Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider - Biskra
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

Master 2 IOD

Cours Fouille de données avancée

Dr. Abdelhamid DJEFFAL
Site web : www.abdelhamid-djeffal.net

Année Universitaire 2016/2017

Plan du cours

Références

[1] J. Han, M. Kamber, and J. Pei. *Data mining : concepts and techniques.* Morgan Kaufmann Pub, 2011.

[2] Suhasini Itkar and Uday Kulkarni. Distributed sequential pattern mining : A survey and future scope. *International Journal of Computer Applications*, 94(18), 2014.

[3] M. Kantardzic. *Data mining : concepts, models, methods, and algorithms.* Wiley-Interscience, 2003.

[4] P. Preux. Fouille de données, notes de cours. *Disponible sur internet*, 2006.

Chapitre 1

Rappel des concepts de base

1.1 Définition de la fouille de données

La fouille de données est un domaine qui est apparu avec l'explosion des quantités d'informations stockées, avec le progrès important des vitesses de traitement et des supports de stockage. La fouille de données vise à découvrir, dans les grandes quantités de données, les informations précieuses qui peuvent aider à comprendre les données ou à prédire le comportement des données futures. Le datamining utilise depuis sont apparition plusieurs outils de statistiques et d'intelligence artificielle pour atteindre ses objectifs.

La fouille de données s'intègre dans le processus d'extraction des connaissances à partir des données ECD ou (KDD : Knowledge Discovery from Data en anglais). Ce domaine en pleine expansion est souvent appelé le data mining.

La fouille de données est souvent définie comme étant le processus de découverte des nouvelles connaissances en examinant de larges quantités de données (stockées dans des entrepôts) en utilisant les technologies de reconnaissance de formes de même que les techniques statistiques et mathématiques. Ces connaissances, qu'on ignore au début, peuvent être des corrélations, des patterns ou des tendances générales de ces données. La science et l'ingénierie modernes sont basées sur l'idée d'analyser les problèmes pour comprendre leurs principes et leur développer les modèles mathématiques adéquats. Les données expérimentales sont utilisées par la suite pour vérifier la correction du système ou l'estimation de quelques paramètres difficiles à la modélisation mathématiques. Cependant, dans la majorité des cas, les systèmes n'ont pas de principes compris ou qui sont trop complexes pour la modélisation mathématique. Avec le développent des ordinateurs, on a pu rassembler une très grande quantité de données à propos de ces systèmes. La fouille de données vise à

exploiter ces données pour extraire des modèles en estimant les relations entre les variables (entrées et sorties) de ses systèmes. En effet, chaque jour nos banques, nos hôpitaux, nos institutions scientifiques, nos magasins, ... produisent et enregistrent des milliards et des milliards de données. La fouille de données représente tout le processus utilisant les techniques informatiques (y compris les plus récentes) pour extraire les connaissances utiles dans ces données. Actuellement, La fouille de données utilise divers outils manuels et automatiques : on commence par la description des données, résumer leurs attributs statistiques (moyennes, variances, covariance,...), les visualiser en utilisant les courbes, les graphes, les diagrammes, et enfin rechercher les liens significatifs potentiels entre les variables (tel que les valeurs qui se répètent ensemble). Mais la description des données toute seule ne fournit pas un plan d'action. On doit bâtir un modèle de prédiction basé sur les informations découvertes, puis tester ce modèle sur des données autres que celles originales. La fouille de données a aujourd'hui une grande importance économique du fait qu'elle permet d'optimiser la gestion des ressources (humaines et matérielles). Elle est utilisée par exemple dans :

- organisme de crédit : pour décider d'accorder ou non un crédit en fonction du profil du demandeur de crédit, de sa demande, et des expériences passées de prêts ;
- optimisation du nombre de places dans les avions, hôtels, ...) surréservation
- organisation des rayonnages dans les supermarchés en regroupant les produits qui sont généralement achetés ensemble (pour que les clients n'oublient pas bêtement 'acheter un produit parce qu'il est situé à l'autre bout du magasin). Par exemple, on extraira une règle du genre : "les clients qui achètent le produit X en fin de semaine, pendant l'été, achètent généralement également le produit Y" ;
- organisation de campagne de publicité, promotions, ... (ciblage des offres)
- diagnostic médical : "les patients ayant tels et tels symptômes et demeurant dans des agglomérations de plus de 104 habitants développent couramment telle pathologie" ;
- analyse du génome
- classification d'objets (astronomie, ...)
- commerce électronique
- analyser les pratiques et stratégies commerciales et leurs impacts sur les ventes
- moteur de recherche sur internet : fouille du web
- extraction d'information depuis des textes : fouille de textes
- évolution dans le temps de données : fouille de séquences.

