dit, si en élaguant $k$ feuilles de l'arbre $T$ on trouve que l'arbre qui minimise $CC$ est $T_k^*$ , alors on est sure que l'arbre $T_{k+1}^*$ qui minimise le $CC$ dans l'ensembles des arbre obtenu en élaguant $k+1$ feuilles de l'arbre $T$ , est contenu dans l'arbre $T_k^*$.
Donc il suffit de trouver la suite $T_0,T_1,\ldots,T_k,T_{k+1},\ldots, T_n$ tel que $T_{i+1} \subset T_i$

Cela est fait comme suit :

1- soit $T_0$ l'arbre obtenu juste après la phase d'apprentissage.

2- on initialise $k$ avec 1 ; et l'arbre $T$ avec l'arbre $T_0$

3- on prend l'arbre $T$ et on le transforme en un arbre à $l(T) - 1$ feuilles en unissant deux feuilles issue du même nœud parent en une feuille qui va remplacer ce même nœud parent. On calcule le $CC$ pour ce nouvel arbre.

4- on reprend le même arbre $T$ et on refait l'étape 3 avec deux autres feuilles jusqu'à ce qu'on les ait toutes utilisées.

5- de l'étape 3 et 4 on garde l'arbre qui minimise $CC$. Soit $T_k^*$ cet arbre.
6- si l'arbre $T_k^*$ possède des feuilles alors :
$T$ devient $T_k^*$
$k = k + 1$
On passe à l'étape 3

7- fin de l'algorithme

Où $T_0$ : l'arbre obtenu par la phase d'apprentissage avant la phase d'élagage
$k = 0$ ;
$T = T_0$;
$T_{tmp} = Petit\_Elagage(T,N)$

Après la fin de la première phase, on obtient donc la suite d'arbre $T_0,T_1,\ldots,T_k,T_{k+1},\ldots, T_n$. la deuxième phase consiste à sélectionner l'arbre qui minimise l'erreur sur l'ensemble de tests. Pour déterminer la valeur du paramètre $\alpha$, l'algorithme CART propose d'utiliser la validation croisée pour construire l'arbre qui minimise le CC sans passer par aucune étape d'élagage.

Puis une fois cet arbre construit, on prend le α correspondant à cet arbre et on l'utilise pour la suite.

### II-1-4-5-3) Algorithme C4.5

Cet algorithme a été proposé par J. R. Quinlan en 1993 **[43]**, il est une amélioration de ID3 du même auteur présenté en 1986. Comme l'algorithme CART, cet algorithme peut être utilisé sur des données à attributs binaires, des données à attributs multimodales ou des données à attributs continus. Par contre ici, les arbres produits peuvent être n-aire (un nœud père a n nœuds fils). Cela est mis à profits lorsque l'attribut en cours d'analyse est multimodale non ordonné. A ce moment l'algorithme crée un nœud fils pour chaque valeur possible pour cet attribut. La sélection de l'attribut et du test à lui appliquer est faite en même temp. Pour cela l'algorithme recherche à la fois l'attribut et le test à appliquer en parcourant toutes les possibilité au niveau du nœud courant. Comme critère de sélection de test, l'algorithme utilise l'entropie et le gain d'information.

$$Gain(E,T) = Entropie(E) - \sum P_i \, Entropie(SE_i, T) \qquad (2.18)$$

Où $E$ est l'ensemble qui correspond au nœud en cours de partitionnement
$SE_i$ sont les sous ensemble issus de $E$ par le test $T$
$P_i$ est la proportion dans $E$ qui vont dans $SE_i$

En suite, il suffit de prendre le test qui maximise le gain d'information et passer au nœud suivant.

L'algorithme C4.5 applique aussi un postélagage. Une des particularités de l'algorithme d'élagage utilisé ici, est qu'il utilise les mêmes données utilisées durant l'apprentissage. L'algorithme élague l'arbre à la recherche du sous arbre qui minimise le risque réel. Le risque réel de l'arbre est calculé comme suit :
Tout d'abord on calcule $P$, la probabilité d'erreur pour chaque feuille. Cette probabilité est sensée représenter la probabilité d'erreur réel pour cette feuille. Pour cela, on se fixe un paramètre de confiance $F$. On note $E$ le nombre d'exemple mal classé par la feuille. $N$ le nombre d'exemple total dans la feuille.
Et on calcule :

$$P = \max\{ p \mid \Pr(E_p \leq E) \geq F\} \qquad (2.19)$$

Où $E$ est une variable aléatoire de loi binomiale de paramètre ( $N$, $p$).

Une fois qu'on a calculé l'erreur réelle pour chaque feuille, on calcule l'erreur réelle de l'arbre comme suit :

$$R = \sum P_i \cdot \frac{E_i}{N_i} \qquad (2.20)$$

Où $P_i$ est l'erreur réelle de la feuille $i$
$E_i$ est le nombre d'erreurs commises par la feuille $i$ sur l'ensemble d'apprentissage
$N_i$ est le nombre d'exemples dans la feuille $i$

# II-2) Détection de clusters :

La détection automatique de cluster est un type d'apprentissage non supervisé. L'objectif du clustering est double, dans un premier temps il détecte dans un ensemble de données les groups homogènes. Cela est fait durant la phase d'apprentissage. Dans un deuxième temps, le système classe les données qu'on lui présente en se basant sur les groupes qu'il a découverts dans la première phase. Un des avantages du clustering et qu'il permet de travailler sur des données qui n'ont pas besoin d'être classifiées par l'homme au préalable.

Un bon clustering doit garantir une grande homogénéité intraclusters et une faible homogénéité interclusters. La qualité du résultat du clustering dépend en grande partie de la distance choisie.

Il existe plusieurs méthodes et algorithmes de clustering, et on trouve aussi diverses classifications de ces méthodes. Dans ce qui suit nous allons donner un petit aperçu de quelques méthodes de base **[42]**.

## II-2-1) Méthodes par partitionnement :

Dans ce genre de méthodes on construit un partitionnement de l'ensemble de données (clusters) et on l'améliore à chaque étape. Selon l'algorithme utilisé un cluster est repéré par :

- Le centre du cluster : qui est la moyenne de tous les exemples du cluster. Utilisé dans k-means, ISODATA.
- Le medoide : qui est le point le plus proche de tous les autres points du cluster. Généralement le medoide est utilisé dans le cas de points décrit par des attributs qualitatifs (exemple couleur). Utilisé dans PAM, CLARA, CLARANS.

### II-2-1-1)Clustering avec k-moyennes (k-means):

Cette algorithme été proposé par J. B. MacQueen (1967) **[42]**. L'objectif de la méthode est de construire les *k* meilleurs centres de groupes de l'ensemble de données d'apprentissage. L'algorithme se déroule comme suit :

1- Choisir *k* centres de groupe de manière arbitraire
2- parcourir tous les exemples et affecter chacun d'eux au groupe dont le centre lui est le plus proche.
3- calculer les centres des groupes.
4- si les centres des groupes on changé aller vers 2 si non fin de l'algorithme.

A la fin, l'algorithme aura minimisé la quantité :

$$Q = \sum_{j=1}^{k} \sum_{i=1}^{n_j} D(x_j, c_j) \qquad (2.21)$$

Où $n_j$ est le nombre d'exemple dans le cluster j
$c_j$ est le centre du cluster j

$D$ la distance choisie

### II-2-1-2)Clustering avec les K-moyennes floues (fuzzy K-means):

Cet algorithme a été proposé par J. C. Dunn (1973) **[42]**. Il a le même fonctionnement que k-means sauf qu'ici on utilise la logique floue. Un exemple peut appartenir à plusieurs groupes (clusters) avec des degrés différents. $u_{ij}$, le degré d'appartenance de l'exemple $i$ au cluster $j$ ; $c_j$ , le centre du cluster $j$ sont calculés comme suit :

$$\begin{cases} u_{ij} = \left( \sum_{l=1}^{k} \left( \frac{D(x_i, c_j)}{D(x_i, c_l)} \right)^{\frac{1}{m}} \right)^{-1} \\ c_j = \frac{\sum_{i=1}^{n_j} u_{ij}^m \cdot x_i}{\sum_{i=1}^{n_j} u_{ij}^m} \end{cases} \qquad (2.22)$$

Où $m$ est une constante choisie au préalable, $m \geq 1$
$n_j$ est le nombre d'exemples dans le cluster $j$
$x_i$ parcoure tous les exemples du cluster $j$

Dans cet algorithme la condition d'arrêt est aussi un peut différente, elle est exprimée comme suit :

Si $\text{Max}_{ij} ( | u_{ij}^{(t+1)} - u_{ij}^{(t)} | ) < \varepsilon$ Alors arrêter l'algorithme,

Où $0 \leq \varepsilon \leq 1$
$u_{ij}^{(t)}$ est le degré d'appartenance de l'exemple $i$ au cluster $j$ lors de l'itération $t$.

L'exécution de l'algorithme aura pour conséquence la minimisation du critère suivant :

$$Q = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij}^{m} D(x_i , c_j) ; \qquad m \geq 1 \qquad (2.23)$$

## II-2-2) Méthodes Hiérarchiques :

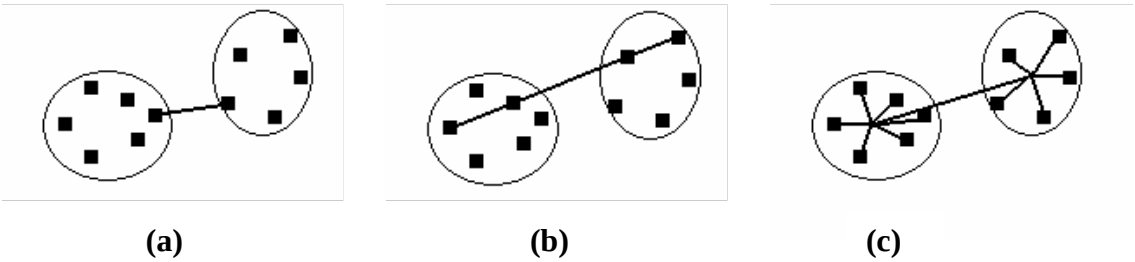
Avec ce genre de méthodes, le résultats du clustering est présenté sous forme d'arbre de clusters on dit aussi dendrogramme

### II-2-2-1) Clustering hiérarchique ascendant:

Proposé par S. C. Johnson (1967) **[42]**, cet algorithme construit les clusters à l'étape $t+1$ en se basant sur les résultats de l'étape $t$. Le déroulement de l'algorithme est comme suit :

1- Créer autant de clusters que d'exemples, chaque exemple constitue un cluster, et calculer les distances entre tous les clusters.
2- Chercher les deux cluster les plus proches l'un de l'autre et les unir pour en former qu'un seul.
3- Calculer à nouveau les distances qui séparent tous les clusters.
4- S'il reste plus d'un cluster, aller à l'étape 2, si non, fin de l'algorithme.

Il existe plusieurs variantes de cet algorithme, leur principale différence réside dans la distance utilisée mais surtout dans la manière dont cette distance est calculée. Pour ce dernier critère on peut citer :
- Single Link : la distance entre deux clusters est la distance entre les membres les plus proches l'un de l'autre de ces deux clusters (figure 2.9 (a)).
- Complet Link : la distance entre deux clusters est la distance entre les membres les plus éloignés l'un de l'autre de ces deux clusters (figure 2.9 (b)).
- Average Link : la distance entre deux clusters est la distance moyenne entre tous les membres de ces deux clusters.
- La distance entre les centres des deux clusters (figure 2.9 (c)).



**(a)** **(b)** **(c)**

**fig. 2.9 :** Le calcul des distances entre clusters

A la fin de l'algorithme pour obtenir k clusters il suffit de prendre les clusters de l'étape $(n + 1 – k)$. En faite on pourrait tous simplement arrêter l'algorithme à cette étape là.

### II-2-2-2) Clustering hiérarchique descendant:

Dans ce qui a précédé nous avons vue le clustering hiérarchique ascendant. Il existe un autre clustering hiérarchique dit descendant. Ce clustering part d'un cluster initial contenant tous les exemples et subdivise à chaque étape le cluster adéquat jusqu'à ce qu'il y'ai autant de clusters que d'exemples d'apprentissage. Cela est notamment fait dans l'algorithme DIANA. Ce genre d'algorithme n'est pas très utilisé car très gourmand en calculs.

## II-2-3) Méthodes basées sur la densité:

Ce genre d'algorithme utilise la densité pour trouver les clusters. Les zones de densité de point élevé constituent les clusters. Les zones de faible densité constituent les bordures des clusters. Un des avantages de ces algorithmes est de pouvoir construire des clusters de différentes formes notamment de forme concaves ce qui n'est pas le cas des algorithmes cités précédemment. De plus les résultats obtenus sont moins sensibles au bruit. Dans cette ligné

d'algorithmes on trouve DBSCAN, OPTICS, CLIQUE,… . Ci-dessous nous donnons en exemple DBSCAN.

### Clustering avec DBSCAN:

Cet algorithme a été présenté par Ester M., Kriegel H.-P., Sander J., Xu X. **[42]**. Cet algorithme nécessite d'initialiser deux paramètres : *Eps* , le rayon maximal de voisinage ; et *MinPts* , le nombre de points minimal dans le voisinage. La fonction suivante est donnée:

$$N(x_i) = \{ x_j \mid D(x_i , x_j) \leq Eps\} \qquad (2.24)$$

Aussi, les notions suivantes sont définies
$x_i$ est 'directly density reacheable' par $x_j$ si et seulement si:
- $x_i \in N(x_j)$ et
- $| N(x_j) | \geq MinPts$

$x_i$ est 'density reacheable' par $x_j$ si et seulement si il existe une suite de points $z_1, z_2, \ldots, z_{k-1}, z_k$ tel que $z_1 = x_j$ et $z_k = x_i$ et $z_{m+1}$ est 'density reacheable' $z_m$ pour $m = 1$ à $(k-1)$.

Deux exemples illustratifs sont donnés sur la figure 2.10. Sur la figure 2.10(a) on a le point croix est 'directly density reacheable' par le point rond. Sur la figure 2.10(b) le point croix est 'density reacheable' par le point rond. Sur cette dernière figure, on remarque nettement la forme concave du cluster.
Le déroulement de l'algorithme est comme suit :

1- initialiser *Data* avec l'ensemble des points d'apprentissage.
2- choisir un point $x_i$ arbitrairement de *Data*
3- chercher tous les points $x_j$ appartenant à *Data* qui sont 'density reacheable' du point $x_i$.
4- former un cluster avec les points $x_j$ et le point $x_i$ et les hauteurs de l'ensemble *Data*.
5- Si *Data* n'est pas vide aller vers 1 si non fin de l'algorithme.

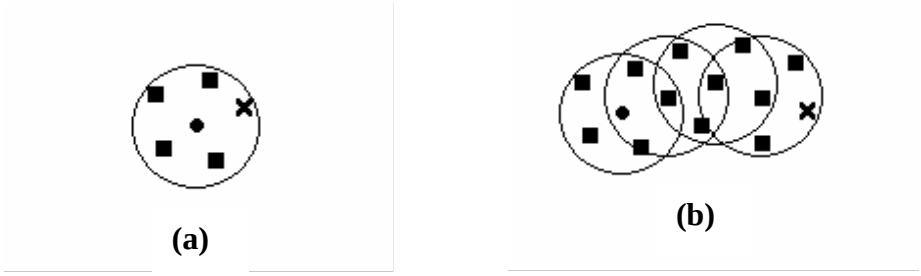



**fig. 2.10 :** Le calcul des distances entre clusters

## II-2-4) Méthodes basées sur un découpage (Grid based ) :

Ce genre de méthode découpe l'ensemble de données en un ensemble de cellules formant une grille. Les cellules de la grille sont traitées une à une. L'avantage ici est la possibilité de traiter des ensembles de données très très large, puisque les données n'ont pas

besoin de tenir en mémoire centrale. Un autre avantage est la relative facilitée de parallélisation. Parmi les algorithmes qui utilisent cette technique on cite STING, GRIDCLUS, WaveCluster, CLIQUE,…. Ci-dessous nous présentons de manière sommaire l'algorithme STING

## Algorithme STING :

Cet algorithme a été proposé par Wei Wang, Jiong Yang et Richard Muntz **[42]**. Cet algorithme fait plusieurs niveaux de découpage de l'ensemble de données. Le premier niveau de découpage est l'ensemble de données en entier. Le prochain niveau de découpage fait découper à nouveau chaque cellule du niveau précédent en un ensemble de cellules (une sorte de zoom sur les cellules du niveau précédent). Des informations statistiques sont associées à chaque cellule. Les informations statistiques des cellules d'un certain niveau sont calculées uniquement à partir des cellules du niveau immédiatement en dessous. Ces calculs sont effectués au préalable et une fois pour tout. Les informations en question sont : *n*, le nombre de points dans la cellule; *m*, la moyenne de tous les points dans la cellule; *s* la déviation standard de tous les attributs dans la cellule; *min*, les valeurs minimales de tous les attributs dans la cellule; *max*, les valeurs maximales de tous les attributs dans la cellule; et enfin *dist* qui est le type de distribution statistique de la cellule (normal, uniforme,…). Le calcul des informations des cellules de haut niveau se fait de la manière suivante :

$$\begin{cases} n = \sum_i n_i \\ m = \frac{1}{n}\sum_i m_i \cdot n_i \\ s = \left( \sum_i \left( \frac{\left(s_i^2 + m_i^2\right)\cdot n_i}{n} - m^2 \right) \right)^{\frac{1}{2}} \\ min = \text{Min}\ (min_i) \\ max = \text{Max}\ (max_i) \end{cases} \qquad (2.25)$$

pour *dist* le calcul est un peu plus compliqué, il se fait comme suit :

si ( $dist_i \neq dist$) et ($m_i \approx m$) et ($s_i \approx s$) alors *confl* est incrémenté de $n_i$
si (($m_i \neq m$) ou ($s_i \neq s$)) alors *confl* est affecté de *n*
si ($dist_i = dist$) et ($m_i = m$) et ($s_i = s$) alors *confl* n'est pas modifié
si $\frac{confl}{n} > t$ alors *dist* est affecté de *unknown*,

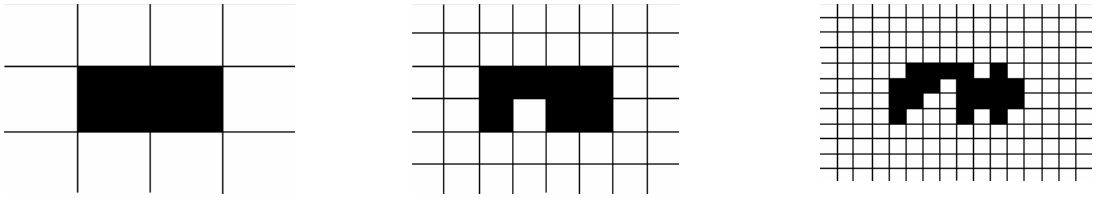
t est un seuil fixé au début de l'algorithme.

L'algorithme se déroule comme suit :

1- Sélectionner un niveau avec peu de cellules pour commencer l'exécution et mettre toutes ces cellules dans *C*.

2- Calculer le niveau de confiance dans chaque cellule de *C*.
3- Eliminer les cellules de *C* dont le niveau de confiance est faible.
4- Remplacer chaque cellule de *C* par les cellules de niveau inférieur lui correspondant et aller vers 2 si non fin de l'algorithme.

L'algorithme STING permet de répondre à plusieurs types de questions du genre quelle est la région qui satisfait à tel critère ? Quelle est la valeur de tel critère dans telle région. Sur la figure 2.11 on voit un exemple de déroulement de l'algorithme.



niveau 1 niveau 2 niveau 3

**fig. 2.11 :** Déroulement de STING

## II-2-5) Méthodes basées sur les modèles:

Dans ce type de clustering, on suppose que les données suivent un modèle mathématique paramétrable, et on cherche à trouver les paramètres qui vont adapter au maximum le modèle et les données. On trouve dans cette catégorie les réseaux SOM (Self Organized Maps) les modèles en mélange de distributions,…etc.

### II-2-5-1) Clustering avec mélange de gaussiennes

Dans cet algorithme, on suppose l'existence de sous groupes dans l'ensemble des exemples. Chaque groupe suit une loi gaussienne. Le nombre de ces groupes est fixé à l'avance. Le problème ici est de trouver les paramètres de ces différentes gaussiennes et le poids à accorder à chacune de ces gaussiennes.

La distribution totale des données suit la loi :

$$L(\theta) = \sum_{i=1}^{g} \pi_i \, G_i(\mu_i , \sigma_i) \qquad (2.26)$$

où $\theta = (\theta_1, \theta_2,.., \theta_g) = (\pi_1 ,\mu_1 ,\sigma_1 , \pi_2 ,\mu_2 ,\sigma_2 ,\ldots,\pi_g ,\mu_g ,\sigma_g)$

On cherche à maximiser la vraisemblance

$$P(x \mid \theta) = \prod_{i=1}^{n} P(x_i \mid \theta) = \prod_{i=1}^{n} \left( \sum_{j=1}^{g} \pi_j G_j(\mu_j, \sigma_j, x_i) \right) \qquad 2.27$$

Pour faciliter le calcul on cherche plutôt à maximiser le logarithme de la vraisemblance :

$$Log\left(P(x \mid \theta)\right) = \sum_{i=1}^{n} Log\left( \sum_{j=1}^{g} \pi_j G_j(\mu_j, \sigma_j, x_i) \right) \qquad 2.28$$

Le calcul analytique de cette formule n'est pas possible, alors on utilise l'algorithme EM (Expectation Minimization) introduit par A.P. Dempster, N.M. Laird et D.B. Rubin **[42]**. Dans la suite, nous donnerons les grandes lignes de cet algorithme.

1. Initialiser aléatoirement les paramètres $\theta = \theta^0$ ;
2. Etape E : calculer les responsabilités de chaque point dans chaque composante

$$\gamma_{ik} = \frac{\pi_k \cdot G(\theta_k, x_i)}{\sum_{j=1}^{g} \pi_j \cdot G_j(\theta_j, x_i)}$$

3. Etape M : estimation des paramètres du mélange

$$\mu_k = \frac{\sum_{j=1}^{n} \gamma_{ik} x_i}{\sum_{j=1}^{n} \gamma_{ik}}$$

$$\sigma_k = \frac{\sum_{j=1}^{n} \gamma_{ik} (x_i - \mu_k) \cdot^t (x_i - \mu_k)}{\sum_{j=1}^{n} \gamma_{ik}}$$

$$\pi_k = \frac{\sigma_k \cdot \gamma_{ik}}{n}$$

4. Répéter 2 et 3 jusqu'il y'ai convergence ;

### II-2-5-2) La carte auto organisatrice de Kohonen

La carte SOM (pour Self Organizing Map) a été proposée dans **[42]**. Elle trouve son inspiration dans le cortex cérébrale de la vision. Elle se compose de deux couches, une couche d'entrée et une couche de sortie avec un nombre de neurones largement inférieure à celui de la couche d'entrée. Les neurones de la couche de sortie sont à fonction d'activation localisée telle que les neurones à RBF. Chaque neurone de la couche de sortie possède un vecteur poids

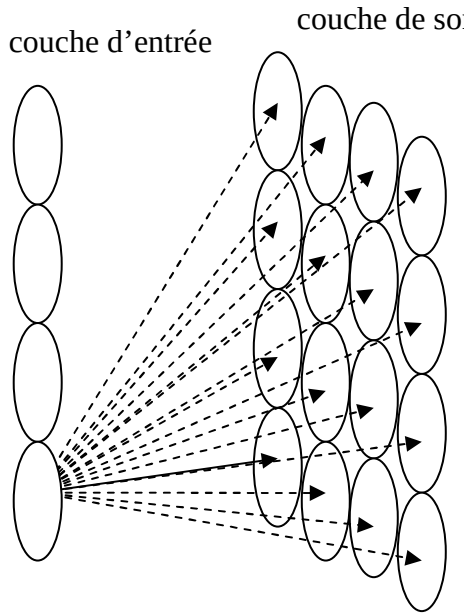
qui est le centre de sa fonction d'activation. La topologie de la couche de sortie est généralement en deux ou trois dimensions (d'où l'appellation carte). Bien que possible, de plus grandes dimensions rendent l'interprétation des résultats difficile. Chaque neurone de la couche d'entrée est connecté vers tous les neurones de la couche de sortie. Le fonctionnement du réseau SOM est très simple.

Lorsque une entrée $x_i$ est présentée au réseau, le neurone dont le vecteur poids $\mu_i$ est le plus proche de $x_i$ au sens d'une distance $D(.,.)$**,** est activé. La distance utilisée est généralement la distance euclidienne

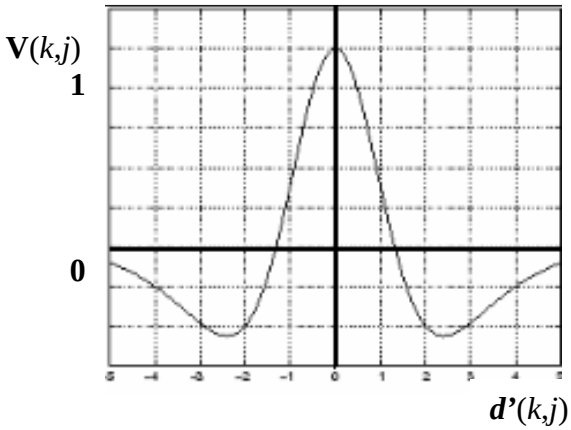
$$D(x_i, \mu_i) = \sqrt{\sum_j (x_i^j - \mu_i^j)^2}$$

mais d'autres distances peuvent être utilisées tel que la distance absolue, …etc.

Une fois un neurone sélectionné, une deuxième phase ajuste les poids de ce neurone et de ses voisins. Le voisinage d'un neurone peut être simplement les neurones qui lui sont adjacents ou bien tous les neurones inclus dans un périmètre dont il est le centre.




Carte SOM




Variation du voisinage **V** en fonction de la distance ***d'*** entre les neurones k et j

**fig. 2.12 :** Réseau SOM

Généralement la valeur de la fonction voisinage décroît avec l'accroissement de la distance entre les neurones (voir exemple sur la figure 2.12). L'ajustement des poids peut être exprimé par :

$$\mu_k = \mu_k + ( \eta \cdot V(k,j) \cdot D( x_i , \mu_i) ) \qquad 2.29$$

Où $j$ le neurone activé
$k$ un neurone voisin
$\mu_n$ le vecteur poids du neurone $n$
$\eta$ cœfficient d'apprentissage
$V(k,j)$ une fonction de voisinage tel que $V(j,j)=1$ ;

Une fois l'apprentissage terminé, il ne reste plus qu'à interpréter les résultats. Il faut noter que l'apprentissage ici est non supervisé. La carte fournie (la couche de sortie) conserve la topologie des données d'entrée i.e. les entrées proches dans l'espace d'entrée restent proches dans l'espace de sortie et vice versa. Donc, avec les réseaux SOM, on passe d'un espace de grande dimension vers un espace de petite dimension (2 ou 3) tout en gardant la topologie des données.

## II-2-6) Clustering semi supervisé :

Lors du clustering, on cherche à trouver le regroupement des éléments homogènes **[6]**. Cependant il y'a des cas ou l'on peut avoir plusieurs types d'homogénéité.

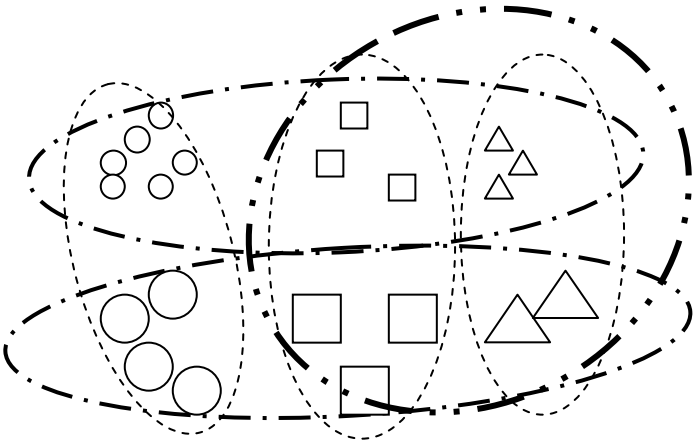


**fig. 2.13 :** Influence de la supervision dans le clustering

.

Prenons pour exemple la classification de livres. Si on demande à un humain de classifier un ensemble de livres, il pourra le faire de plusieurs façons : selon l'auteur, selon le thème, selon la langue, selon le nombre de pages,… . Toutes ces manières de classifier les livres sont correctes mais laquelle nous intéresse ? Dans cet exemple il suffit de fixer à l'avance le type d'homogénéité désiré et définir la distance à utiliser en fonction de cette dernière. Dans les problèmes réels traités par le clustering on peut définir l'homogénéité désirée mais il est difficile de construire une distance en fonction de celle-ci. Ou bien encore, on peut avoir une idée vague du type d'homogénéité souhaitée. Pour cela le clustering supervisé utilise un autre moyen : dans l'ensemble d'apprentissage, une partie des exemples est étiquetée. Ces exemples étiquetés servent comme un guide qui indique à l'algorithme la bonne voie à suivre.

# II-3) Règles d'association :

## II-3-1) présentation et définition:

La découverte de règles d'association est une technique introduite par Agrawal en 1993 dans **[7]**. L'application typique de cette technique est dans l'analyse du panier de la ménagère. Son principe est le suivant : étant donnée un ensemble d'articles et un ensemble de tickets de caisse, on examine les tickets un par un pour en extraire les articles qui sont souvent présents ensemble dans un même ticket. Une fois fait, cela nous permet d'obtenir des règles d'association entre articles. Ces règles sont composées en deux parties une partie condition et une partie résultat. Par exemple, si dans l'ensemble des tickets de caisse on trouve que les articles A, B et C sont souvent présents ensemble, on peut conclure des règles du genre $A,B \rightarrow C$ qui signifie si une ménagère achète le produit A et B alors elle va probablement acheter le produit C, ou bien $A,C \rightarrow B$ qui signifie que si la ménagère achète les produits A et C alors elle va probablement acheter le produit B, …etc. Bien évidemment on ne garde que les règles qui répondent à certains critères. Le premier critère est le support. Il indique la probabilité de présence de la règle (partie condition et partie résultat ensemble) dans l'ensemble des tickets (espace d'entrée). Si on veut calculer le support pour la règle $X \rightarrow Y$, alors on doit calculer le nombre de tickets total $N_T$ et le nombre de tickets qui comportent les articles X et Y en même temps $N_{XY}$. Puis faire une simple division pour avoir le support.

$$Support(X \rightarrow Y) = N_{XY} / N_T \qquad (2.30)$$

Le deuxième critère est la confiance. Il mesure le degré de confiance qu'on peut avoir en une règle. Si on veut calculer la confiance qu'on peut avoir dans la règle précédente $X \rightarrow Y$ alors on doit calculer le nombre de tickets qui comportent les articles X et Y en même temps, dénoté $N_{XY}$, et le nombre de tickets qui comporte l'article X avec ou sans l'article Y, dénoté $N_X$, puis calculer le quotient des deux :

$$Confiance(X \rightarrow Y) = N_{XY} / N_X \qquad (2.31)$$

Il faut noter que

$$Support(X \rightarrow Y) = Support(Y \rightarrow X)$$
$$Confiance(X \rightarrow Y) \neq Confiance(Y \rightarrow X)$$

Lorsque on extrait les règles on ne garde que les règles qui on un support supérieur au minimum *MinSprt* et un degré de confiance supérieur à un autre minimum *MinConf* généralement plus élevé.

Une fois les règles extraites, leur utilisation est très effective. Elles répondent à des questions du genre quel est la meilleure disposition des articles, par exemple si on sait qu'une ménagère qui achète du lait achètera probablement aussi du café, alors on dispose le café à coté du lait. Ou bien encore, quelle conséquence aura le faite de cesser de vendre le produit A sur les autres produits…etc. L'analyse du panier de la ménagère ne se résume pas uniquement à l'analyse des produits achetés ensemble. On peut aussi l'étendre à un ensemble d'achats effectués durant une longue période. Par exemple, si on sait qu'un client qui achète un micro-ordinateur aujourd'hui, va acheter un filtre à écran dans quinze jours et que l'on a vendu 100 micro-ordinateurs sans filtre à écran durant cette semaine. Alors, un simple calcul nous donnera l'estimation du nombre de filtres à écran dont le magasin aura besoin pour

satisfaire ses clients. L'utilisation des règles d'association ne se limite pas à l'analyse du panier de la ménagère, on peut aussi l'utiliser pour l'analyse des données de recensement, pour la sélection de candidats à des postes de travail,…etc.

On voit que la recherche des règles d'associations se résume à parcourir une longue suite de tickets de caisse par exemple et à compter le nombre de fois où un ensemble d'articles sont présents ensembles. Les différentes approches proposées essayent d'optimiser au maximum le temps de calcul nécessaire.

Par la suite en essayera d'adopter la syntaxe de Agrawal. Un item dans ce chapitre est typiquement un article, un produit, ...etc. Ce sont tous les éléments qui peuvent apparaître dans une règle d'association. L'ensemble des tickets de caisse est sauvegardé dans une table de base de données ou un fichier, pour être analysé. Chaque ligne de la table est un tuple .Un itemset est un ensemble d'items. Un itemset peut être l'ensemble des items apparaissant dans une règle ( {A, B, C} est un itemset dont est extraite la règle A,B,→ C ), ou l'ensemble des items apparaissant dans la partie condition d'une règle ({A,B}). Dans un tuple de $n$ items, on peut trouver $2^n$ itemset (toutes les combinaisons possibles des items présents dans ce tuple sans oublier l'ensemble vide). Un k-itemset est un itemset dont le nombre d'items est k (cardinalité k). Un itemset fréquent est un itemset qui se répète (se présente) avec une fréquence d'au moins *MinSprt*.

Un lemme très intéressant dit qu'un k-itemset fréquent est composé de (k-1)-itemset fréquents. En effet si un itemset $I_k$ se répète au moins *MinSprt* alors tous les itemset $I_{(k-1)}$ inclus dedans se répètent aussi au moins autant de fois. Donc, si on a tout les k-itemset fréquents on peut trouver tous les (k+1)-itemset susceptibles d'être des itemset fréquents. Pour cela il suffit de vérifier que tous les k-itemset sous ensembles d'un (k+1)-itemset sont bien des k-itemset fréquents. Évidemment les nouveaux itemset ainsi construits ne sont pas forcément des itemset fréquents et doivent donc être vérifiés.

Cette idée est exploitée dans les algorithmes de recherche de règles d'association. Ainsi ces algorithmes commencent par rechercher les 1-itemset fréquents, puis les 2-itemset fréquents, puis les 3-itemset, …etc.

## II-3-2) Algorithme Apriori de recherche de règles

Après les algorithmes AIS et SETM qui soufraient de la lenteur dû à la génération des itemset candidats à la volé, l'algorithme Apriori fut le premier algorithme réellement efficace et applicable à de grandes bases de données. Il a été proposé par Agrawal en 1994. L'algorithme travaille par étapes ou passes. A chaque nouvelle passe, un ensemble d'itemsets susceptibles d'être fréquents sont sélectionnés. Ces itemsets sont dit candidats. A la première passe, l'ensemble des itemsets candidats est pris comme étant tous les itemset constitué d'un seul item. C'est-à-dire que pour chaque item on crée un itemset qui se résume à ce seul item. Un compteur est associé à chaque itemset candidat. Puis les tuples de la table (les tickets de caisse) son examinés un par un. Pour chaque tuple **t**, on cherche les itemsets candidats qui apparaissent dedans, on fait incrémenter le compteur de chacun des itemsets présent dans ce tuple **t**.

Une passe se termine lorsque tous les tuples de la table ( *NbrTuple* tuples) ont été examiné. Pour la nouvelle passe $\mathbf{P_{k+1}}$, on doit construire un nouvel ensemble d'itemsets candidat. Pour cela, on extrait parmi les itemset candidats de la passe précédente $\mathbf{P_k}$, tous les itemset candidats qui ont leurs compteurs respectifs au moins égals à $MinSprt * NbrTuple$. En effet ces itemsets ont une fréquence d'apparition dans la table qui dépasse *MinSprt*. Ils sont

dits itemset fréquents. Avec ces k-itemsets fréquents, on construit les (k+1)-itemset susceptibles d'être fréquents. Un (k+1)-itemset sera candidat pour être compté durant la passe suivante, si tous les k-itemset qui peuvent être construits à partir de ses items sont des k-itemsets fréquents. Après ça, on fait une autre passe dans la table avec les nouveaux itemsets candidats. Et ainsi de suite pour chaque passe, jusqu'à ce qu'on n'ait plus de nouveaux itemset fréquents et donc plus la possibilité de construire de nouveaux itemset candidats.

**Algorithme Apriori**

//$L_k$= ensemble des k-itemset frequents
//$C_k$= ensemble des k-itemset candidats
//D = table des transactions, liste des tickets de caisse

$L_1$ ensemble des 1-itemset frequents
**pour** ($k = 2$ ; $L_{k-1} \neq \{\}$;k++)
  $C_k$ = GenererCandidats($L_{k-1}$);
  **pour tous** transaction $t \in D$
    $C_t$= SousEnsemble($C_k$,t) ;// $C_t$ contient les éléments de $C_k$ contenus dans t
    **pour tous** $c \in C_t$
      c.compt ++;
  $L_k$= { $c \in C_k$ / c.compt $\geq$ MinSprt};
**return** $L_1 \cup L_2 \cup L_3 \cup \ldots \cup L_n$

Si on a les 3-itemset fréquents{a,b,c}{a,b,d}{a,b,e}{a,c,d}{a,c,e}{b,c,d}{b,c,e}et uniquement cela alors on pourra générer les 4-itmest candidats suivants :

- {a,b,c,d} (grâce a {a,b,c}{a,b,d}{a,c,d}{b,c,d})
- {a,b,c,e} (grâce a {a,b,c}{a,b,e}{a,c,e}{b,c,e})

Mais on ne pourra pas construire les 4-itemset candidats suivants :

- {a,b,d,e} (parce que {a,d,e}{b,d,e} ne sont pas des itemsets fréquents)
- {a,c,d,e} (parce que {a,d,e}{c,d,e} ne sont pas des itemsets fréquents)
- {b,c,d,e} (parce que {b,d,e}{c,d,e} ne sont pas des itemsets fréquents)

Exemple de construction d'itemsets

## II-3-3) Algorithme AprioroTID

L'algorithme apriori de recherche de règles nécessite *n* passes dans la table, *n* étant le nombre d'items dans le plus grand itemset candidats. Agrawal a proposé une amélioration de cet algorithme en 1994. Le principe est le même. Pour chaque tuple, on associe un identifiant TID (Tuple IDentifier). Lorsque on fait la première passe on crée pour chaque 1-itemset candidat un ensemble de TID vide TIDSet . Lorsque l'on parcourt la table on compte le nombre d'occurrence de chaque itemset (comme pour le premier algorithme) et chaque fois qu'un itemset candidat $\mathbf{I_c}$ apparaît dans un tuple **t** on met le $TID_t$ de ce tuple dans l'ensemble $TIDSet_c$ de cette itemset. A la deuxième phase, on calcule l'ensemble des itemset candidats de la même manière que dans l'algorithme précédant. Mais pour compter les nombres d'occurrences des nouveaux itemset candidats, plus besoin de parcourir la table. Pour chaque nouveau (k+1)-itemset candidat, on calcule son TIDSet à partir de l'intersection des TIDSet des k-itemset qui l'on engendré. Une fois les TIDSet des nouveaux itemset candidats sont calculés, il suffit de compter le nombre d'éléments dans chaque TIDSet pour connaître le nombre d'occurrence de ces nouveaux itemset dans la table.
Ce nouvel algorithme exige que l'espace mémoire soit assez grand pour pouvoir contenir tous les TID lors des calculs, autrement les performances se dégradent.

Les TIDSet nécessitent un très grand espace mémoire lors des premières passes et moins d'espace lors des dernières passes, vu que le support pour les k-itemset est généralement inversement proportionnel à k . Agrawal a proposé une hybridation des algorithmes Apriori et ArioriTID où il propose d'utiliser Apriori lors des premières passes et AprioriTID lors des dernières passes.

## II-3-4) Algorithme Partitionné

Cet algorithme a été proposé par Savasere en 1995 [38] et résout le problème de l'espace mémoire. L'algorithme divise la table des tuples en **D** partitions puis calcule les itemset fréquents pour chaque partition. L'exécution se fait en deux phases. Lors de la première phase, on cherche tous les itemset fréquents (1-itemset, 2-itemset, …etc.) dans chaque partition prise indépendamment des autres partitions. Les itemset ainsi obtenus sont dit localement fréquents. Dans la deuxième phase, on construit un ensemble d'itemset candidats globaux constitués de tous les itemsets localement fréquents. Avec cet ensemble d'itemset candidats, on effectue un dernier parcours complet de la table des tuples. Ce qui permet de ne garder que les itemset globalement fréquents.

Dans son article Savasere a démontré l'exactitude et la complétude de son algorithme. Pour être exacte, l'algorithme ne doit pas donner de faux itemset fréquents. Ce qui est assuré par la deuxième phase. Pour la complétude, la démonstration est faite par l'absurde. Pour cela on cherche à trouver un itemset I globalement fréquent mais qui n'est localement fréquent dans aucune partition.
Soit **I** un itemset globalement fréquent
$P_i$ la partition i et Taille($P_i$) le nombre de tuple de cette partition
$Nb_i$(**I**) le nombre de tuple de $P_i$ contenant l'itemset **I**
Supposant que l'itemset **I** n'est localement fréquent dans aucune partition donc

$[Nb_1(\mathbf{I}) / Taille(P_1) < \mathbf{MinSprt}] \wedge [Nb_2(\mathbf{I}) / Taille(P_2) < \mathbf{MinSprt}] \wedge \ldots \wedge [Nb_n(\mathbf{I}) / Taille(P_n) < \mathbf{MinSprt}]$

Donc on aura

$[Nb_1(\mathbf{I}) < Taille(P_1) * \mathbf{MinSprt}] \wedge [Nb_2(\mathbf{I}) < Taille(P_2) * \mathbf{MinSprt}] \wedge \ldots \wedge [Nb_n(\mathbf{I}) < Taille(P_n) * \mathbf{MinSprt}]$

Ce qui donnera aussi

$Nb_1(\mathbf{I}) + Nb_2(\mathbf{I}) + \ldots + Nb_n(\mathbf{I}) < (Taille(P_1) + Taille(P_2) + \ldots + Taille(P_n)) * \mathbf{MinSprt}$

Mais

$(Taille(P_1) + Taille(P_2) + \ldots + Taille(P_n))$ est le nombre de tuple de la table Taille(Table)

et

$Nb_1(\mathbf{I}) + Nb_2(\mathbf{I}) + \ldots + Nb_n(\mathbf{I})$ est le nombre de transaction dans la table contenant l'itemset **I**

Donc on a

$$Nb_{Table}(\mathbf{I}) < Taille(Table) * \mathbf{MinSprt}$$

Ce qui contredit la supposition et termine la démonstration.

L'algorithme de Savasere apporte une meilleure gestion de la mémoire car une taille bien choisie pour les partitions permet de charger chacune d'elles en mémoire centrale en une seule fois. Donc l'accès vers la table sur mémoire secondaire se fera en tous, deux fois. L'algorithme offre aussi l'opportunité de paralléliser le calcul lors de la première phase en traitant chaque partition sur un processeur différent.

L'efficacité de l'algorithme dépend du nombre d'itemset candidat à la deuxième phase. Si ce nombre est trop grand, il générera une charge de travail non négligeable vu que lors de cette phase il faudra gérer autant de compteur que d'itemset candidats. Ce problème est d'autant plus grave lorsque le nombre d'itemset globalement fréquents retenus à la fin est largement inférieur au nombre de candidats car à ce moment là ce ne serait plus une surcharge de calcul mais un gaspillage de ressource. La raison de ce gaspillage est que la majorité des itemset candidats ne sont réellement que localement fréquents. Ceci peut arriver lorsque des phénomènes insolites (exemple : vague de chaleur en plein hivers) viennent perturber le fonctionnement habituelle et engendrer des comportements insolites (achat de crème solaire et crème antigel à la même période). Pour parer à ce problème Savasere a proposé d'emmêler la table des tuples avant de construire les partitions. Cela fait ''diluer'' les phénomènes insolites dans l'ensemble de la table et atténue leurs effet. Ce qui rend les partitions globalement homogènes.

## II-3-5) Algorithme de comptage dynamique (Dynamic Itemset Counting)

Proposé par Brin en 1997 dans **[8]**, cet algorithme est une amélioration de l'algorithme Apriori. Il diminue le nombre de fois ou l'on parcourt la table. L'idée de Brin est de parcourir la table en faisant des arrêts à chaque Q tuples. A chaque arrêt, on génère d'éventuels nouveaux itemsets. Ces derniers peuvent êtres des itemset de même rang ou des itemsets de rangs supérieurs.

L'algorithme commence une première étape en comptant les 1-itemsets candidats. Puis après avoir parcouru Q tuples de la table on s'arrête et on essaye de générer les

2-itemsets candidats. A partir de là, on compte les 1-itemset et les 2-itemset candidats jusqu'au prochain arrêt. Là, on génère les 2-itemsets et les 3-itemsets candidats. A partir de là on continue le parcours en comptant les 1-itemsets, 2-itemsets et 3-itemsets candidats. Et ainsi de suite jusqu'à la fin de la table. Puis on revient au tous début de la table et on continue le parcours après avoir généré les éventuels itemset candidats mais cette fois ci, seul les 2-itemsets, 3-itemsets,…etc. sont comptés. Après Q tuples on s'arrête encore et on génère les éventuels itemsets puis on parcours les Q tuples suivants en comptant uniquement les 3-itemset, 4-itemsets,…etc. et ainsi de suite jusqu'à ce qu'on n'ait plus de itemset candidats et que tous les itemset candidats générés aient parcouru toute la table.

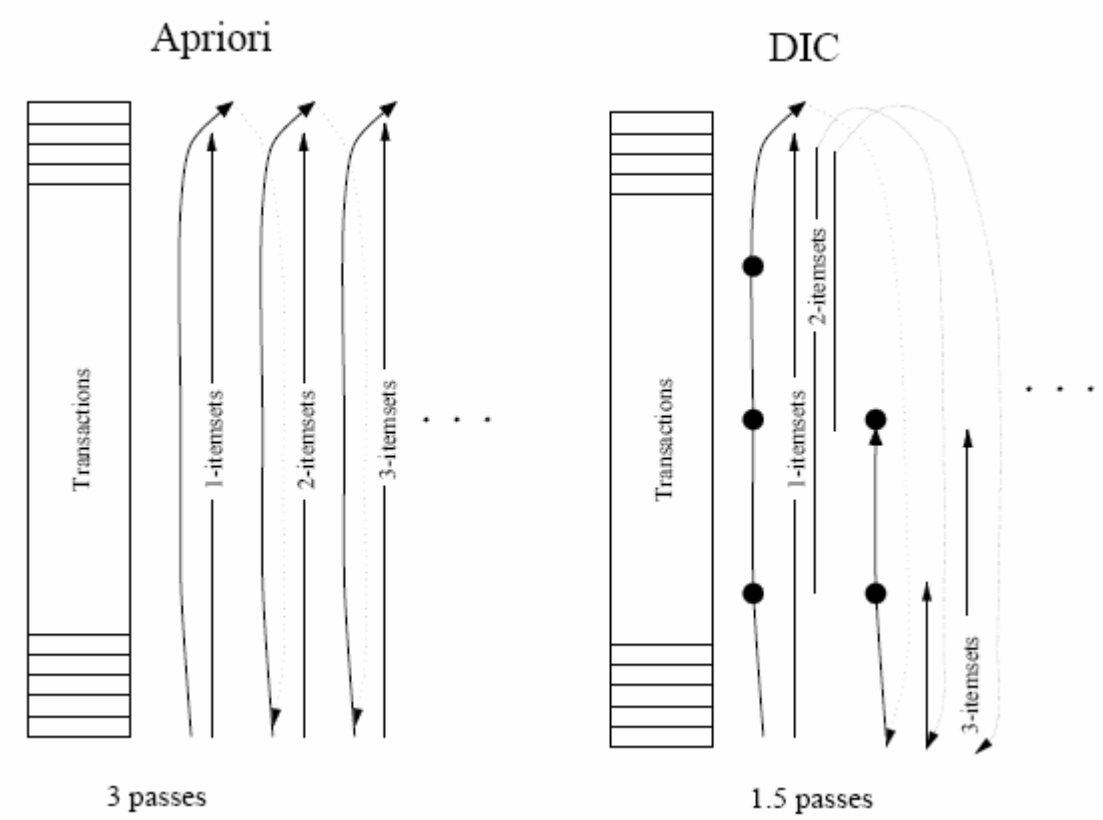



**fig. 2.14 :** Fonctionnement du comptage dynamique

Dans son article Brin a fait la similitude avec un train dont les passagers sont les itemsets candidats. A chaque arrêt, un ensemble d'itemsets montent tandis que d'autres descendent. Un itemset peut au maximum faire un tour complet. Dans le train on trouve 4 types de passagers :

- Les itemset confirmés fréquents : ce sont les itemset qu'on a fini de compter à travers toutes la table (i.e. ceux qui on fait un tour complet) et qui on un support supérieur au seuil *minsup*.
- Les itemsets confirmés non fréquents : ce sont les itemset qu'on a fini de compter à travers toute la table et qui ont un support inférieur au seuil *minsup*.
- Les itemsets suspectés fréquent : ce sont les itemset qu'on a pas encore fini de compter à travers toute la table (i.e. ce qui n'ont pas fait encore un tour complet) et qui ont un support supérieur au seuil *minsup*.

- Les itemset suspectés non fréquents : ce sont les itemset qu'on a pas encore fini de compter à travers toute la table et qui ont un support inférieur au seuil *minsup*.

Lors d'un arrêt, on génère de nouveau (k+1)-itemsets candidats. Pour cela, on utilise seulement les k-itemsets confirmés ou suspectés fréquents. Les nouveaux (k+1)-itemsets sont classés confirmés non fréquents.

Dans la figure 2.14, on voit la différence entre l'algorithme Apriori et l'algorithme de Brin. Avec l'algorithme Apriori (à gauche) il faut 3 passes complètes sur la table pour trouver tous les 1-itemsets, 2-itemsets et 3-itemsets fréquents. Tandis qu'avec l'algorithme de Brin (à droite) il faut une passe et demi pour le même résultat.

L'algorithme de Brin n'est pas toujours aussi efficace. Si les itemsets ont des supports proche de *minsup* et que leur distribution est homogène tous le long de la table alors un itemset candidats devra parcourir presque toute la table avant de pouvoir dire s'il est fréquent ou non. Et donc pour pouvoir générer de nouveaux itemset candidats il faudra à chaque fois parcourir quasiment la totalité de la table des tuples.

## II-3-6) Algorithme BitMap

Cet algorithme a été proposé dans **[9]** en 1998. Comme l'algorithme AprioriTID, l'algorithme bitmap essaye de réduire l'espace mémoire nécessaire au comptage des supports des itemset. L'algorithme se scinde en deux versions, la version naïve N-BM et la version hiérarchique H-BM. Dans la version naïve, on construit une matrice de bits. A chaque tuple (ticket de caisse) correspond une ligne dans la matrice de bits. A chaque k-itemset correspond une colonne dans la matrice des bits. Un bit prend la valeur 1 si le tuple qui correspond à sa ligne contient l'itemset qui correspond à sa colonne. Dans le cas contraire, il prend la valeur 0. De cette manière, une colonne de la matrice indique tous les tuples qui contiennent l'itemset de cette colonne. De même qu'une ligne indique tous les itemset qui appartiennent au tuple de cette même ligne. Le support d'un itemset est donné par le rapport entre le nombre de bits à 1 de sa colonne et le nombre de bits total de cette même colonne.

L'algorithme s'inspire un peu de AprioriTID dans la mesure où il essaye de charger toute l'information nécessaire en mémoire centrale. Cependant avec l'algorithme bitmap l'espace mémoire nécessaire est beaucoup plus petit. Dans l'algorithme AprioriTID on a besoin d'un entier court (soit 16 bits) pour représenter le faite qu'un itemset est contenu dans un tuple (ceci est vrai si le nombre de tuple est compris entre $2^8 = 256$ et $2^{16} = 65536$). Tandis que dans l'algorithme Bitmap, on a besoin d'un bit pour représenter la même information. On voit que le rapport est de 1/16. Évidemment ceci n'est pas tout à fait vrai car dans l'algorithme bitmap, le bit est utilisé que le itemset soit contenu ou pas dans le tuple. Le deuxième avantage de l'algorithme BitMap est dans sa manière de calculer le support des k-itemset (k>1). Grâce au bit, une simple opération logique ''AND'' entre les colonnes de deux k-itemset permet d'obtenir la colonne du nouveau (k+1)-itemset correspondant. Ceci permet un gain de temps considérable dû au faite que l'opération logique est implémentée directement dans le hardware et elle s'exécute sur un ensemble de bits à la fois. Comme il a été dit dans **[9]**, on peut même prévoir des architectures matérielles optimisées pour ce genre d'opérations

La deuxième version de l'algorithme BitMap est la version hiérarchique. Dans cette version l'objectif est de réduire encore plus le temps d'exécution. Comme on l'a déjà signalé plus haut, un bit à 1 est utilisé pour exprimer le faite qu'un itemset soit contenu dans un tuple.

Dans le cas contraire, le bit est à 0. Cependant dans beaucoup de cas pratiques, on se retrouve avec des suites de 0. Ce qui fait qu'on se retrouve à calculer des AND logiques avec des valeurs nulles. Alors pour mieux exploiter le temps de calcul, l'algorithme H-BM étend l'algorithme N-BM en gardant l'index initial $Idx_I$ et en créant un index $Idx_G$ de plus sur chaque colonne de k-itemset (k > 1). Dans l'index $Idx_G$ un bit est associé à un group de bits de l'index $Idx_I$ (par exemple un groupe de 16 bits). Un bit de l'index $Idx_G$ est à 0 si tous les bits du groupe lui correspondants dans l'index $Idx_I$ son à 0 et prend la valeur 1 dans le cas contraire. De cette manière un bit de l'index $Idx_G$ résume en quelque sorte l'information contenue dans un groupe de bits de l'index $Idx_I$ . Lors du calcul de la colonne d'un nouveau (k+1)-itemset un AND logique est calculé d'abord entre les indexes $Idx_G$ des k-itemset puis pour chaque bit non nul dans l'index $Idx_G$ du nouveau (k+1)-itemset, on calcule les valeurs des bits du groupe qu'il résume dans $Idx_I$. Pour chaque bit nul dans $Idx_G$ du nouveau (k+1)-itemset on fait une économie de 16 - 1=15 opérations de AND logique par rapport à l'algorithme N-BM.

Si on a les item A,B,C et les tuples suivant :
$t_1$ ={B}, $t_2$ ={A,B,C}, $t_3$ ={B,C}, $t_4$ ={B,C}, $t_5$ ={A,B}, …, $t_n$ ={A,C}

La représentation avec N-BM correspondante est donnée sur le tableau de gauche (figure 2.15). La représentation en H-BM est donnée sur le tableau de droite. Là, on a choisi de grouper les bits des colonnes en groupe de 4 bits.

| | | Itemsets | | | | | |
|---|---|---|---|---|---|---|---|
| | | A | B | C | AB | AC | ABC |
| Tuples | $t_1$ | 0 | 1 | 0 | 0 | 0 | 0 |
| | $t_2$ | 1 | 1 | 1 | 1 | 1 | 1 |
| | $t_3$ | 0 | 1 | 1 | 0 | 0 | 0 |
| | $t_4$ | 0 | 1 | 1 | 0 | 0 | 0 |
| | $t_5$ | 1 | 1 | 0 | 1 | 0 | 0 |
| | $t_6$ | 0 | 1 | 1 | 0 | 0 | 0 |
| | $t_7$ | 0 | 1 | 0 | 0 | 0 | 0 |
| | $t_8$ | 0 | 1 | 1 | 0 | 0 | 0 |
| | $t_9$ | 1 | 0 | 1 | 0 | 1 | 0 |
| | $t_{10}$ | 1 | 0 | 0 | 0 | 0 | 0 |
| | ⋮ | | | | | | |
| | $t_n$ | 1 | 0 | 1 | 0 | 1 | 0 |

| | | Itemsets | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | A | B | C | AB | | AC | | ABC | | |
| | | | | | **I** | **G** | **I** | **G** | **I** | **G** |
| Tuples | $t_1$ | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| | $t_2$ | 1 | 1 | 1 | 1 | | 1 | | 1 | |
| | $t_3$ | 0 | 1 | 1 | 0 | | 0 | | 0 | |
| | $t_4$ | 0 | 1 | 1 | 0 | | 0 | | 0 | |
| | $t_5$ | 1 | 1 | 0 | 1 | 1 | 0 | 0 | X | 0 |
| | $t_6$ | 0 | 1 | 1 | 0 | | 0 | | X | |
| | $t_7$ | 0 | 1 | 0 | 0 | | 0 | | X | |
| | $t_8$ | 0 | 1 | 1 | 0 | | 0 | | X | |
| | $t_9$ | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| | $t_{10}$ | 1 | 0 | 0 | 0 | | 0 | | 0 | |
| | $t_{11}$ | 0 | 0 | 1 | 0 | | 0 | | 0 | |
| | $t_{12}$ | 1 | 0 | 1 | 0 | | 1 | | 0 | |
| | $t_{13}$ | 0 | 0 | 1 | 0 | | 0 | | 0 | |
| | ⋮ | | | | | | | | | |
| | $t_n$ | 1 | 0 | 1 | 0 | | 1 | | 0 | |

**fig. 2.15** Exemple de représentation avec N-BM et H-BM

Avec H-BM, pour calculer l'intersection AC avec AB, on calcule d'abord l'intersection entre les indexes $Indx_G$ . Puis lorsque un bit dans l'indexe $Indx_G$ de l'itemset ABC est non nul, on calcule les valeur des bits de l'index $Indx_I$ qu'il résume. Sur la figure 2.15 de droite les cases marquées avec X correspond à des cases qu'on a pas besoin de calculer.

## II-3-7) Algorithme de découverte de règles multi niveaux

Ce concept a été introduit dans **[10]** en 1995. Habituellement, les règles d'associations générées sont de la forme A→B où A et B sont des catégories de produits assez générales tel que lait, pain, viande. Mais dans un supermarché, par exemple, on trouve plusieurs sous catégories pour chaque catégorie. Par exemple pour le lait on peut avoir lait 2%, lait écrémé, lait bio, marque x ou y, …etc. Il est évident que des règles plus précises seraient un atout pour les gérants d'un supermarché. Une première idée serait de générer directement des règles de cette forme là. Cela pourra se faire en créant non pas un item pour chaque catégorie générale de produit mais un item pour chaque produit. Le problème dans ce cas là est que pour avoir des item fréquents, il faudra diminuer la valeur de *minsup*. Mais en faisant cela on générera beaucoup de règles insignifiantes. L'auteur de **[10]** a proposé une approche qui permet de générer des règles précises tout en écartant les règles insignifiantes. L'idée est de classer les différents produits dans des hiérarchies de classes (voir figure 2.16). Dans une hiérarchie, on peut trouver plusieurs niveaux. Le premier niveau désigne la catégorie mère, exemple lait, viande. Le deuxième niveau contient des sous catégories exemple pour viande : viande blanche, viande rouge. Le troisième niveau pour viande blanche pourra à son tour avoir les sous catégories : poulet, dinde, canard,…etc.

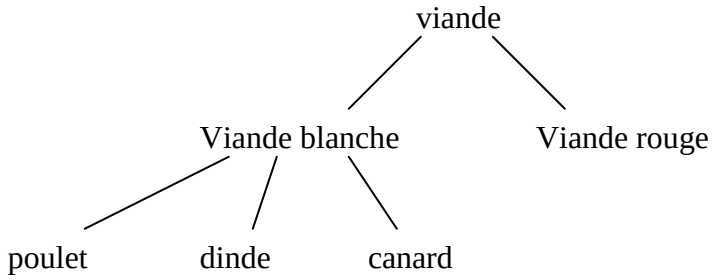



**fig. 2.16 :** Règles hiérarchiques

Une fois la hiérarchie faite, on passe à la génération des règles. Comme dans les algorithmes précédents, on cherche les itemset fréquents puis on génère les règles qui on un support minimal. La recherche des itemset se fait niveau par niveau. Tout d'abord on génère l'ensemble des 1-itemset fréquents de niveau 1 (noté $T$ [2] dans l'article). Puis on utilise cet ensemble pour la génération des k-itemset de tous les niveaux. A chaque niveau on utilise support minimal spécifique $minsup[\ell]$ , tel que :

$$minsup[1] \geq minsup[2] \geq \ldots minsup[\ell\text{-}1] \geq minsup[\ell] \geq \ldots$$

Le fait d'utiliser $T$ [2] pour la génération des k-itmset fréquents de tous les niveaux, assure qu'un k-itemset de niveau $\ell$ est forcement constitué de items issue de catégories mères fréquentes.

Une fois qu'on a tous les k-itemsets de tous les niveaux, il ne reste plus qu'à rechercher les règles dont le support excède *minconf*. Là aussi on a un *minconf* **[** $\ell$ **]** spécifique à chaque niveau. Dans **[10]** plusieurs versions de l'algorithme on été proposé la première est ML-T2, puis pour augmenter les performances ML-T1, ML-Tmax et ML-T2+ on été proposés.

## II-3-8) Algorithme de recherche de règles d'association avec des items pondérés

Lorsque on recherche des règles d'associations, on accorde la même importance à tous les produits. Mais lorsque on analyse un panier de ménagère on peut vouloir s'intéresser uniquement à certains produits. Dans **[11],** il a été proposé d'associer aux produits des poids proportionnels à l'intérêt que leur porte l'analyste. Contrairement à ce que l'on pourra croire il ne suffit pas de garder les produits qui ont les plus grands poids et de faire de la recherche de règles avec. Car une telle façon de faire négligerait les règles ou des produits moins important au regard de l'analyse en cours, influe sur des produits plus importants. Par exemple si on s'intéresse uniquement au produit chocolat, gâteau et bonbon alors on ne pourra pas générer les règles lait → chocolat, lait → gâteau,…etc. Pour générer de telles règles, les algorithmes MINWAL(O) et MINWAL(W) on été proposé dans **[11]**.

Dans MINWAL(O) on utilise le support pondéré *WSup* pour rechercher les règles. Etant donné les items (produits) ($i_1$, $i_2$, $i_3$, …$i_n$), on associe à chacun d'eux respectivement les poids ($w_1$, $w_2$, $w_3$,…, $w_n$). Le support pondéré d'un itemset X est défini par

$$WSup(X) = \left(\sum_{i_j \in X} w_j\right) * support(X) \qquad (2.32)$$

Soit les item a,b,c avec les poids $w_a = 0.6$ ; $w_b = 0.5$ ; $w_c = 0.4$.

Le support de chacun des itemset est :
Compt({a}) = 90; Compt({b}) = 80 ; Compt({c}) = 70 ;
Compt({b,c}) = 50 ; Compt({a,b}) = 50 ; Compt({a,c}) = 55 Compt({a,b,c}) = 40 .

Le support pondéré de chacun des itemset est :
WSup({a}) = 54 ; WSup({b}) = 40 ; WSup({c}) = 28;
WSup({b,c}) = 45 ; WSup({a,b}) = 55 ; WSup({a,c}) = 55 ; WSup({a,b,c}) = 60 .

Soit T le nombre de tuple de la table
Si on fixe le seuil wminsup * T = 60 alors on voit bien que {a,b,c} est un 3-itemset frequent mais que tous les itemset qui sont inclus dedans ne sont pas des itemset frequents.

Exemple de calcul de support pondéré

Un itemset pondéré *X* est fréquent si son support pondéré *Wsup*(*X*) est supérieur à une borne *wminsup*. Dans les algorithmes vu jusqu'à là le calcul des itemset fréquents se base sur le même principe à savoir, calculer les k-itemset fréquents puis à partir de ces derniers générer

les (k+1)-itemset candidats (tout k-itemset contenu dans un (k+1)-itemset fréquent est lui-même un itemset fréquent). Dans le cas des item pondérés, ce principe n'est plus valable à cause des poids (voir l'encadré de la page précédente). Pour remédier à ça, une autre mesure a été introduite, le k-support bound. Cette mesure s'explique comme suit :
Compt(X) est le nombre de tuple contenant le k-itemset X. Pour que X soit un itemset fréquent il faut qu'il vérifie :

$$Compt(X) \geq (wminsup \cdot T)/ W(X); \qquad (2.33)$$

tel que $T$ est le nombre de transactions totales de la table

$W(X)$ le poids de l'itemset $X$ ; $\quad W(X) = \sum_{i_j \in X} w_j$

Soit $I$ l'ensemble de tous les item, supposons que $Y$ est un q-itemset ou q < k. Dans l'ensemble des item $(I - Y)$ restant, soit $M$={ $i_{r1}$, $i_{r2}$, $i_{r3}$, …$i_{r\ k-q}$ } les (k-q) plus lourds item (qui on les plus grands poids). On peut dire que le poids maximal pour n'importe quel itemset contenant $Y$ est

$$W(Y,k) = \sum_{i_j \in Y} w_j + \sum_{m \in M} w_m \qquad (2.34)$$

À partir de ces deux équations, on peut déduire que si un k-itemset X contient le q-itemset Y, le nombre de tuple contenant X pour que ce dernier soit un k-itemset fréquent, ne doit pas être en dessous de

$$B(Y,k) = (wminsup * T) / W(Y,k) \qquad (2.35)$$

L'algorithme MINWAL(O) exploite cette propriété. A chaque fois qu'on est sûr qu'un k-itemest X contenant le q-itemset Y ne peut avoir un support supérieur à $B$(Y,k) on élimine sa candidature à l'ensemble des k-itemset fréquents. Toute foie on garde sa candidature pour les ensembles des j-itemset fréquents (j > k).

Dans MINWAL(W), on utilise une autre mesure qui est le support pondéré normalisé. Cette dernière est défini comme suit :

$$WNSup(X) = (1/k) * (\sum_{i_j \in X} w_j) * support(X); \qquad (2.36)$$

où k est la taille de l'itemset X

Un k-itemset pondéré normalisé X est dit fréquent si son support dépasse un certain seuil *wnminsup* fixé par l'analyste

$$WNSup(X) \geq wnminsup \qquad (2.37)$$

Le k-support bound est redéfini comme suit :

$$B(Y,k) = (k * wnminsup * T) / W(Y,k) \qquad (2.38)$$

Bien que l'algorithme MINWAL(O) puisse être utilisé ici avec cette nouvelle mesure, un autre algorithme plus performant a été proposé, il s'agit de MINWAL(W). Pour cela d'autres notions ont été introduites :

**Définitions**

- Pour un itemset $X = \{x_1, x_2, x_3, \ldots, x_n\}$, soit $w_i$ le plus petit poids dans X. Un itemset $Y = X \cup Z$, ou tous les items de Z ont des poids inférieur à $w_i$ est dit superset d'ordre inférieur de X.
- Un itemset $Y \subset X$, tel que tout item de Y a un poids supérieur ou égale à tous les poids dans X – Y, est dit subset d'ordre supérieur de X.

**Lemme 1**

- Si un itemset Y est fréquent, alors tous subset d'ordre supérieur de Y est aussi fréquent.

**Preuve**

Soit X un subset d'ordre supérieur de Y. La moyenne des poids de X est supérieur ou égale à la moyenne des poids de Y. le support de X est supérieur ou égale au support de Y. De là, on aura

$$WNSup(X) \geq WNSup(Y)$$

**Lemme 2**

- Un (k+1)-itemset fréquent est forcément un superset d'un ou plusieurs k-itemset fréquents.

**Preuve**

Si X est un itemset fréquent, alors tout subset d'ordre supérieur de X est aussi fréquent. Soit x l'item le plus léger (qui a le plus petit poids) de X. Alors Y = X – {x} est un subset d'ordre supérieur de X. Et inversement X est un superset d'ordre inférieur de Y.

Grâce à ces propriétés la génération de l'ensemble des (k+1)-itemset fréquents candidats dans MINWAL(W) est faite à partir des k-itemset fréquents contrairement à MINWAL(O). En effet selon le lemme 2, pour qu'un (k+1)-itemset soit fréquent, il faut déjà qu'au moins un de ses subsets soit fréquent.

## II-3-10) Algorithme de recherche de règles d'association quantitative

Jusqu'ici tous les algorithmes qu'on a cité considèrent uniquement les règles de la forme $X_1 X_2 X_3 \ldots X_n \rightarrow Y$ ou $X_1 X_2 X_3 \ldots X_n$ et Y sont des items. Le comportement d'un item à l'égard d'une règle est binaire : soit que l'item participe à la règle auquel cas il y apparaît, soit il ne participe pas à cette règle auquel cas il n y apparaît pas. Par exemple, dans le cas de l'analyse du panier de la ménagère, l'ensemble d'item est constitué de produit tel que chocolat, lait,…etc.,

Mais si en plus des produits que vend le supermarché, on ajoute d'autres données à l'analyse tel que l'age ou le nombre de points de fidélité (grâce au carte de fidélité ou tout autre moyen) comment est ce que ce nouveau élément pourra t'il être inclus dans les règles. La réponse à cette question a été donnée dans **[12]**. Dans cet article, on a proposé un algorithme qui permet d'extraire des règles où les items sont des triplés de la forme <atrib,min,max> où atrib est un attribut qui prend plusieurs valeur tel que age, salaire, …etc. min est max délimitent un intervalle parmi toutes les valeurs possibles de atrib. Ainsi une règle quantitative a la forme

$<atrib_1,min_1,max_1>$, $<atrib_2,min_2,max_2>$,…, $<atrib_n,min_n,max_n> \rightarrow <atrib_i,min_i,max_i>$

Par exemple

<age,10,14>, <poids,40,60> → <obèse,oui>

L'une des difficultés majeures lors de la génération de telles règles est le choix des intervalles. Par exemple pour age on optera pour le découpage { ] 0, 10] ] 10, 20] … ]90, +∞]} ou bien { ] 0, 9] ] 9, 18] … ]90, +∞]},…etc. On voit bien que le nombre de possibilité est infini. L'article introduit de nouvelles métriques, à savoir la 'K complétude' et le 'R intérêt' qui permettent d'extraire les règles.

## II-4) Conclusion

Dans ce chapitre, nous avons vu plusieurs méthodes de data mining. Certes, on n'a pas présenté toutes les méthodes existantes, mais nous avons essayé de donner un aperçu des différentes techniques. Nous avons aussi essayé de les regrouper dans une certain mesure mais cela n'est pas toujours évident car les concepts sont liées et les frontières mal définies. Ainsi, nous avons présenté des techniques issues des statistiques et de l'intelligence artificielle. Pour la fin nous avons choisi de présenter les règles d'association qui illustre bien le challenge du data mining face au gros volumes de données.

# Chapitre 3 : Présentation des SVM

## 3-1) Introduction

Dans toutes les techniques de data mining présentées jusqu'à maintenant on utilisait le principe de la minimisation du risque empirique. Les SVM (Support Vector Machine) exploitent un principe différent qui est la minimisation du risque structurel. Les recherches remontent aux années 1960. Vapnik et Chervonenkis proposèrent le principe du risque structurel et la VC-dimension pour caractériser la capacité d'une machine d'apprentissage. Mais ce n'est qu'en 1995 **[14]** que les recherches dans ce domaine ont explosées grâce notamment à l'utilisation des fonctions noyau. En effet ces dernières permettent d'augmenter les chances de trouver un bon hyperplan séparateur.

## 3-2) SVM pour la classification

### 3-2-1) SVM pour la classification (cas linéaire)

#### 3-2-1-1) SVM à marge dure

Soit un ensemble d'exemples constitué de deux classes $\{x_i, y_i\}$, $i = 1..n$, $x_i \in R^d$, $y_i \in \{0,1\}$. On suppose que les deux classes d'exemples sont linéairement séparables. On se donne pour tâche la construction d'un hyperplan séparateur $\Delta$. $\Delta$ a pour équation :

$$w \cdot x + b = 0.$$

La fonction de décision sera $f(x) = wx + b$

$f(x) < 0$ si $x \in$ classe 0
$f(x) > 0$ si $x \in$ classe 1

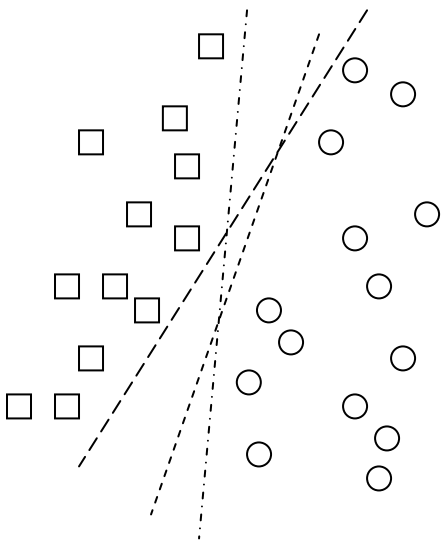


**fig 3.1:** plusieurs séparateurs

Il est évident qu'on peut avoir plusieurs hyperplans séparateurs (voir la figure 3.1). Pour avoir le meilleur séparateur possible, on cherche celui qui s'éloigne le plus des points

(exemples). Soit $\Delta$ cet hyperplan. Soit $d_j$ la distance entre $\Delta$ et le point de la classe j le plus proche de $\Delta$, $j \in \{0,1\}$. On définit la marge comme étant $d_0 + d_1$. L'algorithme SVM cherche à maximiser cette marge.

Supposons que tous les points arrivent à satisfaire les contraintes suivantes :

$w \cdot x_i + b \geq +1$ pour $y_i = +1$
$w \cdot x_i + b \leq -1$ pour $y_i = -1$

Ce qui donne
$y_i (w \cdot x_i + b) \geq +1$

Soit $H_0$ et $H_1$ deux hyperplan parallèles à $\Delta$ qui ont pour équations

$H_0 : wx + b = +1.$
$H_1 : wx + b = -1.$

Si ***o*** est le point origine alors on a les distances suivantes:

$D(\boldsymbol{o}, \Delta) = |b| / \|w\|$
$D(\boldsymbol{o}, H_0) = |b-1| / \|w\|$
$D(\boldsymbol{o}, H_1) = |b+1| / \|w\|$

D'où les distances suivantes

$D(\Delta, H_0) = 1 / \|w\|$
$D(\Delta, H_1) = 1 / \|w\|$

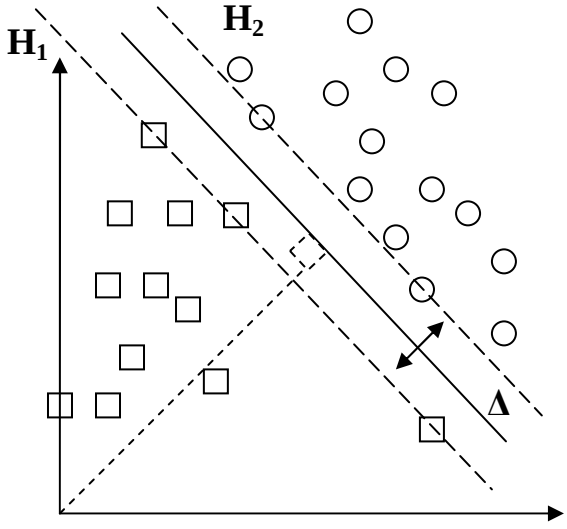



**fig 3.2:** meilleur séparateurs

Si maintenant les plus proches points de $\Delta$ appartiennent au hyperplan $H_0$ et $H_1$ alors on aura $d_0 = d_1 = 1 / \|w\|$ . Et donc la marge est $2 / \|w\|$ .
Pour trouver l'hyperplan séparateur qui maximise la marge, il suffit de trouver l'hyperplan $\Delta$ qui à la fois minimise $\|w\|$ et satisfait l'équation $y_i (w \cdot x_i + b) \geq +1$.

Donc on peut formuler le problème comme un problème d'optimisation quadratique

$$\begin{cases} \text{Min } ½ \, w^2 \\ \text{sous contraintes} \\ \quad y_i (wx_i + b) \geq +1 \quad \forall i = 1 \ldots n \end{cases} \qquad (3.1)$$

On peut changer la formulation de ce problème en utilisant la méthode des multiplicateurs de Lagrange, on aura

$$\begin{cases} \text{Min } L_P \equiv ½ \, w^2 - \sum_{i=1}^{n} \alpha_i [y_i (wx_i + b) - 1] \\ \text{sous contraintes} \\ \quad \alpha_i \geq 0 \quad \forall i = 1 \ldots n \end{cases} \qquad (3.2)$$

Le minimum de $L_P$ se trouve sur le point selle. Pour trouver ce point il faut minimiser $L_P$ par rapport à w et b et le maximiser par rapport à $\alpha_i$ . Le point selle vérifie les conditions de (Karush 1939, Kuhnand Tucker 1951) KKT qui sont les suivantes :

$$\partial L_P / \partial w = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \quad \Leftrightarrow \quad w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\partial L_P / \partial b = - \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\alpha_i [y_i (w x_i + b) - 1] = 0$$

En injectant ces résultats dans $L_P$ On obtient :

$$L_D (\alpha) \equiv \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j$$

Pour avoir le point selle, il faut maximiser $L_D (\alpha)$ par rapport à α. On obtient donc la formulation finale du problème :

$$\begin{cases} \text{Max } L_D (\alpha) \equiv \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j \\ \text{sous contraintes} \\ \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \quad \alpha_i \geq 0 \qquad \forall i = 1 \ldots n \end{cases} \qquad (3.3)$$

les conditions de KKT associées

$y_i (w \cdot x_i + b) - 1 \geq 0 \qquad \forall i = 1 \ldots n$

$\alpha_i [y_i (w \cdot x_i + b) - 1] = 0 \qquad \forall i = 1 \ldots n$

Après avoir optimisé $L_D (\alpha)$, on trouve beaucoup de $\alpha_i$ égaux à zéro. Les $\alpha_i$ qui sont différents de zéro, correspondent à des points (exemples) qu'on appel vecteur de support. Les vecteurs de support sont les points les plus proches de l'hyperplan séparateur.
Après avoir maximisé $L_D (\alpha)$, on peut tirer la valeur de w et b en utilisant uniquement les vecteurs de support comme suit :

$$w = \sum_{S=1}^{ns} \alpha_S y_S x_S$$ **s** parcourt uniquement les vecteurs de support

$$b = (1/ns) \sum_{S=1}^{ns} (1/y_S - w x_S)$$ ns est le nombre de vecteurs de support

### 3-2-1-2) SVM avec marge molle

Soit le même ensemble de points $\{x_i, y_i\}$ décrit précédemment. Cependant, ici on suppose que les deux catégories de points ne sont pas linéairement séparables. Dans ce cas, si on applique l'algorithme précédant, on ne risque pas de trouver de résultat. Pour pallier à cet

handicape, Corinna Cortes et Vladimir Vapnik **[14]** ont proposé un moyen pour rendre les contraintes plus souples.

$$\begin{aligned} wx_i + b &\geq +1 - \xi_i \quad \text{pour } y_i = +1 \\ wx_i + b &\leq -1 + \xi_i \quad \text{pour } y_i = -1 \\ \xi_i &\geq 0 \quad \forall\, i = 1 \ldots n \end{aligned}$$

Pour qu'un point soit mal placé par rapport à l'hyperplan, il faut que $\xi_i > 1$. On peut quantifier les erreurs commises par un hyperplan avec $\sum_{i=1}^{n} \xi_i$ . On peut reformuler maintenant le problème comme suit :

$$\begin{cases} \text{Min } \tfrac{1}{2}\, w^2 + C \left(\sum_{i=1}^{n} \xi_i\right) & C \text{ est un paramètre qui pondère la somme des erreurs} \\ \text{sous contraintes} \\ \quad y_i\,(wx_i + b) \geq +1 - \xi_i & \forall\, i = 1 \ldots n \\ \quad \xi_i \geq 0 & \forall\, i = 1 \ldots n \end{cases} \qquad (3.4)$$

On passe à nouveau au Lagrangien :

$$\begin{cases} \text{Min } L_P \equiv \tfrac{1}{2}\, w^2 + C \left(\sum_{i=1}^{n} \xi_i\right) - \sum_{i=1}^{n} \alpha_i \left[y_i\,(wx_i + b) - 1 + \xi_i\right] - \sum_{i=1}^{n} \mu_i \xi_i \\ \text{sous contraintes} \\ \quad \alpha_i \geq 0 \quad \forall\, i = 1 \ldots n \\ \quad \mu_i \geq 0 \quad \forall\, i = 1 \ldots n \end{cases} \qquad (3.5)$$

Les conditions de KKT pour ce cas sont :

$$\partial L_P / \partial\, w = w - \sum_{i=1}^{n} \alpha_i\, y_i\, x_i = 0$$

$$\Leftrightarrow \quad w = \sum_{i=1}^{n} \alpha_i\, y_i\, x_i$$

$$\partial L_P / \partial\, b = - \sum_{i=1}^{n} \alpha_i\, y_i = 0$$

$$\partial L_P / \partial\, \xi_i = C - \alpha_i - \mu_i = 0$$

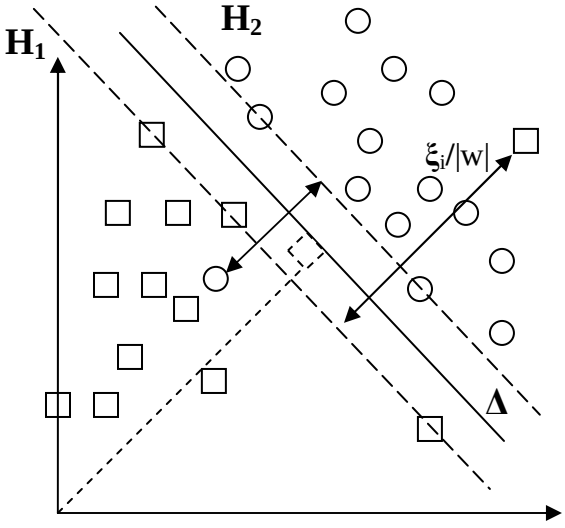



**fig 3.3:** séparateur avec marge molle

et les conditions de complémentarités

$$
\begin{array}{ll}
y_i\,(wx_i + b) - 1 + \xi_i \geq 0 & \forall\, i = 1 \ldots n \\
\xi_i \geq 0 & \forall\, i = 1 \ldots n \\
\alpha_i \geq 0 & \forall\, i = 1 \ldots n \\
\mu_i \geq 0 & \forall\, i = 1 \ldots n \\
\alpha_i\,[y_i\,(wx_i + b) - 1 + \xi_i] = 0 & \forall\, i = 1 \ldots n \\
\mu_i\,\xi_i = 0 & \forall\, i = 1 \ldots n
\end{array}
$$

A partir des équations $C - \alpha_i - \mu_i = 0$ et $\mu_i\,\xi_i = 0$ on conclue que si $\alpha_i < C$ alors $\xi_i = 0$.
Avec un peu d'algèbre élémentaire, on obtient la formulation duale suivante:

$$
\left\{
\begin{array}{l}
\text{Max } L_D\,(\alpha) \equiv \displaystyle\sum_{i=1}^{n} \alpha_i \; - \tfrac{1}{2} \sum_{i,j=1}^{n} \alpha_i\,\alpha_j\,y_i\,y_j\,x_i\,x_j \\
\text{sous contraintes} \\
\qquad 0 \leq \alpha_i \leq C \\
\qquad \displaystyle\sum_{i=1}^{n} \alpha_i\,y_i = 0 \\
\\
\qquad y_i\,(wx_i + b) - 1 + \xi_i \geq 0 \qquad \forall\, i = 1 \ldots n \\
\qquad \xi_i \geq 0 \qquad \forall\, i = 1 \ldots n \\
\qquad \alpha_i \geq 0 \qquad \forall\, i = 1 \ldots n \\
\qquad \mu_i \geq 0 \qquad \forall\, i = 1 \ldots n \\
\qquad \alpha_i\,[y_i\,(wx_i + b) - 1 + \xi_i] = 0 \qquad \forall\, i = 1 \ldots n \\
\qquad \mu_i\,\xi_i = 0 \qquad \forall\, i = 1 \ldots n
\end{array}
\right. \qquad (3.6)
$$

Une fois optimisé, il devient facile de calculer w et b.

## 3-2-2) SVM pour la classification (cas non linéaire)

Dans beaucoup de problèmes réels, il est impossible de trouver un hyperplan séparateur même en utilisant le deuxième algorithme. Pour étendre les capacités de l'algorithme précédent et pouvoir le porter à des problèmes plus vaste, on utilise une transformation non linéaire de l'espace d'entrée vers un autre espace de très grande dimension **[14] [18] [20]**. La très grande dimension de ce nouvel espace augmente les chances de trouver un hyperplan séparateur.

$$
\begin{array}{ll}
\Phi : & R^B \longrightarrow R^H \\
& x \longrightarrow \Phi(x)
\end{array}
$$

La dimension de $R^H$ est tellement grande (peut aller jusqu'à l'infini) qu'il devient impossible de calculer $\Phi(x)$ directement. Pour remédier à cet handicape, on utilise une astuce due à N. Aronszajn (1950). On utilise les fonctions noyaux (kernel en Anglais)

$$K(a,b) = \Phi(a)\,\Phi(b)$$

Puisque dans la formulation de $L_D$, les $x_i$ n'apparaissent que sous forme de produit, il suffit de les remplacer par les fonctions noyaux comme suit :

$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \Phi(x_i) \Phi(x_j)$$
$$= \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \qquad (3.7)$$

Grâce à cet astuce, on peut passer de l'espace original $R^B$, à n'importe quel espace $R^H$, à condition toutes fois de trouver la fonction $K(.,.)$ qui va avec la transformation non linéaire $\Phi(.)$ utilisée. En faite, on a même pas besoin de connaître $\Phi(.)$, il suffit d'avoir une fonction $K(.,.)$ qui correspond à une fonction noyau. On peut donner un exemple de fonction noyau simple :

$$\Phi : \quad R^2 \longrightarrow R^3$$
$$x = (x_1, x_2) \longrightarrow \Phi(x_1, x_2) = ((x_1)^2, \sqrt{2} * x_1 * x_2, (x_2)^2) \qquad (3.8)$$

Ou $(x_1, x_2)$ sont les coordonnées de x

Maintenant, si on fait le produit des points $\Phi(x)$ et $\Phi(x')$ dans l'espace engendré par $\Phi(\ )$ on obtiendra cela

$$\Phi(x) * \Phi(x') = ((x_1)^2, \sqrt{2} * x_1 * x_2, (x_2)^2) * ((x'_1)^2, \sqrt{2} * x'_1 * x'_2, (x'_2)^2)$$
$$= ((x_1)^2 * (x'_1)^2, 2 * x_1 * x_2 * x'_1 * x'_2, (x_2)^2 * (x'_2)^2)$$
$$= ((x_1 * x'_1)^2, 2 * (x_1 * x'_1) * (x_2 * x'_2), (x_2 * x'_2)^2)$$
$$= ((x_1 * x'_1) + (x_2 * x'_2))^2$$
$$= (x * x')^2$$

On voit à travers cet exemple que l'on peut tout à fait calculer $\Phi(x) * \Phi(x')$ sans connaitre la fonction $\Phi(\ )$.

Il existe une infinité de fonction qui peuvent servir de noyau. Pour qu'une fonction $K$ soit une fonction noyau elle doit satisfaire la condition de (Mercer 1909), qui suit:

Une fonction $K(x,y)$ est une fonction noyau si et seulement si pour toute fonction $f(\ )$ tel que $\int f(x)^2 dx$ est finie, on a $\int K(x,y) f(x) f(y) dx dy \geq 0$.

Le problème avec cette définition est que l'on n'a aucune indication sur la méthode à adopter pour construire une fonction noyau. Donc au lieu de cela on utilise des fonctions désormais standard **[16]**:

| | |
|---|---|
| Lineaire | $K(x,y) = (x * y)$ |
| Polynomial | $K(x,y) = ((a * x * y) + b)^d$ |
| Gaussiene | $K(x,y) = \exp(-\|\|x - y\|\|^2 / \sigma^2)$ |
| Laplacien | $K(x,y) = \exp(-\|\gamma\| \, \|\|x - y\|\|)$ |

Mais si pour une raison ou pour une autre on a besoin d'une autre fonction, on exploite le plus souvent une propriété des fonctions noyau qui nous évite de recourir à des démonstrations mathématiques fastidieuses. Cette propriété dit qu'en combinant deux fonctions noyau, on

obtient toujours une fonction noyau. Cette propriété permet notamment de construire des fonctions noyau plus complexes.

Dans la version du SVM pour le cas linéaire la fonction de décision était $f(x)=wx + b$. nous avons vu précédemment comment calculer w et b. Dans le cas présent nous allons avoir

$$w = \sum_{S=1}^{ns} \alpha_S \, y_S \, \Phi(x_S)$$

**s** parcourant uniquement les vecteurs de support (3.9)

Si la dimension de $R^H$ est trés grande ou infinie, il devient impossible de calculer w. Ici aussi l'astuce des fonctions noyau intervient. Au lieu de calculer w on le laisse tel qu'il est (somme de produit) et on tire avantage de l'égalité $K(a,b) = \Phi(a)\,\Phi(b)$

$$f(x)= \left( \sum_{S=1}^{ns} \alpha_S \, y_S \, \Phi(x_S)\, \Phi(x) \right) + b = \left( \sum_{S=1}^{ns} \alpha_S \, y_S \, K(x_S , x) \right) + b \qquad (3.10)$$

de la même façon on calcule b

$$b = (1/ns) \left( \sum_{S=1}^{ns} (1/y_S - \left( \sum_{S=1}^{ns} \alpha_S \, y_S \, \Phi(x_S)\, \Phi(x_S) \right) \right)$$

$$= (1/ns) \left( \sum_{S=1}^{ns} (1/y_S - \left( \sum_{S=1}^{ns} \alpha_S \, y_S \, K(x_S , x_S) \right) \right) \qquad (3.11)$$

# 3-3) SVM pour la régression

## 3-3-1) SVM pour la régression (cas linéaire)

Les SVM ont été adapté à la régression par Vapnik dans **[15]** en 1995 **[13]**. Etant donnée un ensemble de points, exemples $\{\boldsymbol{x_i},\boldsymbol{y_i}\}$, $\boldsymbol{i}= 1..\boldsymbol{n}$ , $\boldsymbol{x_i} \in \boldsymbol{R}^{\boldsymbol{d}}$, $\boldsymbol{y_i} \in \boldsymbol{R}$. On cherche à trouver une fonction $f(\,)$ qui va reconstituer le plus fidèlement possibles les $\boldsymbol{y_i}$ . Comme il est pratiquement impossible de trouver une fonction $f(\,)$ fidèle à cent pour cent (à cause notamment du bruit), Vapnik a introduit une fonction de perte qui tolère une certaine déviation de $f(\,)$ par rapport au données d'origines.

La fonction de pertes utilisé est comme suit :

$$L(x) = |\, x \,|_\varepsilon = \begin{cases} 0 & \text{si } |\, x \,| < \varepsilon \\ |\, x - \varepsilon| & \text{si non} \end{cases}$$

Cette fonction de perte possède un paramètre **ε** qui règle sa sensibilité aux erreurs. Plus **ε** est grand, plus les erreurs (ou écarts) tolérées sont grandes aussi. Un peu comme si on choisissait un pinceau pour dessiner la fonction $f(\,)$ , plus le pinceau est épais plus le nombre d'erreurs (points qui sort du trait) est petit.

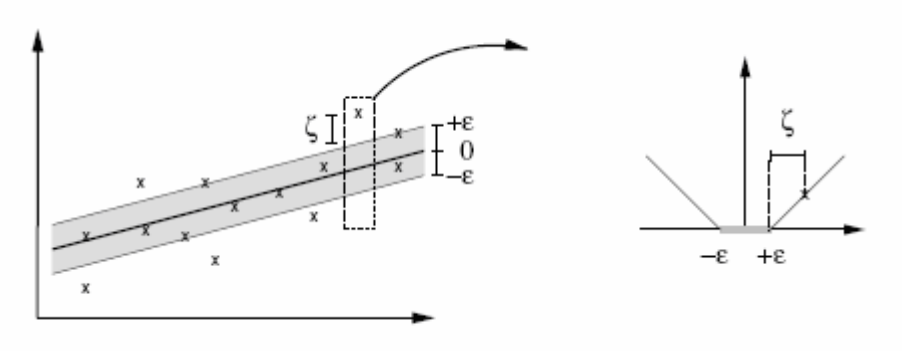



**fig 3.4:** forme de la fonction de perte

La formulation mathématique du problème :

$$\begin{cases} \text{Min } ½\ w^2 \\ \\ \text{sous contraintes} \\ \quad y_i - (wx_i + b) \le \varepsilon \qquad \forall\, i = 1 \ldots n \\ \quad (wx_i + b) - y_i \ge \varepsilon \qquad \forall\, i = 1 \ldots n \end{cases} \qquad 3.12$$

Cependant même avec cette fonction de perte, il peut y avoir des points aberrants qui vont rendre la régression impossible. Pour éviter ce problème, Vapnik a ajouté un facteur qui permet une certaine marge d'erreur. Dans cette nouvelle fonction objective, un facteur ***C*** permet de pondérer cette marge. Plus la constante ***C*** est grande moin la marge d'erreur admise est importante.
La formulation mathématique du problème devient :

$$\begin{cases} \text{Min } ½\ w^2 + C\,(\sum_{i=1}^{n}(\xi_i + \xi_i') \qquad C \text{ pondère la somme des erreurs} \\ \\ \text{sous contraintes} \\ \quad y_i - (wx_i + b) \le \varepsilon + \xi_i \quad \forall\, i = 1 \ldots n \\ \quad (wx_i + b) - y_i \ge \varepsilon + \xi_i' \quad \forall\, i = 1 \ldots n \\ \quad \xi_i\ ,\ \xi_i' \ge 0 \qquad\qquad \forall\, i = 1 \ldots n \end{cases} \qquad 3.13$$

En passant au Lagrangien on obtient :

$$\begin{cases} \text{Min } L_P \equiv ½\ w^2 + C \sum_{i=1}^{n}(\xi_i + \xi_i') - \sum_{i=1}^{n}((\mu_i\ \xi_i) + (\mu_i'\ \xi_i')) - \sum_{i=1}^{n}\alpha_i(\varepsilon + \xi_i - y_i + wx_i + b) \\ \quad - \sum_{i=1}^{n}\alpha_i'(\varepsilon + \xi_i' + y_i - wx_i - b) \\ \\ \text{sous contraintes} \\ \quad \mu_i', \mu_i\ ,\ \alpha_i\ ,\ \alpha_i' \ge 0 \end{cases}$$

Les conditions de KKT pour ce problème :

$$\partial L_P / \partial\, w = w - \sum_{i=1}^{n} (\alpha_i - \alpha_i')\, x_i = 0$$

$$\Leftrightarrow \quad w = \sum_{i=1}^{n} (\alpha_i - \alpha_i')\, x_i$$

$$\partial L_P / \partial\, b = - \sum_{i=1}^{n} (\alpha_i - \alpha_i') = 0$$

$$\partial L_P / \partial\, \xi_i = C - \alpha_i - \mu_i = 0$$

$$\partial L_P / \partial\, \xi_i' = C - \alpha_i' - \mu_i' = 0$$

Et les conditions de complémentarité sont

$$\begin{array}{ll}
\varepsilon + \xi_i - y_i + w x_i + b \geq 0 & \forall\, i = 1 \ldots n \\
\varepsilon + \xi_i' + y_i - w x_i - b \geq 0 & \forall\, i = 1 \ldots n \\
\xi_i \geq 0\ ,\ \xi_i' \geq 0 & \forall\, i = 1 \ldots n \\
\alpha_i \geq 0\ ,\ \alpha_i' \geq 0 & \forall\, i = 1 \ldots n \\
\mu_i \geq 0\ ,\ \mu_i' \geq 0 & \forall\, i = 1 \ldots n \\
\alpha_i(\varepsilon + \xi_i - y_i + w x_i + b) = 0 & \forall\, i = 1 \ldots n \\
\alpha_i'(\varepsilon + \xi_i' + y_i - w x_i - b) = 0 & \forall\, i = 1 \ldots n \\
\mu_i \xi_i = 0\ ,\ \mu_i' \xi_i' = 0 & \forall\, i = 1 \ldots n
\end{array}$$

En prenant les équations : ($C - \alpha_i - \mu_i = 0$ et $\mu_i \xi_i = 0$ ) on obtient $(C - \alpha_i)\, \xi_i = 0$
($C - \alpha_i' - \mu_i' = 0$ et $\mu_i' \xi_i = 0$ ) on obtient $(C - \alpha_i')\, \xi_i' = 0$

En substituant les égalités si haut dans $L_P$ on obtient, moyennant un peu d'algèbre:

$$\left\{
\begin{array}{l}
\text{Max } L_D \equiv -\tfrac{1}{2} \displaystyle\sum_{i=1,\ j=1}^{n} (\alpha_i - \alpha_i')(\alpha_j - \alpha_j')\, x_i x_j - \varepsilon \sum_{i=1}^{n} (\alpha_i + \alpha_i') + y_i \sum_{i=1}^{n} (\alpha_i - \alpha_i') \\
\text{sous contraintes} \\
\quad \displaystyle\sum_{i=1}^{n} (\alpha_i - \alpha_i') = 0 \\
\quad \begin{array}{ll}
0 \leq \alpha_i\ ,\ \alpha_i' \leq C & \forall\, i = 1 \ldots n \\
\varepsilon + \xi_i - y_i + w x_i + b \geq 0 & \forall\, i = 1 \ldots n \\
\varepsilon + \xi_i' + y_i - w x_i - b \geq 0 & \forall\, i = 1 \ldots n \\
\xi_i \geq 0 & \forall\, i = 1 \ldots n \\
\alpha_i \geq 0 & \forall\, i = 1 \ldots n \\
\alpha_i(\varepsilon + \xi_i - y_i + w x_i + b) = 0 & \forall\, i = 1 \ldots n \\
\alpha_i'(\varepsilon + \xi_i' + y_i - w x_i - b) = 0 & \forall\, i = 1 \ldots n \\
(C - \alpha_i)\, \xi_i = 0 & \forall\, i = 1 \ldots n \\
(C - \alpha_i')\, \xi_i' = 0 & \forall\, i = 1 \ldots n
\end{array}
\end{array}
\right. \qquad (3.14)$$

A partir de l'équation $(C - \alpha_i)\ \xi_i = 0$ et $(C - \alpha_i')\ \xi_i' = 0$ on conclue que seule les points pour lesquels la fonction de perte est non nulle (les points qui dépassent le trait décrivant la régression) ont leurs cœfficients $\alpha_i = C$ ou $\alpha_i' = C$. De plus vu qu'un point ne peut dépasser que d'un coté du trait (tracé de la régression), alors $\forall i\ \alpha_i * \alpha_i' = 0$

En résolvant ce problème d'optimisation on obtient ici aussi beaucoup de paire $(\alpha_i , \alpha_i')$ qui sont nulles. Les paires $(\alpha_i , \alpha_i')$ qui ne le sont pas sont les vecteurs de support. Une fois optimisé on calcule w et b

$$w = \sum_{i=1}^{n} (\alpha_i - \alpha_i')\ x_i$$

b peut être calculé en calculant un $b_i$ pour chaque $x_i$ et en calculant la moyenne de ces $b_i$.

Et finalement on obtient la régression suivante :

$$f(x) = w*x + b\ ;$$

## 3-3-2) SVM pour la régression (cas non linéaire)

Tous comme pour le cas de la classification les SVM pour la régression ont un plus grand intérêt lorsque on est face à des données dont la régression linéaire est impossible. L'adaptation de la régression avec SVM pour le cas non linéaire est toute aussi facile que pour le cas de la classification. Ici aussi, on utilise les fonctions noyau pour passer d'un espace de faible dimension à un espace de très grande dimension.

On peut reprendre la démonstration précédente (régression linéaire) et au lieu de la faire dans l'espace de faible dimension $R^B$ on la fait dans l'espace de haut dimension $R^H$ en utilisant $\Phi(x_i)$ au lieu $x_i$ . On aboutira à la formulation **[13]** :

$$\text{Max } L_D \equiv -\tfrac{1}{2} \sum_{i=1,\ j=1}^{n} (\alpha_i - \alpha_i')(\alpha_j - \alpha_j')\,\Phi(x_i)\Phi(x_j) - \varepsilon \sum_{i=1}^{n} (\alpha_i + \alpha_i') + y_i \sum_{i=1}^{n} (\alpha_i - \alpha_i')$$

En prenant la fonction noyau $K(a,b) = \Phi(a)\,\Phi(b)$, on obtient la formulation finale

$$\text{Max } L_D \equiv -\tfrac{1}{2} \sum_{i=1,\ j=1}^{n} (\alpha_i - \alpha_i')(\alpha_j - \alpha_j') K(x_i, x_j) - \varepsilon \sum_{i=1}^{n}(\alpha_i + \alpha_i') + y_i \sum_{i=1}^{n}(\alpha_i - \alpha_i')$$

sous contraintes

$$\sum_{i=1}^{n}(\alpha_i - \alpha_i') = 0$$

$$\begin{array}{ll}
0 \le \alpha_i, \alpha_i' \le C & \forall i = 1 \ldots n \\
\varepsilon + \xi_i - y_i + w x_i + b \ge 0 & \forall i = 1 \ldots n \\
\varepsilon + \xi_i' + y_i - w x_i - b \ge 0 & \forall i = 1 \ldots n \\
\xi_i \ge 0 & \forall i = 1 \ldots n \\
\alpha_i \ge 0 & \forall i = 1 \ldots n \\
\alpha_i(\varepsilon + \xi_i - y_i + w x_i + b) = 0 & \forall i = 1 \ldots n \\
\alpha_i'(\varepsilon + \xi_i' + y_i - w x_i - b) = 0 & \forall i = 1 \ldots n \\
(C - \alpha_i)\xi_i = 0 & \forall i = 1 \ldots n \\
(C - \alpha_i')\xi_i' = 0 & \forall i = 1 \ldots n
\end{array} \qquad (3.15)$$

Ici aussi après l'optimisation on obtient beaucoup de paire $(\alpha_i, \alpha_i')$ nulles. Ceci nous est utile dans la formulation de la fonction de régression.

Ici nous avons $f(x) = \Phi(w) * \Phi(x) + b$ et $w = \sum_{i=1}^{n}(\alpha_i - \alpha_i')\Phi(x_i)$

D'où

$$f(x) = \sum_{i=1}^{n}(\alpha_i - \alpha_i')\Phi(x_i) * \Phi(x) + b$$

$$= \sum_{i=1}^{n}(\alpha_i - \alpha_i') K(x_i, x) + b \qquad (3.16)$$

## 3-4) Les fondements théoriques des SVM

Toutes les méthodes de data mining qu'on a vue se basent sur la minimisation du risque empirique. Mais minimiser le risque empirique n'implique pas forcément la minimisation du risque réel.

Supposant qu'on veut apprendre à une machine une certaine tâche. Pour cela nous allons choisir un espace d'hypothèses $H$ défini par la fonction $f(.,.)$ et nous allons chercher à trouver la fonction $f_0(.,.) = f(\alpha_0,.)$ qui va apprendre cette tâche en minimisant le risque empirique. Une fois la fonction $f(\alpha_0,.)$ trouvée, quel est le pouvoir de généralisation de cette dernière? Connaissant le risque empirique $R_{emp}$ , quel est le risque réel $R_{reel}$ de cette fonction ? Vapnik a répondu à cette question en donnant une borne au risque réel avec la formule suivante **[15]** :

$$R_{reel}(\alpha) \leq R_{emp}(\alpha) + \sqrt{\left( \frac{h(\log(2l/h)+1) - \log(\eta/4)}{l} \right)} \qquad (3.17)$$

Cette formule est vraie avec une probabilité au moins égale à 1- $\eta$

Où α est le(s) parametres d'apprentissage
$h$ est la VC dimension
$\eta$ $0 \leq \eta \leq 1$
$l$ le nombre d'exemples d'apprentissage.

## 3-4-1) La VC dimension

La VC dimension (Vapnik-Chovronik) exprime la richesse d'une famille de fonction $f(.,.)$ ou d'un espace d'hypothèses. On peut donner sa définition comme suit :
la VC dimension ***h*** d'une famille de fonction $f(.,.)$ est le nombre maximal de points d'un ensemble S, tel que tous les points de S peuvent être séparés dans toutes les configurations possibles à l'aide de la famille $f(.,.)$. Il faut noter qu'il suffit de trouver un seul ensemble S qui vérifie cette condition.

**Exemple :** la VC dimension d'un perceptron à ***d*** entrées est égale à ***d*** +1.
Pour la famille des perceptrons avec d = 1 entrées, on a h = 2. En effet dans un espace à une dimension on ne peut pas séparer plus de deux points dans toutes les configurations possibles (voir fig 3.5(a)).

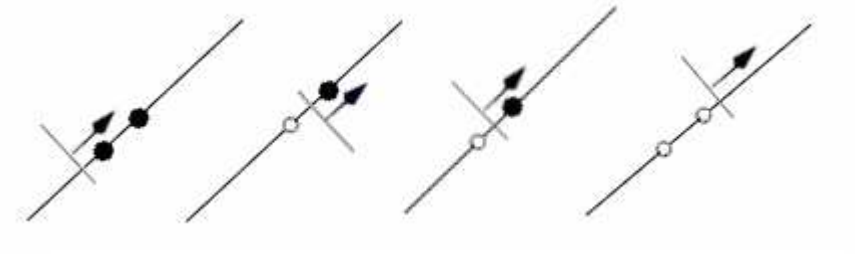


**fig 3.5(a):** VC-dimension d'un perceptron à une entrée

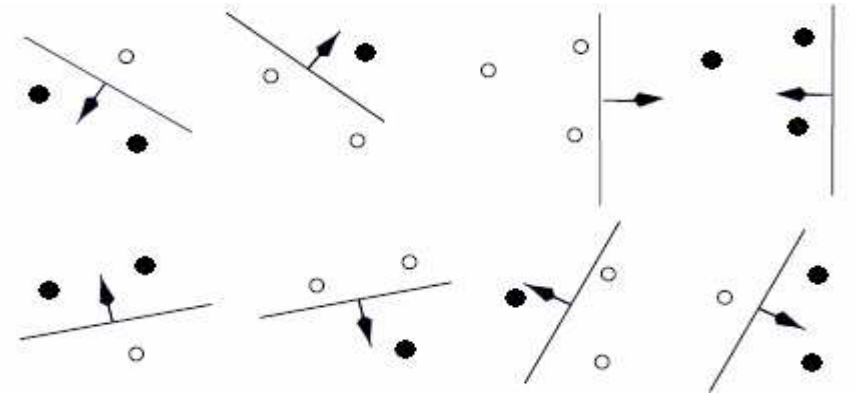


**fig 3.5(b):** VC-dimension d'un perceptron à deux entrées

Pour la famille des perceptrons avec d = 2 entrées, on a h = 3. Ici le séparateur correspond à une droite dans un plan et le nombre maximale de points qu'on peut séparer dans toutes les configurations ne peut pas dépasser trois (voir fig 3.5(b)).

Il faut faire attention à ne pas tomber dans l'erreur de croire que la VC dimension et le nombre de paramètres d'une famille de fonction sont la même chose. Il est vrai que pour l'exemple que nous avons donnée, ces notions sont liées mais on peut tout aussi bien donner une famille de fonction où le nombre de paramètres est limité tandit que la VC dimension est infini. Tel est le cas pour la famille de fonction due à E. Levin et J. S. Denker **[15]** :

$$f(\alpha, x) = 2 \text{ sinus } (\alpha \cdot x) + 0.5 \qquad \alpha \text{ est un réel} \qquad (3.18)$$

**Preuve**

On prend n'importe quel nombre $l$ de points. Soit $\{x_1, x_2, \ldots x_l\}$ ces points.
On choisi ces points tel que $x_i = 10^{-i}$ $i = 1 \ldots l$
Pour n'importe quel étiquetage $y_1, y_2, \ldots, y_l \in \{0, 1\}$ pour que $f(\alpha, x)$ reconstitue cet étiquetage il suffit de prendre :

$$\alpha = \pi(1 + \sum_{i=1}^{l} \frac{(1 - y_i)10^i}{2}). \qquad (3.19)$$

Donc puisque l n'est pas borné, on conclue que effectivement la VC dimension de cette fonction est infinie.

**D'autres mesures de la VC-dimension**

- Les SVM avec noyau RBF ont une VC-dimension infinie.

- Les SVM avec noyau polynomial de degré $p$ et un espace d'entrée de dimensin $d$ ont une VC-dimension égale à : $\left(\binom{d+p-1}{p}+1\right)$

## 3-4-2) Principe de la minimisation du risque structurel (SRM)

Comme le décrit la formule précédente (3.17), la minimisation du risque empirique n'implique pas forcément la minimisation du risque reél. On voit dans la formule que plus le nombre d'exemples augmente plus l'ecart entre $R_{emp}$ et $R_{reel}$ diminue. Plus la VC dimension augmente plus l'ecart entre $R_{emp}$ et $R_{reel}$ augmente. Vapnik (1979) **[17]** a proposé une methodologie qui permet de maximiser le pouvoir de généralisation de la fonction $f(\alpha_0, x)$ apprise. Cette méthodologie exploite un nouveau concept qui est la minimisation du risque structurel. En resumé, Vapnik propose de créer des familles de fonctions imbriquées $f_0(\alpha, x) \subset f_1(\alpha, x) \subset f_2(\alpha, x) \subset \ldots \subset f_q(\alpha, x)$. Puis pour chaque famille de fonction $f_i(\alpha, x)$ on cherche la fonction $f_i(\alpha_0, x)$ qui minimise le risque empirique. Au finale, on prend la fonction $f_q(\alpha_0, x)$ qui minimise la borne sur le risque réel.

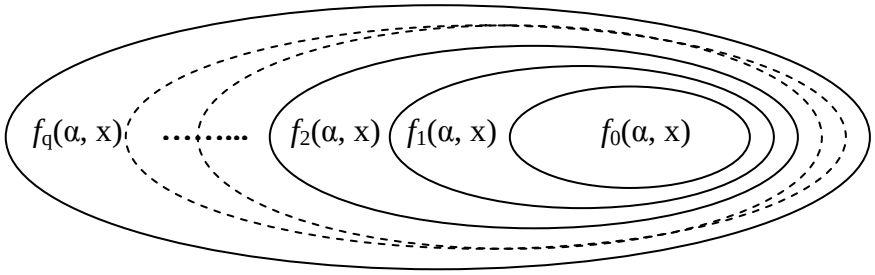



**fig. 3.6:** Imbrication des espaces d'hypothèses

**Définition hyperplan canonique (Vapnik):**

Dans un ensemble d'hyperplans définis par $w \cdot x + b = 0$, tous les hyperplans qui vérifient $| w \cdot x + b | = 1$ sont des hyperplans canoniques,

**Théorème : (Vapnik)**

La VC dimension *h* d'un ensemble d'hyperplans canoniques est borné comme suit :

$$h < \min(d, (D^2M^2)) + 1 \qquad 3.20$$

Où d est la dimension de l'espace des exemples ;
D est le diamètre de la plus petite hypersphère qui englobe tous les exemples ;
M est un réel tel que $2 / \| w \| \leq M$ (c'est la largeur de la marge séparatrice);

Ce théorème donne une borne sur la VC-dimension d'un hyperplan canonique. Pour un espace de petite dimension on voit que la borne *h* est trés petite et de ce fait la borne sur l'écart $R_{reel} - R_{emp}$ l'est aussi. Cependant, dés que l'on passe à un espace de grande dimension par l'utilisation de fonction noyau les deux bornes augmentent drastiquement et cela rend cette borne non significative.

Vapnik a donné une autre mesure qui permet de quantifier le pouvoir de généralisation des SVM. Elle est donnée par le théorème suivant :

**Théorème : (Vapnik)**

Pour un hyperplan optimal passant par l'origine on a la borne suivante :

$$E(P(\text{erreur})) \leq E(D^2/M^2) / \ m \qquad 3.21$$

Où P(erreur) est la probabilité d'erreur sur un ensemble de tests de taille m ;
M est la largeur de la marge séparatrice ;
D est le diamètre de la plus petite hupershpere qui englope tous les exemples ;
$E(D^2/M^2)$ est l'esperence du quotient $D^2/M^2$ calculé sur l'ensemble d'apprentissage de taille (m-1) ;

Une autre borne sur la probabilité de l'erreur a été donnée par Vapnik en 1995. Cette probabilité est calculée pour le cas où la méthode de validation «leave one out» est utilisée.

**Théorème : (Vapnik)**

Soit m le nombre d'exemple d'apprentissage. Soit *s* le nombre de vecteur de support de l'hyperplan optimal séparant les données d'apprentissage. L'espérance de la probabilité d'erreur sur un exemple issue de l'ensemble de tests est borné par :

$$E(P(\text{erreur})) \leq E(s) / ( m - 1 ) \qquad 3.22$$

Ou P(erreur) est la probabilité d'erreur pour une machine entraînée sur un ensemble d'apprentissage de taille m-1 ;

E(P(erreur)) est l'espérance de la probabilité d'erreur P(erreur) sur tout ensemble d'apprentissage de test de taille m-1

E(*s*) est l'espérance du nombre de vecteur *s*

Ce théorème suggère tous simplement que plus le nombre d'exemples augmente plus le pouvoir de généralisation augmente.

## 3-4) Fonctionnement des SVM :

On peut schématiser les SVM sous forme de réseau à couches à la manière des réseaux de neurones **[22]**. Dans le cas des SVM, le nombre de couches est de deux. La première couche contient les vecteurs de support. Ce sont les points de références. La deuxième couche pondère et somme les sorties de la première couche (voir figure ci dessous). Cependant, les SVMs ont de sérieux avantages par rapport au réseau de neurones, et ce malgré la relative jeunesse des recherches les concernant.

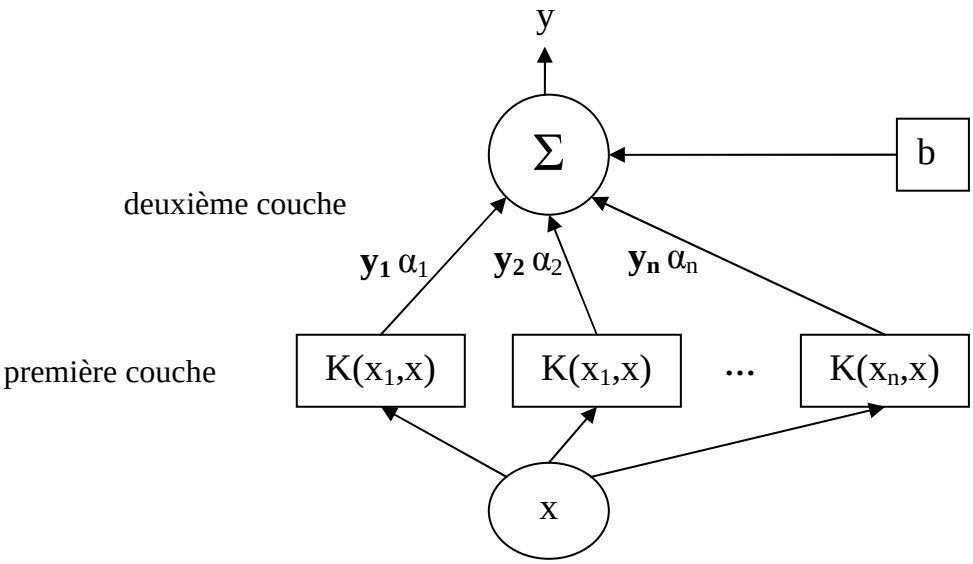



**fig.3.7:** Schéma fonctionnement SVM

Comme premier avantage, nous citons, le nombre de paramètres, dans le cas des réseaux de neurones, il faut fixer au préalable le nombre de couches cachées, le nombre de neurones dans chaque couche, les fonctions de transferts, le pas d'apprentissage, les poids initiales des neurones. Tous ces paramètres sont initialisés de manière expérimental à chaque nouvel apprentissage. Avec autant de paramètres il est ardu de trouver le bon point de départ de l'apprentissage. Du coté des SVM par contre, on n'a que deux paramètres. Comme deuxième avantage, le problème des SVM est un problème d'optimisation quadratique convexe, il n'y a qu'un seul optimum local. Dans le cas des réseaux de neurones, on a plusieurs optima locaux, on ne sait jamais si on a trouvé la solution globale. Comme troisième avantage, les SVMs utilise le principe de minimisation de risque structurel. Les réseaux de neurones utilisent le principe de la réduction du risque empirique.

Bien que les résultats formulés par la théorie ne sont pas très pratique (les bornes sur le risque réel sont très grandes) La suprématie des SVMs a été démontrée dans de nombreux travaux **[16]** et même face aux indétrônables réseaux de neurones **[21] [23]**.

## 3-5) Conclusion

Dans ce chapitre, nous avons vu le fonctionnement des SVM et leurs fondements théoriques. Ainsi nous avons vue le fonctionnement pour la classification et pour la régression respectivement dans le cas linéaire et non linéaire. Puis nous nous somme penché sur les fondements théoriques, à savoir les espaces d'hypothèses, la minimisation du risque structurel,…. Dans le prochain chapitre nous allons voire des algorithmes de mise en œuvre des SVM.

# Chapitre 4 : Algorithmes pour les SVM

## 4-1) Introduction

L'inconvénient majeur dans les SVM est le temps de calcul qu'ils exigent **[16]**. En effet la formulation finale du problème, est un problème d'optimisation quadratique. La formulation matricielle du problème primale est :

$$\begin{cases} \text{Min } ½\ w^T w + C\ 1^T \xi \\ \text{sous contraites} \\ \qquad Y(Xw - 1\ b) + \xi \geq 1 \\ \qquad \xi \geq 0 \end{cases} \qquad (4.1)$$

et la formulation finale pour le cas linéaire est :

$$\begin{cases} \text{Min } ½\ \alpha^T Y X X^T Y \alpha - 1^T \alpha \\ \text{sous contraintes} \\ \qquad 1^T Y \alpha = 0 \\ \qquad 0 \leq \alpha^T 1 \leq C\ 1 \end{cases} \qquad (4.2)$$

Pour le cas non linéaire

$$\begin{cases} \text{Min } ½\ \alpha\ Y\ K(X, X^T)\ Y \alpha - 1^T \alpha \\ \text{sous contraintes} \\ \qquad 1^T Y \alpha = 0 \\ \qquad 0 \leq \alpha^T 1 \leq C\ 1 \end{cases} \qquad (4.3)$$

Où
- $\alpha$ le vecteur constitué des $\alpha_i$
- $\xi$ le vecteur constitué des $\xi_i$
- Y la matrice diagonale constituée des $y_i$
- X la matrice constituée des $x_i$
- $A^T$ la transposé de A
- K(A,B) la fonction noyau appliquée aux matrices A,B ou $K_{ij} = K(A_i , B_j)$

Bien qu'on puisse utiliser des résolveurs génériques pour les problèmes d'optimisation quadratique, ce type de résolveur nous oblige à nous limiter à des ensembles de taille relativement limitée. Cette limitation est due notamment à l'exigence en taille mémoire de ce type de résolveurs. En effet, cette dernière s'accroît quadratiquement avec la taille de l'ensemble de données d'apprentissage. Comme résolveur quadratique, on peut citer CPLEX, MINOS, ou bien LOQO. Pour pallier à cet handicape, des algorithmes de résolution spécialement adaptés au problème des SVM on été proposés.

Dans ce qui suit nous allons présenter quelques différentes approches utilisées pour la résolution du problème de SVM. Nous allons nous limiter au cas de la classification binaire vu que notre travail rentre dans ce cadre. Nous allons tous d'abord voir quelques méthodes de résolution du problème du SVM standard puis nous verrons quelques méthodes qui utilisent d'autre formulation mathématique du problème des SVM et enfin nous verrons des méthodes basées sur la réduction de l'ensemble des données.

# 4-2) Méthodes Basées sur la décomposition

## 4-2-1) Algorithme Chunking

Cet algorithme a été proposé par Vapnik dans **[15]**. Comme l'inconvénient majeur des résolveurs de problèmes quadratiques génériques est l'espace mémoire occupé par la matrice, cet algorithme exploite deux propriétés des SVM :

- Enlever les lignes et colonnes dont les $\alpha_i$ sont nuls ne change rien au problème.
- Une solution $\alpha$ est valide dés qu'elle satisfait les conditions de KKT ;

L'algorithme cherche les $\alpha_i$ nuls et les fait sortir de la résolution. Pour cela, il découpe l'ensemble des $\alpha_i$ en deux : l'ensemble actif et l'ensemble passif. Au début, l'ensemble actif est choisi de manière arbitraire. L'algorithme cherche une solution optimale pour l'ensemble actif. Les $\alpha_i$ nuls sont enlevés de l'ensemble actif. On parcours l'ensemble passif à la recherche des ***m*** exemples qui violent au maximum (avec la plus grande marge d'erreur) les conditions de KKT et on ajoute les $\alpha_i$ leur correspondant à l'ensemble actif. Puis l'optimisation est réitérée sur l'ensemble actif. A chaque itération, l'optimisation est faite à l'aide d'un résolveur de problèmes quadratiques. Ce processus est répété plusieurs fois jusqu'à ce que tous les $\alpha_i$ soit non nuls et que tous les exemples de l'ensemble passif ne violent plus les conditions de KKT. La dernière itération fournie la solution finale.

Cette algorithme fonctionne parfaitement lorsque le nombre de vecteurs de support (exemple dont les $\alpha_i$ sont non nuls) n'est pas trop grand. Autrement la taille de la matrice correspondante à l'ensemble actif ne pourra pas tenir en mémoire.

## 4-2-2) Algorithme de Décomposition

Proposé par Osuna dans **[24]**. Cet algorithme procède aussi par découpage des $\alpha_i$ en deux sous ensembles : actif A et passif B. Contrairement à l'algorithme précédant, la taille de l'ensemble actif est fixe et ne change pas au cour de la résolution. Osuna a constaté et a prouvé la proposition suivante :

**Proposition :**

Etant donnée une solution optimale du sous problème défini par A, remplacer $\alpha_i = 0$ tel que $x_i \in A$, par $\alpha_j = 0$ tel que $x_j \in B$, qui satisfait $y_* f(x_j) < 1$ , ( $f(x_j)$ est la fonction de décision) génère un nouveau sous problème dont l'optimisation abouti a une amélioration de la fonction objective de l'ensemble du problème.

A partir de là, Osuna a proposé son algorithme comme suit :

Soit Data l'ensemble des exemples d'apprentissage

1- Initialiser A de manière aléatoire, B = Data – A ;
2- Résoudre le sous problème constitué de A à l'aide d'un résolveur de problèmes quadratiques ;
3- S'il existe un exemple x de B qui viole les conditions de KKT, enlever un exemple de A et le remplacer par x. si non fin de l'algorithme.

Dans son algorithme, Osuna remplace une seule variable à la fois. Dans la pratique on remplace plusieurs variables, cela permet d'avoir une convergence plus rapide.

### 4-2-2) Algorithme SMO

Proposé dans **[25]**, SMO (pour Sequential Minimal Optimization) peut être vu comme un cas extrême de l'algorithme de Osuna. Ici l'ensemble des $\alpha_i$ actif est limité à deux coefficients. A chaque itération l'algorithme résout un sous problème quadratique de taille deux. Comme la taille du sous problème est réduite l'algorithme utilise une méthode de résolution analytique au lieu d'une méthode de résolution de problème quadratique. Cela fait que l'algorithme exécute rapidement chaque itération. Pour être efficace, l'algorithme doit bien choisir la paire de $\alpha_i$ à optimiser à chaque itération. Dans la littérature, on trouve plusieurs versions du SMO qui apportent principalement des optimisations à cet algorithme **[36] [35][37]**. Parmi les optimisations, on peut citer l'utilisation de caches, les méthodes de sélection des $\alpha_i$, la parallelisation de l'execution ….

## 4-3) Algorithmes Basés sur d'autres formulations mathématiques du séparateur

### 4-3-1) Algorithme kernel Adatron

Cet algorithme a été proposé dans **[26]**. Son fonctionnement ressemble beaucoup a celui du perceptron multicouches. L'algorithme présente aussi l'avantage d'être simple. On réécrit la fonction objective en lui ajoutant la condition $\sum \alpha_i y_i = 0$ comme suit :

$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \lambda \sum_{i=1}^{n} \alpha_i y_i \qquad (4.4)$$

On maximise $L_D(\alpha)$ en utilisant la technique du gradient :

$$\delta\alpha_k = \eta (\partial L_D / \partial \alpha_k) = \eta \left( 1 - y_k \sum_{i=1}^{n} \alpha_j y_j K(x_k, x_j) - \lambda y_k \right) \qquad (4.5)$$

En calculant la variation dans la fonction objective on obtient :

$$\begin{aligned} \delta L_D &= L_D(\alpha + \delta\alpha_k) - L_D(\alpha) \\ &= \delta\alpha_k \left(1 - y_k \sum_{i=1}^{n} \alpha_j y_j K(x_k, x_j) - \lambda y_k \right) - \tfrac{1}{2} (\delta\alpha_k)^2 K(x_k, x_k) \\ &= \left( (1/\eta) - \tfrac{1}{2} K(x_k, x_k) \right) (\delta\alpha_k)^2 \end{aligned} \qquad (4.6)$$

On voit que $\delta L_D > 0$ si on a : $2 > \eta K(x_k, x_k) > 0$

On donne ici la version simple de l'algorithme :

1- initialiser $\alpha_k^0 = 0$
2- pour i =1 a n répéter les étapes 3 et 4
3- calculer $z_i = \sum_{j=1}^{n} \alpha_j y_j K(x_k , x_j)$
4- calculer $\delta\alpha_i^t = \eta (1 - z_i y_i)$ ;
   Si $(\alpha_i^t + \delta\alpha_i^t) \leq 0$ alors $\alpha_i^{t+1} = 0$
   Si non $\alpha_i^{t+1} = \alpha_i^t + \delta\alpha_i^t$
5- si le nombre d'itérations est dépassé ou $\gamma = ½ (Max(z_i^+) - Min(z_i^-))$ est approximativement égale a 1 alors arrêter l'algorithme.
   Ou $Max(z_i^+) = Max(\{z_i \text{ tel que } y_i > 0\})$
   $Min(z_i^-) = Min(\{z_i \text{ tel que } y_i < 0\})$

## 4-3-2) Algorithme SOR SVM

Cet algorithme a été proposé dans **[28]**. SOR pour Successive Over Relaxation utilise une autre formulation mathématique du problème du SVM. La distance entre les marges séparatrices est mesurée dans l'espace à (n + 1) dimensions. La nouvelle formulation est :

$$\begin{cases} \text{Min } ½ (w^2 + b^2) + C (\sum_{i=1}^{m} \xi_i) \text{ ; } C \text{ est un paramètre qui pondère la somme des erreurs} \\ \text{sous contraintes} \\ \quad y_i (w x_i + b) \geq +1 - \xi_i \quad \forall i = 1 \ldots m \\ \quad \xi_i \geq 0 \quad \forall i = 1 \ldots m \end{cases} \qquad (4.7)$$

Le problème dual est:

$$\begin{cases} \text{Max } L_D (\alpha) \equiv \sum_{i=1}^{m} \alpha_i - ½ \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j x_i x_j - ½ \sum_{i=1}^{m} \alpha_i^2 \\ \text{sous contraintes :} \\ \quad 0 \leq \alpha_i \leq C \end{cases} \qquad (4.8)$$

On a les égalités suivantes :

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i$$

$$b = - \sum_{i=1}^{n} y_i \alpha_i$$

$$\xi_i = (1 - y_i (w x_i - b))_+ = Max ( (1 - y_i (w x_i - b)) , 0)$$

Nous allons passer à la formulation matricielle du problème tel que cela a été fait par l'auteur pour exprimer la solution proposée par ce dernier :

$$\begin{cases} \text{Max } L_D (\alpha) \equiv - ½ \alpha^T Y X X^T Y \alpha - ½ \alpha^T Y 1^T 1 Y \alpha + 1^T \alpha \\ \text{sous contraintes} \\ \quad 0 \leq \alpha \leq C 1 \end{cases} \qquad (4.9)$$

Sans oublier les égalités :

$w = X^T Y \alpha$

$b = - 1^T Y \alpha$

$\xi = (1- Y(Xw - 1b))_+$

En définissons :

$H = Y(X - 1)$

$L + E + L^T = HH^T$ ou L est la partie triangulaire basse de la matrice symétrique $HH^T$

E est la diagonale de $HH^T$

Le problème dual peut être reformulé (en passant à un problème de minimisation par la même occasion) à nouveau comme suit:

$$\begin{cases} \text{Min } ½ \| H^T \alpha \|^2 - 1^T \alpha \\ \text{sous contraintes} \\ \qquad 0 \le \alpha \le C\, 1 \end{cases} \qquad (4.10)$$

L'auteur présente la condition nécessaire et suffisante d'optimalité suivante :

$\alpha = (\alpha - w E^{-1} ( H H^T \alpha - 1 ))_\# \; ; w > 0.$

$$\text{Où } ((\alpha)_\#)_i = \begin{cases} 0 & \text{si } \alpha_i \le 0 \\ \alpha_i & \text{si } 0 < \alpha_i < C \\ C & \text{si } \alpha_i \ge C \end{cases} \qquad (4.11)$$

L'algorithme SOR avance à petit pas, à chaque pas, il calcule un nouveau α comme suit :

$$\alpha^{i+1} = (\alpha^i - w E^{-1} ( H H^T \alpha^i - 1 + L (\alpha^{i+1} - \alpha^i )))_\# \qquad (4.12)$$

L'algorithme s'arrête lorsque $\| \alpha^{i+1} - \alpha^i \|$ est inférieur à un certain seuil

Le calcul des $\alpha_j^{i+1}$ se fait dans l'ordre j = 1...m. Le calcul de $\alpha_j^{i+1}$ se fait en fonction de ($\alpha_1^i$, $\alpha_2^i$…, $\alpha_{j-1}^i$ , $\alpha_j^i$ , $\alpha_{j+1}^{i+1}$ ,…, $\alpha_m^{i+1}$ )

**NB:** il faut noter que SOR n'utilise pas de transformation dans un espace de haute dimension.

## 4-3-3) Smooth SVM

Cet algorithme a été proposé dans **[29]**. Cet algorithme aussi utilise une formulation du SVM différente de la formulation standard. La distance entre les marges séparatrices est mesurée dans l'espace à (n + 1) dimensions. De plus c'est la somme des carrés des erreurs $\xi_i$ qui apparaît dans la fonction objective. La nouvelle formulation est :

$$\begin{cases} \text{Min } ½ (w^2 + b^2) + ½ C \left(\sum_{i=1}^{m} \xi_i^2\right) \quad ; C \text{ pondère la somme des erreurs} \\ \text{sous contraintes} \\ \qquad y_i (w x_i + b) \ge +1 - \xi_i \quad \forall i = 1 \ldots m \\ \qquad \xi_i \ge 0 \quad \forall i = 1 \ldots m \end{cases} \qquad (4.13)$$

La forme matricielle est :

$$\begin{cases} \text{Min } ½ (w^T w + b^2) + ½ C (\xi^T \xi \ ) \\ \text{sous contraintes} \\ \qquad Y (X w + 1\, b) + \xi \geq 1 \\ \qquad \xi \geq 0 \end{cases} \qquad (4.14)$$

A partir de là on a :

$$\xi_i = (1 - y_i (w\, x_i - b))_+$$
$$= \text{Max} ( (1 - y_i (w\, x_i - b)) , 0)$$

En remplaçant $\xi_i$ par sa valeur dans la fonction objective on aura la nouvelle formulation suivante :

$$\begin{cases} \text{Min } ½ (w^2 + b^2) + ½ C (\sum_{i=1}^{m} \|(1 - y_i (w\, x_i - b))_+\|^2 \ ) \\ \text{soumise à aucune contrainte} \end{cases} \qquad (4.15)$$

Cette nouvelle fonction objective est fortement convexe. Comme cette fonction n'est pas deux fois différentiable on ne peut pas directement utiliser la méthode de Newton. Pour cela l'auteur a remplacé la fonction $(x)_+$ par la fonction dérivable (lisse, smooth) suivante:

$$p(x,t) = x + (1/t) \log(1 + \varepsilon^{-t\,x}) ; \qquad t > 0 \qquad (4.16)$$

à nouveau la fonction objective :

$$\text{Min } L_D \ (w,b) \equiv ½ (w^2 + b^2) + ½ C (\sum_{i=1}^{m} \| p(1 - y_i (w\, x_i - b), t)\|^2 \ ) \qquad (4.17)$$

La forme matricielle de la fonction objective :

$$\text{Min } L_D \ (w,b) \equiv ½ (w\, w^T + b^2) + ½ C (\| p(1 - Y (X\, w - 1\, b), t)\|^2 \ ) \qquad (4.18)$$

L'algorithme proposé utilise la méthode de Newton Armijo. Le principe est une descente de gradient appliqué à la fonction objective.

## 4-3-4) SVM avec $L_1$ et $L_{+\infty}$

Cette variante a été proposée dans **[30]**. L'idée est d'utiliser d'autre mesure de distance à la place de $L_2$. Cette méthode crée un séparateur linéaire (pas de transformation non linéaire dans un espace à haute dimension). L'auteur a tous d'abord démontré la relation suivante :
On défini :

la q norme $\| w \|_q = \left( \sum_{i=1}^{n} |w_i|^q \right)^{(1/q)}$ (4.19)

la q distance $D_p(H_1,H_2) = \min \| x - y \|_p$ tel que $x \in H_1$ et $y \in H_2$ (4.20)

Si on a : $1/p + 1/q = 1$ p,q des entier positifs
$H_1$ un hyper plan qui a pour équation $w x + b_1 = 0$
$H_2$ un hyper plan qui a pour équation $w x + b_2 = 0$

Alors $D_p(H_1,H_2) = |b_1 - b_2| / \| w \|_q$ (4.21)

### 4-3-4-1) SVM avec $L_1$

On cherche à maximiser la distance entre les marges $H_1$ et $H_2$ en utilisant la distance $L_1$. Soient les hyperplans suivant :

$H_1 : w x + (b+1) = 0$
$H_2 : w x + (b-1) = 0$

$$D_1(H_1,H_2) = |(b+1) - (b-1)| / \| w \|_\infty = 2 / \| w \|_\infty = 2 / ( \max_j | w_j | ) \quad (4.22)$$

La formulation du problème devient

$$\begin{cases} \text{Min} \quad a + C \left( \sum_{i=1}^{n} \xi_i \right) \\ \text{sous contraintes :} \\ \quad y_i (w x_i + b) \geq +1 - \xi_i \quad \forall i = 1 \dots n \\ \quad \xi_i \geq 0 \quad \forall i = 1 \dots n \\ \quad | w_j | \leq a \quad \forall j = 1 \dots n \end{cases} \quad (4.23)$$

### 4-3-4-2) SVM avec $L_{+\infty}$

On cherche à maximiser la distance entre les marges $H_1$ et $H_2$ en utilisant la distance $L_{+00}$. Soient les hyperplans suivant :

$H_1 : w x + (b+1) = 0$
$H_2 : w x + (b-1) = 0$

$$D_\infty (H_1,H_2) = |(b+1) - (b-1)| / \| w \|_\infty = 2 / \| w \|_{00} = 2 / ( \textstyle\sum_j | w_j | ) \quad (4.24)$$

La formulation du problème devient :

$$\begin{cases} \text{Min} \quad \sum_{i=1}^{n} a_i + C \left( \sum_{i=1}^{n} \xi_i \right) \\ \text{sous contraintes :} \\ \qquad y_i (w x_i + b) \geq +1 - \xi_i \quad \forall i = 1 \ldots n \\ \qquad \xi_i \geq 0 \quad \forall i = 1 \ldots n \\ \qquad | w_j | \leq a_j \quad \forall j = 1 \ldots n \end{cases} \qquad (4.25)$$

## 4-3-5) Proximal SVM

Cette méthode a été proposée dans **[27]**. Au lieu de chercher un hyperplan séparateur qui maximise la marge, l'auteur propose de chercher pour chaque classe l'hyperplan le plus proche de l'ensemble des points de cette classe.

$$\begin{cases} \text{Min } ½ (w^2 + b^2) + ½ C \left( \sum_{i=1}^{m} \xi_i^2 \right) \quad ; C \text{ pondère la somme des erreurs} \\ \text{sous contraintes} \\ \qquad y_i (w x_i + b) + \xi_i = 1 \quad \forall i = 1 \ldots m \end{cases} \qquad (4.26)$$

La forme matricielle du problème est:

$$\begin{cases} \text{Min } ½ (w^T w + b^2) + ½ C (\xi^T \xi) \\ \text{sous contraintes} \\ \qquad Y (X w + 1 b) + \xi = 1 \end{cases} \qquad (4.27)$$

Après calcul l'auteur obtient la formulation suivante :

$$\alpha = ((1/C) + Y( X X^T + 1 1) Y )^T )^{-1} 1 = ((1/C) + H H^T )^{-1} 1; \quad \text{Ou} \quad H = Y(X - 1) \qquad (4.28)$$

Avec $w = X^T Y \alpha$ ;
$b = -1^T Y \alpha$
$\xi = \alpha / C$

Pour éviter l'inversion de matrice de grande taille m * m l'auteur utilise la formule de Sherman-Morrison-Woodbury et obtient la formule suivante

$$\alpha = C (1 - H( (1/C) + H H^T )^{-1} H^T ) 1 \qquad (4.29)$$

Cette nouvelle formulation demande seulement l'inversion de matrice d'ordre n * n.

Les méthodes basées sur d'autres formulations mathématiques du problème de SVM ont de bonnes performances en terme de temps de calcul. Mais soufrent de manque de généralisation, en effet le séparateur construit avec ces méthodes n'est pas le même que celui

construit avec les SVM classiques. Et donc, ces méthodes ne bénéficient pas des fondements théoriques auxquels ont droit les SVM classiques et qui font la renommé de ces derniers.

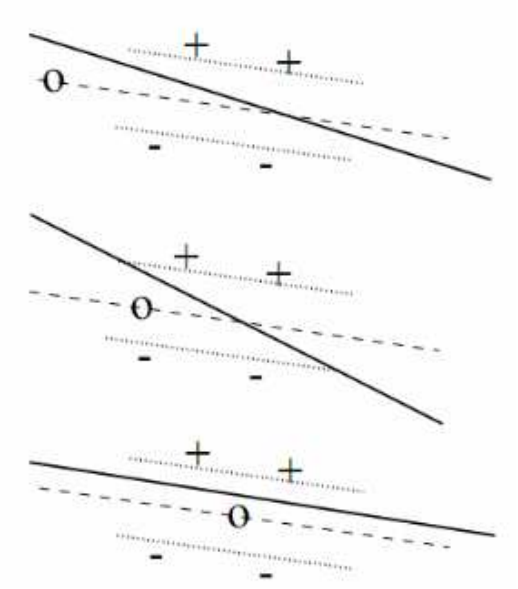
# 4-4) Algorithme Basé sur la réduction de l'ensemble d'apprentissage

## 4-4-1) Random Sampling the data

Cet algorithme a été proposé dans **[33]**. L'algorithme fonctionne suivant un cycle sélection de variable / recherche de séparateur. Un poids $p_i$ est associé à chaque exemple. Ces poids reflètent la pertinence de l'exemple. Initialement tous les pi ont une même valeur. La sélection des exemples se fait uniformément selon les poids pi. Après avoir sélectionné un sous ensemble d'exemples, on cherche un séparateur uniquement avec ce sous ensemble. Une fois le séparateur construit, on classifie l'intégralité des exemples avec ce dernier. Après cela on cherche les exemples mal classés et on double leurs poids, puis on réitère le cycle. Après plusieurs cycles, les vecteurs de support verrant leurs poids se distinguer des autres poids. La solution du dernier cycle va donner la solution globale.

## 4-4-2) Active Learning with SVM

Cet algorithme a été proposé dans **[34]**. L'idée est de sélectionner uniquement les exemples opportuns à la recherche du séparateur optimale. Les exemples sont sélectionnés en fonction de leurs positions par rapport au séparateur. L'algorithme passe par cycle sélection




L'hyperplan séparateur sans (en trait pointillé) et avec (en trait continue) l'influence du point ***o***.

d'exemples / recherche de séparateur. Initialement on choisie au hasard un sous ensemble d'exemples et on l'utilise pour chercher le séparateur. Une fois le séparateur construit, on cherche les exemples proches du séparateur et on constitue avec ces derniers un nouveau sous

ensemble d'apprentissage. Le processus est réitéré plusieurs fois jusqu'à ce que tous les exemples qui sont très proches du séparateur soient tous des vecteurs de support.

La figure ci dessus montre l'influence d'un point sur le hyperplan séparateur. On voit que l'on peut estimer l'influence du point sans connaître sa classe. Sur la figure le point est de classe négative (–) mais l'influence est la même si le point est de classe positive (+)

## 4-4-3) Algorithme CB-SVM

Cet algorithme a été proposé dans **[31]**. Il se déroule en deux phases, une phase de clustering des données et une autre phase pour la recherche du séparateur. La phase de clustering permet de réduire la taille de l'ensemble d'apprentissage. La recherche du séparateur est faite avec les centres des clusters $c_i$ au lieu des $x_i$.

$$\begin{cases} \text{Min } ½\, w^2 + C\,(\sum_{i=1}^{n} \xi_i\ ) & C \text{ est le paramètre qui pondére la somme des erreurs.} \\ \text{sous contraintes :} \\ \quad y_i\,(w c_i + b) \geq\ +1 - \xi_i & \forall\, i = 1 \ldots n \\ \quad \xi_i \geq 0 & \forall\, i = 1 \ldots n \quad \text{(n le nombre de cluster)} \end{cases} \qquad (4.30)$$

La recherche du séparateur se fait en suivant un cycle résolution / décomposition de clusters. A chaque fois qu'on trouve un séparateur, on cherche les centres $c_i$ qui sont très proches du séparateur et on les décompose en sous clusters. L'algorithme s'arrête une fois que l'ensemble des vecteurs de support ne contiendra plus de centre de cluster.

## 4-4-4) Algorithme CB-SOCP

Cet algorithme a été proposé dans **[32]**. Cet algorithme suppose que les données suivent une loi en mélange de gaussiennes et cherche dans un premier temps les gaussiennes (centre et variance) qui représentent les données avec l'algorithme de maximum de vraisemblance. Une fois les gaussiennes trouvés, l'algorithme les utilise pour résoudre le problème de classification au lieu des données d'origine. La résolution est faite avec la formulation Second Ordre Cone Programming.

Après le calcul des gaussiennes on trouve :

$\mu_i$ le centre de la gaussienne i et

$\sigma_i 1$ la matrice de variance covariance de la gaussienne i.

La formulation du problème de classification est :

$$\begin{cases} \text{Min } \sum_{i=1}^{g} \xi_i \\ \text{sous contraintes} \\ \quad y_i\,(w\mu_i + b)\ \geq 1 - \ \xi_i \quad +q\,\sigma_i W & \forall\, i = 1 \ldots g \\ \quad W\ \geq \|w\| \text{ et } \xi_i \geq 0 & \forall\, i = 1 \ldots g \end{cases} \qquad (4.31)$$

Où g est le nombre de gaussiennes
$q = \Phi^{-1}(\eta)$ et

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{s} \exp\left(-s^2/2\right)\, ds$$

## 4-5) Conclusion

Dans ce chapitre, nous avons vu plusieurs algorithmes dédiés à la résolution du problème des SVM. Comme nous l'avons dit au début du chapitre, le problème majeurs des SVM est le temps de calcule et l'espace mémoire qu'ils exigent. Nous avons vu ici des techniques permettant de contourner ces difficultés. Dans le prochain chapitre nous allons présenter une nouvelle approche de résolution du problème des SVM. Celle-ci peut être classée dans la catégorie de méthodes basées sur la réduction de l'ensemble d'apprentissage.

# Chapitre 5 : Contribution, réalisation et discutions

## 5-1) introduction

Le problème majeur avec les SVM est l'espace mémoire qu'ils exigent. Dans le cas du data mining, il est inenvisageable d'utiliser les SVM tel qu'ils ont été défini par Vapnik. D'autres artifices ont été utilisés pour les adapter (voir chapitre précédant). Dans ce chapitre nous allons présenter une nouvelle méthode d'utilisation des SVM plus adapté aux gros volumes de données. Nous avons intitulé cette méthode SMOWR pour Sequential Minimal Optimization with Weighted Ray.

## 5-2) SMOWR (SMO with Weighted Ray)

L'ensemble des données utilisées dans l'apprentissage des SVM est grand. Cela permet de minimiser le risque d'avoir un ensemble d'exemple non représentatif. L'inconvénient est que cela entraîne des temps de calcul très longs. L'idéal serait d'avoir un ensemble d'exemples qui soit à la fois, le plus petit possible et le plus représentatif possible. Notre approche se déroule sur deux phases. Dans la première phase, nous proposons d'utiliser une méthode pour agglomérer les données (sorte de clustering supervisé). Il faut noter que dans cette phase il n'y a aucune supposition sur la distribution statistique des exemples, ce qui constitue un avantage sérieux **[22]**. Dans la deuxième phase on utilise les SVM pour créer un séparateur entre les clusters. Dans un souci de clarté, nous allons d'abord expliquer la méthode pour le cas de SVM linéaire puis nous passerons au SVM non linéaire. Par la suite, nous allons donner la démonstration mathématique de la convergence de SMOWR. Finalement, on donnera les résultats de l'expérimentation sur des données réels.

### 5-2-1) SMOWR pour le cas linéaire

Comme on l'a déjà dit, l'objectif de la première phase est de diminuer le nombre d'exemples, pour cela nous allons utiliser la méthode de clustering avec les contraintes suivantes :

- La distance maximale entre les éléments appartenants à un même cluster ne doit pas dépasser un seuil $\mathbf{r_{max}}$.
- Un cluster ne doit contenir que des éléments appartenants à la même classe.
- Dans un même cluster le point le plus proche de tous les autres points de ce cluster devient le centre du cluster.

- Chaque cluster est muni d'un rayon $r_i$.
- Soit $C_i$ un cluster de centre $c_i$ et de rayon $r_i$ :
  Si pour chaque paire de cluster $C_i$ , $C_j$ de centre $c_i$ , $c_j$ et de rayon $r_i$ , $r_j$ . Si la classe de $C_i$ diffère de la classe de $C_j$ alors
  $r_i + r_j < D(c_i , c_j)$ Où $D(c_i , c_j)$ est la distance entre $c_i$ , $c_j$

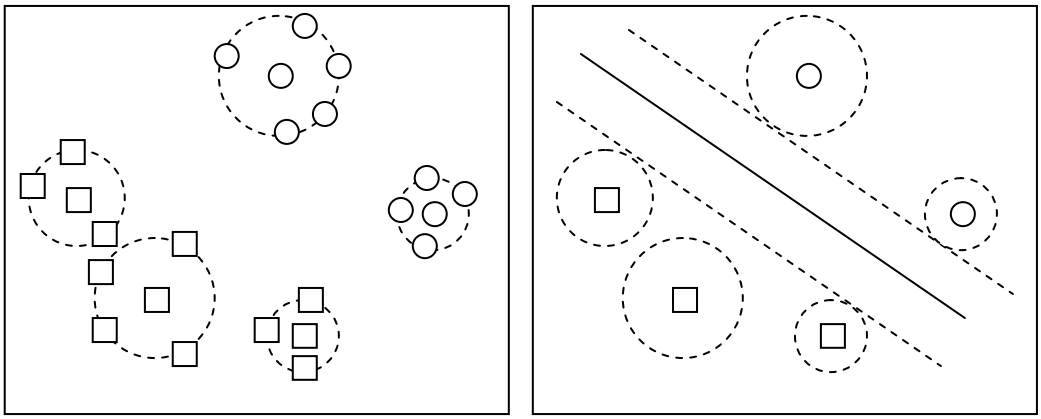
Le calcul du rayon d'un cluster se fait en utilisant la distance entre le centre du cluster et les autres éléments du cluster. Cela peut se faire de plusieurs façons, par exemple :
La distance maximale :

$$r_i = \max \{D(c_i , x_j) \mid x_j \in C_i \}$$

La moyenne des distances :

$$r_i = (1/n_i) \sum D(c_i , x_j) \; ; \; x_j \in C_i$$ Où $(n_i + 1)$ est le cardinale de $C_i$

Une fois les clusters formés, on ne garde que les centres des classes et on va les utiliser comme données d'apprentissage pour la deuxième phase qui est la recherche du séparateur. Il ne faut pas perdre de vue que les données qu'on va utiliser pour cette phase sont des cluster et donc les représenter sous forme d'hyper sphères est plus approprié.



**(a)** Création des clusters **(b)** Création du séparateur

**fig. 5.1 :** Les étapes de l'algorithme

Pour que l'hyperplan soit un bon séparateur, tous les points (ici on fait allusion seulement aux centres des clusters) doivent vérifier la condition suivante :

$wx_i + b \geq (r_i + 1)$ pour $y_i = +1$
$wx_i + b \leq -(r_i + 1)$ pour $y_i = -1$

Comme auparavant, on va assouplir un peut les contraintes sur l'hyperplan et on va tolérer quelques erreurs. Pour cela on va introduire un coefficient d'erreur comme suit :

$wx_i + b \geq (r_i + 1) - \xi_i$ pour $y_i = +1$
$wx_i + b \leq -(r_i + 1) + \xi_i$ pour $y_i = -1$

ce qui donne :

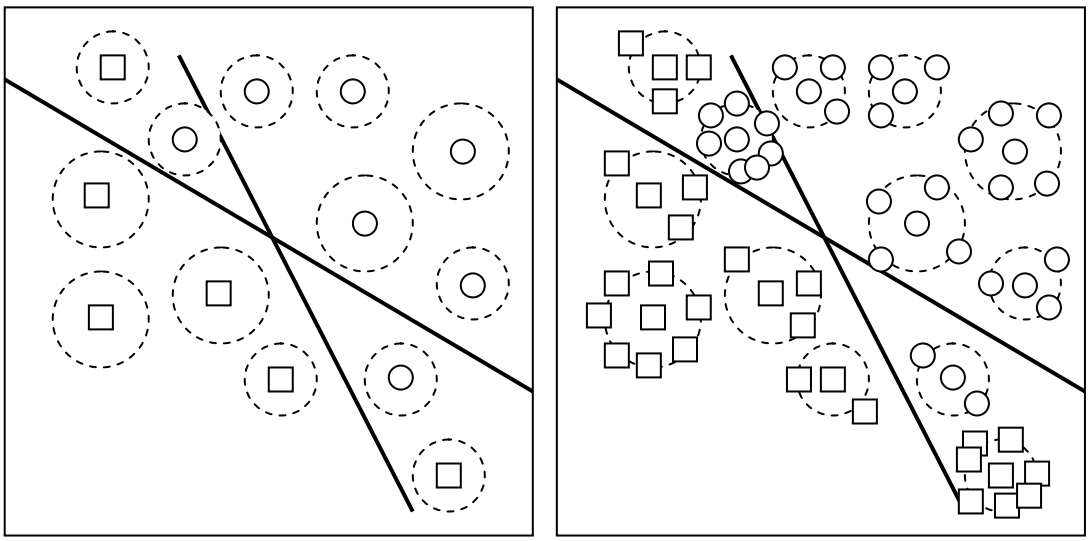
$y_i (wx_i + b) \geq (r_i + 1) - \xi_i$

En suivant le même raisonnement que celui du paragraphe précédent, on obtient la formulation suivante du problème :

$$\begin{cases} \text{Min } ½\, w^2 + C \left(\sum_{i=1}^{n} \xi_i\right) & C \text{ est le paramètre qui pondéré la somme des erreurs.} \\ \text{sous contrainte :} \\ \quad y_i\,(w x_i + b) \geq (r_i + 1) - \xi_i & \forall\, i = 1 \ldots n \\ \quad \xi_i \geq 0 & \forall\, i = 1 \ldots n \quad (n \text{ le nombre de cluster}) \end{cases} \qquad 5.1$$

Le nombre d'exemples et leur densité dans les clusters varient d'un cluster à un autre. Lors de la création du séparateur, il faut aussi tenir compte de cela. Mal classer un cluster qui a beaucoup d'exemples au profit d'un cluster qui a peu d'exemples reviendrait à mal classer un nombre important d'exemples. Pour éviter cela, on va pondérer les centres des clusters avec des poids qui vont refléter leurs importances comme suit :

$P_i = (1/r_i) \cdot n_i \cdot (1/d)$ ou $r_i$ est le rayon du cluster
$n_i$ le nombre d'exemples dans un cluster
d la dimension de l'espace donc ici d = 2



**(a)** **(b)**

**fig. 5.2 :** Intérêt de la pondération

On voit sur la figure 5.2 (a) si dessus deux séparateurs qui on à priori le même taux d'erreurs (deux clusters mal classés pour chacun d'eux). En regardant plus en détail les clusters (figure 2(b)) on voit qu'en réalité le séparateur $\Delta_0$ classe mal sept (7) exemples en tous, tandis que le séparateur $\Delta_1$ lui, classe mal seize (16) exemples en tous. Sachant cela il est évidant que le séparateur $\Delta_0$ est meilleur. La pondération des clusters donc va permettre d'apporter un plus d'information concernant les données.

La reformulation du problème d'optimisation devient alors :

$$\begin{cases} \text{Min } ½\ w^2 + C\ (\sum_{i=1}^{n} P_i \cdot \xi_i\ ) & C \text{ est le paramètre qui pondère la somme des erreurs.} \\ \text{sous contrainte :} & \\ \quad y_i\ (wx_i + b) \geq\ (r_i + 1)\ - \xi_i & \forall\ i = 1 \ldots n \\ \quad \xi_i \geq 0 & \forall\ i = 1 \ldots n \quad \text{(n le nombre de cluster)} \end{cases} \qquad (5.2)$$

puis le Lagrangien :

$$\begin{cases} \text{Min } L_P \equiv\ ½\ w^2 + C\ (\sum_{i=1}^{n} P_i \cdot \xi_i\ ) - \sum_{i=1}^{n} \alpha_i\ [y_i\ (wx_i + b) - (r_i + 1) + \xi_i] - \sum_{i=1}^{n} \mu_i\ \xi_i \\ \text{sous contraintes} \\ \quad \alpha_i \geq 0 \qquad \forall\ i = 1 \ldots n \\ \quad \mu_i \geq 0 \qquad \forall\ i = 1 \ldots n \end{cases}$$

Les conditions de KKT sont :

$$\partial L_P / \partial\ w = w - \sum_{i=1}^{n} \alpha_i\ y_i\ x_i = 0 \quad \Leftrightarrow \quad w = \sum_{i=1}^{n} \alpha_i\ y_i\ x_i$$

$$\partial L_P / \partial\ b = - \sum_{i=1}^{n} \alpha_i\ y_i = 0$$

$$\partial L_P / \partial\ \xi_i\ = P_i\ C - \alpha_i - \mu_i = 0$$

Les conditions de complémentarité

$$\begin{array}{ll} y_i\ (wx_i + b) - (r_i + 1) + \xi_i \geq 0 & \forall\ i = 1 \ldots n \\ \xi_i \geq 0 & \forall\ i = 1 \ldots n \\ \alpha_i \geq 0 & \forall\ i = 1 \ldots n \\ \mu_i \geq 0 & \forall\ i = 1 \ldots n \\ \alpha_i\ [y_i\ (wx_i + b) - (r_i + 1) + \xi_i] = 0 & \forall\ i = 1 \ldots n \\ \mu_i\ \xi_i = 0 & \forall\ i = 1 \ldots n \end{array}$$

A partir des équations $P_i\ C - \alpha_i - \mu_i = 0$ et $\mu_i\ \xi_i = 0$ on conclue que si $\alpha_i < P_i\ C$ alors $\xi_i = 0$.
Avec un peu d'algèbre élémentaire, on obtient la formulation duale suivante:

$$\begin{cases} \text{Max } L_D(\alpha) \equiv \sum_{i=1}^{n}(r_i + 1)\alpha_i - \tfrac{1}{2}\sum_{i=1}^{n}\alpha_i\,\alpha_j\,y_i\,y_j\,x_i\,x_j \\ \text{sous contraintes} \\ \quad 0 \le \alpha_i \le P_i\,C \\ \quad \sum_{i=1}^{n}\alpha_i\,y_i = 0 \\ \quad y_i\,(w x_i + b) - (r_i + 1) + \xi_i \ge 0 \qquad \forall\, i = 1 \ldots n \\ \quad \xi_i \ge 0 \qquad \forall\, i = 1 \ldots n \\ \quad \alpha_i \ge 0 \qquad \forall\, i = 1 \ldots n \\ \quad \mu_i \ge 0 \qquad \forall\, i = 1 \ldots n \\ \quad \alpha_i\,[y_i\,(w x_i + b) - (r_i + 1) + \xi_i] = 0 \qquad \forall\, i = 1 \ldots n \\ \quad \mu_i\,\xi_i = 0 \qquad \forall\, i = 1 \ldots n \end{cases} \qquad (5.3)$$

## 5-2-2) SMOWR pour le cas non linéaire

Dans ce cas, le principe est le même, la seule différence est dans la distance utilisée dans le calcul des cluster. Soit $\Phi(\ )$ la transformation non linéaire utilisée pour passer au nouvel espace et $K(\ ,\ )$ la fonction noyau correspondante. Donc on a :

$$\Phi : \quad R^B \longrightarrow R^H$$
$$x \longrightarrow \Phi(x)$$

$$K(a,b) = \Phi(a) \cdot \Phi(b)$$

Les clusters doivent être formés dans le nouvel espace $R^H$. Le calcul des distances se fera comme suit :

$$\begin{aligned} D(\Phi(a), \Phi(b)) &= \left( (\Phi(a) - \Phi(b))^2 \right)^{1/2} \\ &= \left( \Phi(a)^2 + \Phi(b)^2 - 2\,\Phi(a)\,\Phi(b) \right)^{1/2} \\ &= \left( K(a,a) + K(b,b) - 2\,K(a,b) \right)^{1/2} \end{aligned} \qquad (5.4)$$

Aussi comme dans le paragraphe précédent, la formulation de $L_D(\alpha)$ dans le cas non linéaire devient:

$$L_D(\alpha) = \sum_{i=1}^{n}(r_i + 1)\alpha_i - \tfrac{1}{2}\sum_{i=1}^{n}\alpha_i\,\alpha_j\,y_i\,y_j\,K(x_i, x_j) \qquad (5.5)$$

## 5-3) Convergence de SMOWR

Notre méthode SMOWR est inspirée de SMO. L'avantage de la méthode SMO est qu'elle optimise le modèle SVM originel et non une variante de celui-ci. Nous allons à présent donner les fondements mathématiques de SMOWR.

L'objectif est de maximiser

$$\begin{cases} L_D(\alpha) = \sum_{i=1}^{n}(r_i + 1)\alpha_i - \tfrac{1}{2}\sum_{i=1\_j=1}^{n}\alpha_i\,\alpha_j\,y_i\,y_j\,x_i\,x_j \\ \text{sous contraintes} \\ \quad 0 \le \alpha_i \le P_i\,C \\ \quad \sum_{i=1}^{n}\alpha_i\,y_i = 0 \end{cases}$$

SMOWR optimise deux $\alpha_i$ à la fois en fixant les autres $\alpha_i$ . Soit $\alpha_1$ $\alpha_2$ les cœfficients à optimiser et soit $\alpha_1^{old}$ $\alpha_2^{old}$ leurs anciennes valeurs et $\alpha_1^{new}$ $\alpha_2^{new}$ leurs nouvelles valeurs.

A cause de la contrainte $\sum_{i=1}^{n}\alpha_i\,y_i = 0$ nous obtenant une autre contrainte

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

Soit $s = y_1\,y_2$ nous avons $y_1\,\alpha_1 + y_2\,\alpha_2 = Const$ en multipliant par $y_1$ on obtient

$\alpha_1 + s\,\alpha_2 = y_1 \cdot Const = t$ d'où $\alpha_1 = t - s\,\alpha_2$

et $t = \alpha_1^{old} + s\,\alpha_2^{old}$

En fixant les $\alpha_i$ pour i = 3, 4, ..., n on peut réécrire $L_D(\alpha)$ comme suit :

$$L_D(\alpha) = (r_1 + 1)\alpha_1 + (r_2 + 1)\alpha_2 + \sum_{i=3}^{n}(r_i + 1)\alpha_i - \tfrac{1}{2}\Big(\sum_{i=3\_j=3}^{n}\alpha_i\,\alpha_j\,y_i\,y_j\,x_i x_j$$

$$+ \alpha_1\,\alpha_1\,y_1\,y_1\,x_1x_1 + \alpha_1\,\alpha_2\,y_1\,y_2\,x_1x_2 + 2\sum_{i=3}^{n}\alpha_1\,\alpha_j\,y_1\,y_j\,x_1x_j$$

$$+ \alpha_2\,\alpha_1 y_2\,y_1\,x_2\,x_1 + \alpha_2\,\alpha_2\,y_2\,y_2\,x_2x_2 + 2\sum_{i=3}^{n}\alpha_2\,\alpha_j\,y_2\,y_j\,x_2x_j\Big)$$

$$= (r_1 + 1)\alpha_1 + (r_2 + 1)\alpha_2 - \tfrac{1}{2}\Big(\alpha_1\,\alpha_1\,y_1\,y_1\,x_1x_1 + \alpha_1\,\alpha_2\,y_1\,y_2\,x_1x_2 + 2\sum_{i=3}^{n}\alpha_1\,\alpha_j\,y_1\,y_j\,x_1x_j$$

$$+ \alpha_2\,\alpha_1 y_2\,y_1\,x_2\,x_1 + \alpha_2\,\alpha_2\,y_2\,y_2\,x_2x_2 + 2\sum_{i=3}^{n}\alpha_2\,\alpha_j\,y_2\,y_j\,x_2x_j\Big) + Const$$

Soit $K_{11} = x_1x_1$ , $K_{22} = x_2x_2$ , $K_{12} = x_1x_2$

$$v_j = \sum_{i=3}^{n}\alpha_i y_i x_i x_j$$

$$= x_j\,w^{old} - \alpha_1^{old}\,y_1x_1x_j - \alpha_2^{old}\,y_2x_2x_j$$

$= (x_j w^{old} - b^{old}) + b^{old} - \alpha_1^{old} y_1 x_1 x_j - \alpha_2^{old} y_2 x_2 x_j$
$= u_j^{old} + b^{old} - \alpha_1^{old} y_1 x_1 x_j - \alpha_2^{old} y_2 x_2 x_j$ tel que $u_j^{old} = (x_j w^{old} - b^{old})$

$$L_D(\alpha) = (r_1 + 1)\alpha_1 + (r_2 + 1)\alpha_2 - \tfrac{1}{2}\left( K_{11}\alpha_1^2 + K_{22}\alpha_2^2 + 2 s K_{12}\alpha_1\alpha_2 + 2y_1v_1\alpha_1 + 2y_2v_2\alpha_2\right) + \text{Const}$$

$$L_D(\alpha) = (r_1 + 1)(t - s\alpha_2) + (r_2 + 1)\alpha_2 - \tfrac{1}{2}\left( K_{11}(t - s\alpha_2)^2 + K_{22}\alpha_2^2 + 2 s K_{12}(t - s\alpha_2)\alpha_2 + 2y_1v_1(t - s\alpha_2) + 2y_2v_2\alpha_2\right) + \text{Const}$$

$$= ((r_2 + 1) - s(r_1 + 1))\alpha_2 - \tfrac{1}{2} K_{11}(t - s\alpha_2)^2 - \tfrac{1}{2} K_{22}\alpha_2^2 - s K_{12}(t - s\alpha_2)\alpha_2 - y_1v_1(t - s\alpha_2) - y_2v_2\alpha_2 + \text{Const}$$

$$= ((r_2 + 1) - s(r_1 + 1))\alpha_2 - \tfrac{1}{2} K_{11}t^2 + sK_{11}t\alpha_2 - \tfrac{1}{2} K_{11}s^2\alpha_2^2 - \tfrac{1}{2} K_{22}\alpha_2^2 - s K_{12}t\alpha_2 + s^2 K_{12}\alpha_2^2 - y_1v_1t + s y_1v_1\alpha_2 - y_2v_2\alpha_2 + \text{Const}$$

$$= ((r_2 + 1) - s(r_1 + 1))\alpha_2 + sK_{11}t\alpha_2 - \tfrac{1}{2} K_{11}\alpha_2^2 - \tfrac{1}{2} K_{22}\alpha_2^2 - s K_{12}t\alpha_2 + K_{12}\alpha_2^2 + y_2v_1\alpha_2 - y_2v_2\alpha_2 + \text{Const}$$

$$= (-\tfrac{1}{2} K_{11} - \tfrac{1}{2} K_{22} + K_{12})\alpha_2^2 + ((r_2 + 1) - s(r_1 + 1) + sK_{11}t - s K_{12}t + y_2v_1 - y_2v_2)\alpha_2 + \text{Const}$$

$$= \tfrac{1}{2}(2 K_{12} - K_{11} - K_{22})\alpha_2^2 + ((r_2 + 1) - s(r_1 + 1) + sK_{11}t - s K_{12}t + y_2v_1 - y_2v_2)\alpha_2 + \text{Const}$$

Soit $\eta = (2 K_{12} - K_{11} - K_{22})$, soit $z = ((r_2 + 1) - s(r_1 + 1) + sK_{11}t - s K_{12}t + y_2v_1 - y_2v_2)$

$$z = (r_2 + 1) - s(r_1 + 1) + sK_{11}t - s K_{12}t + y_2v_1 - y_2v_2$$

$$= (r_2 + 1) - s(r_1 + 1) + sK_{11}(\alpha_1^{old} + s\alpha_2^{old}) - s K_{12}(\alpha_1^{old} + s\alpha_2^{old}) + y_2v_1 - y_2v_2$$

$$= (r_2 + 1) - s(r_1 + 1) + sK_{11}(\alpha_1^{old} + s\alpha_2^{old}) - s K_{12}(\alpha_1^{old} + s\alpha_2^{old}) + y_2(u_1^{old} + b^{old} - \alpha_1^{old} y_1K_{11} - \alpha_2^{old} y_2K_{12}) - y_2(u_2^{old} + b^{old} - \alpha_1^{old} y_1K_{12} - \alpha_2^{old} y_2K_{22})$$

$$= (r_2 + 1) - s(r_1 + 1) + sK_{11}\alpha_1^{old} + K_{11}\alpha_2^{old} - s K_{12}\alpha_1^{old} + K_{12}\alpha_2^{old} + y_2 u_1^{old} + y_2 b^{old} - s\alpha_1^{old}K_{11} + \alpha_2^{old}K_{12} - y_2 u_2^{old} - y_2 b^{old} + s\alpha_1^{old}K_{12} + \alpha_2^{old}K_{22}$$

$$= (r_2 + 1) - s(r_1 + 1) + (sK_{11} - s K_{12} - s K_{11} + s K_{12})\alpha_1^{old} + (K_{11} + 2 K_{12} + K_{22})\alpha_2^{old} + y_2(u_1^{old} - u_2^{old})$$

$$= (r_2 + 1) - s(r_1 + 1) + (K_{11} + 2 K_{12} + K_{22})\alpha_2^{old} + y_2(u_1^{old} - u_2^{old})$$

$$= y_2^2(r_2 + 1) - y_1y_2(r_1 + 1) + (K_{11} + 2 K_{12} + K_{22})\alpha_2^{old} + y_2(u_1^{old} - u_2^{old})$$

$$= y_2(y_2(r_2 + 1) - y_1(r_1 + 1) + u_1^{old} - u_2^{old}) - \eta\alpha_2^{old}$$

$$= y_2((u_1^{old} - y_1(r_1 + 1)) - (u_2^{old} - y_2(r_2 + 1))) - \eta\alpha_2^{old}$$

$= y_2 (E_1^{old} - E_2^{old}) - \eta\alpha_2^{old}$

Ou $E_i^{old} = u_i^{old} - y_i (r_i + 1)$ est l'erreur commise précédemment sur le point i

Donc la formulation de la fonction objective devient

$$L_D(\alpha) = \tfrac{1}{2} \eta \alpha_2^2 + (y_2 (E_1^{old} - E_2^{old}) - \eta\alpha_2^{old}) \alpha_2 + \text{Const} \qquad 5.6$$

On a les dérivées premières et seconde comme suit

$$\partial(L_D(\alpha)) / \partial \alpha_2 = \eta \alpha_2 + (y_2 (E_1^{old} - E_2^{old}) - \eta\alpha_2^{old})$$

$$\partial^2(L_D(\alpha)) / \partial \alpha_2^2 = \eta$$

Soit $\partial(L_D(\alpha)) / \partial \alpha_2 = 0$ alors

$$\alpha_2 = - (y_2 (E_1^{old} - E_2^{old}) - \eta\alpha_2^{old}) / \eta$$
$$= \alpha_2^{old} + (y_2 (E_1^{old} - E_2^{old})) / \eta$$

Nous avons $\eta = 2 K_{12} - K_{11} - K_{22} = - (x_1^2 - 2 x_1x_2 + x_2^2) = - (x_1 - x_2)^2 \leq 0$

Nous savons qu'une fonction atteint son maximum lorsque sa dérivée première s'annule et sa dérivée seconde est négative. Donc si $\eta < 0$ alors la formule précédente nous donne un $\alpha_2^{tmp}$ soumis à aucune contrainte qui correspond au maximum.

Nous avons les contraintes suivantes

$$0 \leq \alpha_1 \leq P_1C$$
$$0 \leq \alpha_2 \leq P_2C$$

D'où nous tirons les bornes suivantes :

si $y_1 = y_2$ alors

$L\alpha_2 = \max (0, t - P_1C)$

$H\alpha_2 = \min (t, P_2C)$

si $y_1 = - y_2$ alors

$L\alpha_2 = \max (0, -t)$

$H\alpha_2 = \min (P_2C, P_1C - t)$ avec toujours $t = \alpha_1 + y_1y_2 \alpha_2$

$$\text{donc } \alpha_2^{new} = \begin{cases} L\alpha_2 & \text{si } \alpha_2^{tmp} < L\alpha_2 \\ \alpha_2^{tmp} & \text{si } L\alpha_2 \leq \alpha_2^{tmp} \leq H\alpha_2 \\ H\alpha_2 & \text{si } H\alpha_2 < \alpha_2^{tmp} \end{cases} \qquad 5.7$$

Nous avons

$t = \alpha_1 + y_1y_2 \alpha_2$ et

$t = \alpha_1^{new} + y_1y_2 \alpha_2^{new}$

$\alpha_1 + y_1y_2 \alpha_2 = \alpha_1^{new} + y_1y_2 \alpha_2^{new}$

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

En résumé on optimise $\alpha_1$ et $\alpha_2$ comme suit

1-On calcule $\eta = 2 K_{12} - K_{11} - K_{22}$

2- si $\eta < 0$ alors

$\alpha_2^{tmp} = \alpha_2^{old} + (y_2 (E_1^{old} - E_2^{old})) / \eta$ et veiller à ce que $L\alpha_2 \leq \alpha_2^{new} \leq H\alpha_2$

$\alpha_1^{new} = \alpha_1 - y_1 y_2 (\alpha_2^{new} - \alpha_2)$

3- si $\eta = 0$ alors

Calculer la fonction objective sur les deux bornes $L\alpha_2$ et $H\alpha_2$ puis prendre $\alpha_2^{new}$ égale à la borne qui maximise la fonction objective. On calcule uniquement la partie variable de cette dernière, à savoir

$$L'_D(\alpha) = \frac{1}{2} \eta \alpha_2^2 + (y_2 (E_1^{old} - E_2^{old}) - \eta \alpha_2^{old}) \alpha_2$$

Le reste de l'algorithme est constitué d'heuristiques qui permettent de bien choisir les cœfficients $\alpha_1$ et $\alpha_2$ à optimiser et de mécanismes de mise en cache pour accélérer les calculs.

## 5- 4) Expérimentation

Pour évaluer les performances de notre approche nous avons implémenté deux programmes, le premier pour effectuer la phase de clustering et le deuxième pour la résolution avec la méthode de SVM.

Comme notre méthode réduit clairement le temps d'exécution grâce à la réduction du nombre d'exemples, il ne reste plus qu'à évaluer sa capacité de généralisation.

Pour démontrer l'efficacité de notre méthode, nous avons fait des testes sur l'ensemble de données 'UCI adult data set'. Cette ensemble est constitué de deux parties, une partie pour l'apprentissage de 30162 exemples et une autre partie pour le teste de 15060 exemples. Chaque exemple est décrit par 14 attributs en plus de l'attribut déterminant la classe. Dans cet ensemble on trouve deux classes. Une classe contient les personnes ayant un revenu supérieur à 50000$ et l'autre classe contient les personnes ayant un revenu inférieur ou égale à 50000$.

Dans notre teste nous allons construire un classificateur qui va distinguer entre ces deux catégories de personnes (revenue >50000$ ou ≤ 50000$).

Dans notre expérimentation, nous avons dans un premier temps comparé les résultats de notre méthode SMOWR face à SMO en utilisant un sous ensemble de 'UCI adult data set'. Ce sous ensemble est obtenu en prenant 10% d'exemples de l'ensemble originel d'apprentissage pour construire un sous ensemble d'apprentissage et 10% d'exemples de l'ensemble originel de teste pour construire un sous ensemble de teste. Nous allons voir que notre méthode présente un pouvoir de généralisation équivalent. Dans un second temps nous allons comparer les résultats de notre méthode face à la méthode du simple sampling sur l'ensemble 'UCI adult data set' complet.

Pour comparer les résultats des expérimentations nous avons utilisé la mesure d'erreur suivante :

$$ErreurW = \frac{1}{2} (FP / N) + \frac{1}{2} (FN / P) \qquad (5.8)$$

Au lieu de la classique

$$Erreur = (FP+FN) /(P + N) \quad (5.9)$$

Où N (Négatifs) : le nombre total d'exemples négatifs ;
P (Positifs) : le nombre total d'exemples positifs ;
FP (Faux Positifs): le nombre d'exemple négatifs classé comme étant positif par le classificateur ;
FN (Faux Négatifs): le nombre d'exemple positifs classé comme étant positif par le classificateur ;

Notre choix pour cette mesure, est motivé par le faite que cette dernière soit plus expressive, notamment lorsque la répartition du nombre d'exemples négatifs et du nombre d'exemples positifs est déséquilibrée, comme cela est le cas pour l'ensemble 'UCI adult data set'.

Dans ce premier, tableau on montre les résultats obtenus avec le sous ensemble construit à partir de 'UCI adult data set'. Nous avons pris les meilleures valeurs obtenues pour chaque méthode.

| | SMOWR, rayons activés | SMOWR, rayons désactivés | SMO |
|---|---|---|---|
| Kernel linéaire | 0.170971 | 0.240990 | 0.163265 |
| Kernel RBF | 0.150841 | 0.195553 | 0.146324 |
| Kernel Polynomial | 0.368443 | 0.398643 | 0.365481 |

Nous voyons sur ce tableau que les methodes SMOWR et SMO donne des resultats proches. SMO a de meulleur resultats, cependant son temps d'execution est plus grand.

Dans ce deuxième tableau, on montre les résultats obtenus avec l'ensemble 'UCI adult data set' entier.

| | SMOWR, rayons activés | SMOWR, rayons désactivés | SMO avec simple samplig |
|---|---|---|---|
| Kernel linéaire | 0.180926 | 0.260267 | 0.313274 |
| Kernel RBF | 0.160294 | 0.237185 | 0.275633 |
| Kernel Polynomial | 0.468445 | 0.496543 | 0.395281 |

Nous voyons sur deuxieme tableau SMOWR donne de meulleurs resultats que SMO avec simple sampling.

## 5- 5) Conclusion :

Dans ce chapitre, nous avons presenté notre aproche. Nous avons vu son fonctionnement dans le cas linéaire puis nous l'avons transposé au cas non linéaire. Nous avons aussi donné la démonstration de la convergence de l'algorithme SMOWR. A la fin nous avons fait l'expérimentation sur l'ensemble 'UCI adult data set'. Nous avons comparé les résultats obtenus avec SMOWR face à SMO et à SMO avec échantillonnage simple.

# Conclusion et perspectives

Dans le cadre de ce mémoire, nous avons travaillé dans un domaine de recherche d'actualité très intéressant. Nous avons pu explorer le domaine du data mining et pu voir la puissance de cet outil et les opportunités qu'il offre. Nous avons aussi pu voir les difficultés et problèmes rencontrés dans ce domaine.

Nous avons exploré les rouages de l'apprentissage artificiel et vue plusieurs méthodes de ce domaine. Nous nous sommes particulièrement intéressé aux méthodes des SVM, Support Vector Machine. Nous avons étudié les fondements théoriques de ces dernières. Nous avons aussi exploré plusieurs méthodes liées à elles. En fin nous avons apporté une contribution à ce domaine en proposant une nouvelle méthode des SVM adaptée au data mining que nous avons appelé SMOWR.

Pour l'avenir, nous envisageons de continuer dans cet axe en optimisant encore l'approche que nous avons proposé et en effectuant des séries de tests sur d'autres ensembles de données de référence (benchmarks). Nous envisageons aussi d'apporter de nouvelles méthodes ou applications dans le cadre des SVM.

# Bibliographie:

**[1]**: Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., The KDD process for extracting useful knowledge from volumes of data. Communications of the ACM 39, November 1996.,pp27-34.

**[2]**: Rish, Irina. (2001). "An empirical study of the naive Bayes classifier". IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, 2001.

**[3]**: Cover, T.M. and Hart, P.E., Nearest Neighbor Pattern Classification, IEEE Transactions on Information Theory, Volume IT-13(1), pp.21-27, 1967.

**[4]**: Dudani, S. A. The distance-weighted k-nearest-neighbor rule. IEEE Trans. Syst. Man Cyber., 6:325–327, 1976.

**[5]**: G.Guo, H.Wang, D.Bell, Y.Bi and K.Greer, kNN Model-Based Approach in Classification. In Proc. of CoopIS/DOA/ODBASE 2003 : 986-996, 2003.

**[6]**: Eick, C.F. Zeidat, N. Zhao, Z., Supervised clustering - algorithms and benefits, Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on 15-17 Nov. 2004.

**[7]**: Rakesh Agrawal, Tomasz Imielinski, Arun Swami. Mining association rules between sets of items in large databases. ACM SIGMOD International Conference on Management of Data, mai 1993, pp 207-216.

**[8]**: Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, Shalom Tsur .Dynamic itemset counting and implication rules for market basket data. ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, USA, 1997, pp. 255-264.

**[9]**: G. Gardarin, P. Pucheral, and F. Wu, Bitmap Based Algorithms For Mining Association Rules, In Actes des journées Bases de Données Avancées (BDA'98), Hammamet, Tunisie, October 1998.

**[10]**: Jiawei Han, Yongjian Fu. Mining Multiple-Level Association Rules in Large Databases. IEEE Transactions on Knowledge and Data Engineering archive Volume 11 , Issue 5 (September 1999). pp 798 – 805.

**[11]**: C.H. Cai, Ada W. C. Fu, C. H. Cheng, W. W. Kwong. Mining Association Rules with Weighted Items. International Database Engineering and Applications Symposium (IDEAS 98), Aug 1998.

**[12]**: R. Srikant, R. Agrawal. Mining quantitative association rules in large relationnal tables. ACM SIGMOD conference on management of data, Montreal, Quebec, Canada, June 1996.

**[13]**: A. J. Smola and B. Schölkopf. A tutorial on support vector regression, Stat. Comput. 2004, 14, pp. 199-222.

**[14]**: C. Cortes and V. Vapnik, Support vector networks. M. Learning, 20, pp. 273-297, 1995.

**[15]**: V. N. Vapnik, The Nature of Statistical Learning Theory. Springer-Verlag, 1995.

**[16]**: V. D. Sanchez, Advanced support vector machines and kernel methods, Neurocomputing 2003, 55, 5-20.

**[17]**: V. N. Vapnik, Estimation of dependences based on empirical data, Addendum 1, New York, Springer Verlag 1982.

**[18]**: C. Burges, A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 2 (1998), pp. 121–167.

**[19]**: N. Cristianini, C. Campbell, C. Burges (Eds.). Support vector machines and kernel methods, Machine Learning, 2001.

**[20]**: C. Campbell, Kernel methods: a survey of current techniques, Neurocomputing 2002, 48, pp. 63-84.

**[21]**: Wai-TakWong, Sheng-Hsun Hsu. Application of SVM and ANN for image retrieval. European Journal of Operational Research, 173(2006), pp. 938-950.

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

**[23]**: E. Frias-Martinez, A. Sanchez , J. Velez. Support vector machines versus multi-layer perceptrons for efficient off-line signature recognition. Engineering Applications of Articial Intelligence 19 (2006), pp. 693-704.

**[24]**: E. Osuna, R. Freund, and F. Girosi. An Improved training algorithm for support vector machines. Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop. 24-26 Sept. 1997, pp. 276 – 285

**[25]**: J. C. Platt. Sequential minimal optimization : A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, 1998.

**[26]**: Thilo-Thomas Frieβ and Nello Cristianini and Colin Campbell, The Kernel-Adatron algorithm: a fast and simple learning procedure for Support Vector machines, Proc. 15th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA, 1998, pp. 188-196.

**[27]:** Glenn Fung and Olvi L. Mangasarian. Proximal support vector machine classifiers. In KDD '01: Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 77--86, New York, NY, USA, 2001. ACM Press. Gene H. Golub and Charles F. Van Loan. Matrix computations. Johns Hopkins University Press, Baltimore, MD, USA, 3rd edition, 1996.

**[28]:** O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. IEEE Transactions on Neural Networks, 1999.

**[29]**: Yuh-Jye Lee and O. L. Mangasarian. SSVM: A smooth support vector machine. Computational Optimization and Applications, 2000.

**[30]**: Joao Pedro Pedroso and Noboru Murata. Riken Brain Science Institute support vector machine for linear programming motivation and formulations

**[31]**: Hwanjo Yu, Jiong Yang, Jiawei Han. CB-SVM: Classifying large data sets using SVMs with hierarchical clusters. August 2003, Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining KDD '03.

**[32]**: J. Saketha Nath, Chiranjib Bhattacharyya, M. Narasimha Murty. Clustering based large margin classification: a scalable approach using SOCP formulation. Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006.

**[33]**: Jose L. Balcazar, Yang Dai, and Osamu. Watanabe. A random sampling technique for training support vector machines. In Proc. 13th Int. Conf. Algorithmic Learning Theory, Washington D.C., 2001.

**[34]**: Greg Schohn and David Cohn. Less is more: Active learning with support vector machines. In Proc. 17th Int. Conf. Machine Learning, Stanford, CA, 2000.

**[35]**: D. Hush, C. Scovel, Polynomial-time decomposition algorithms for support vector machines, Mach. Learn. 51 (2003) 51–71.

**[36]**: N. List, H. -U. Simon, General polynomial time decomposition algorithms, in: COLT, 2005, pp.308-322.

**[37]**: L. J. Cao , S. S. Keerthi, C. J. Ong ,P. Uvaraj ,X. J. Fu, H. P. Lee, Developing parallel sequential minimal optimization for fast training support vector machine. Neurocomputing 70 (2006), pp. 93-104.

**[38]** : G. Dreyfus, J.-M. Martinez, M. Samuelides, M. B. Gordon, F. Badran, S. Thiria et L. Hérault. Les reseau de neuron, methodologie et application. Eyrolles 2004.

**[39]** : Jerôme Callut. Implementation efficace des support vector machines pour la classification. 2003. université libre de Bruxelles.(memoire pour l'obtention de grade de maitre en informatqiue).

**[40]** : Nedjem Eddine Ayat. Sélection de modèle automatique des machines à vecteurs de support : application à la reconnaissance d'images de chiffres manuscrits. Université du Quebec 2004.( comme exigence partielle à l'obtention du doctorat en génie Ph. D.)

**[41]** : Pascal Vincent. Modèles à noyaux à structure locale. Université de Montréal. 2003. (thése presenté pour obtention grade Philosophiae Doctor (Ph. D.) en informatique.)

**[42]** : Guillaume Cleuziou. Une methode de classication non-supervisee pour l'apprentissage de regles et la recherche d'information. 2004.
(thése presentée pour l'obtention du grade de Docteur de l'Universite d'Orleans)

**[43]:** Safavian, S.R. Landgrebe, D. A survey of decision tree classifier methodology. Sch. of Electr. Eng., Purdue Univ., West Lafayette, IN;
Systems, Man and Cybernetics, IEEE Transactions. vol. 21(3) pp. 660-674. May/Jun 1991.

**Résumé**

Ce mémoire présente plusieurs intérêts pédagogiques et scientifiques. Apres un bref aperçus du KDD et de l'apprentissage automatique, le mémoire se concentre sur le data mining en présentant plusieurs méthodes de ce dernier, sans pour autant prétendre à l'exhaustivité. Ainsi dans ce mémoire sont présentées des techniques de classification ( k-means, bayes naïf, réseaux de neurones,..) de clustering (partitionnement, hiérarchiques, densité,…) de recherche d'association ( apriori, partitionné, comptage dynamique,…). Par la suite, le mémoire présente les SVM et leurs fondements théoriques. Le mémoire présente aussi plusieurs algorithmes des SVM. À la fin, le mémoire présente notre contribution dans le cadre des SVM et du data mining. Celle-ci consiste en une nouvelle approche d'utilisation des SVM dans le data mining.

**Mots clés :** ECD, extraction de connaissances à partir de données, fouille de données, apprentissage automatique, SVM, machine à vecteur de support, VC-dimension, SRM, minimisation risque structurel, classification, clustering, règles d'association, réseaux de neurones.

**Abstract**

This report presents many pedagogic and scientific interests. After a short sketch of KDD and machine learning, the report focus on the data mining, explaining several methods of this last, without pretending to exhaustiveness. So, in this report are explained techniques of classification (k-means, naive bayes, neural nets,.. ) of clustering (partitioning, hierarchical, density,..) of association searching(apriori, partitioned, dynamic counting,..). Afterward, the report explains the SVM and their theoretical foundations. The report explains also several SVM's algorithm. At the end, the report introduces our contribution in the framework of the SVM and the data mining. This one consists in a new approach of using the SVM in the data mining.

**Keywords :** KDD, knowledge discovery from databases, data mining, machine learning, SVM, support vector machine, VC-dimension, SRM, structural risque minimisation, classification, clustering, association rules, neural nets.