1.2 Processus du data mining

Il est très important de comprendre que le data mining n'est pas seulement le problème de découverte de modèles dans un ensemble de données. Ce n'est qu'une seule étape dans tout un processus suivi par les scientifiques, les ingénieurs ou toute autre personne qui cherche à extraire les connaissances à partir des données. En 1996 un groupe d'analystes définit le data mining comme étant un processus composé de cinq étapes sous le standard CRISP-DM (Cross-Industry Standard Process for Data Mining) comme schématisé ci-dessous :

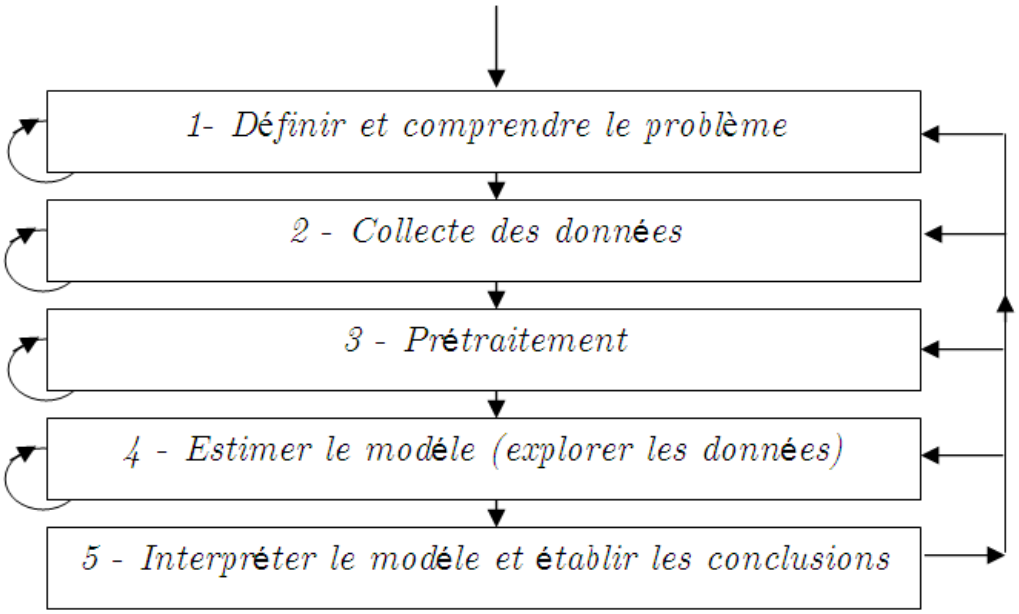


FIGURE 1.1 – Processus de data mining (CRISP-DM)

Ce processus, composé de cinq étapes, n'est pas linéaire, on peut avoir besoin de revenir à des étapes précédentes pour corriger ou ajouter des données. Par exemple, on peut découvrir à l'étape d'exploration (5) de nouvelles données qui nécessitent d'être ajoutées aux données initiales à l'étape de collection (2). Décrivons maintenant ces étapes :

1. Définition et compréhension du problème : Dans la plus part des cas, il est indispensable de comprendre la signification des données et le domaine à explorer. Sans cette compréhension, aucun algorithme ne va donner un résultat fiable. En effet, Avec la compréhension du problème, on peut préparer les données nécessaires à l'exploration et interpréter correctement les résultats obtenus. Généralement, le data mining est effectué dans un domaine particulier (banques, médecine, biologie, marketing, ...etc) où la connaissance et l'expérience dans ce domaine jouent un rôle très important dans

la définition du problème, l'orientation de l'exploration et l'explication des résultats obtenus. Une bonne compréhension du problème comporte une mesure des résultats de l'exploration, et éventuellement une justification de son coût. C'est-à-dire, pouvoir évaluer les résultats obtenus et convaincre l'utilisateur de leur rentabilité.

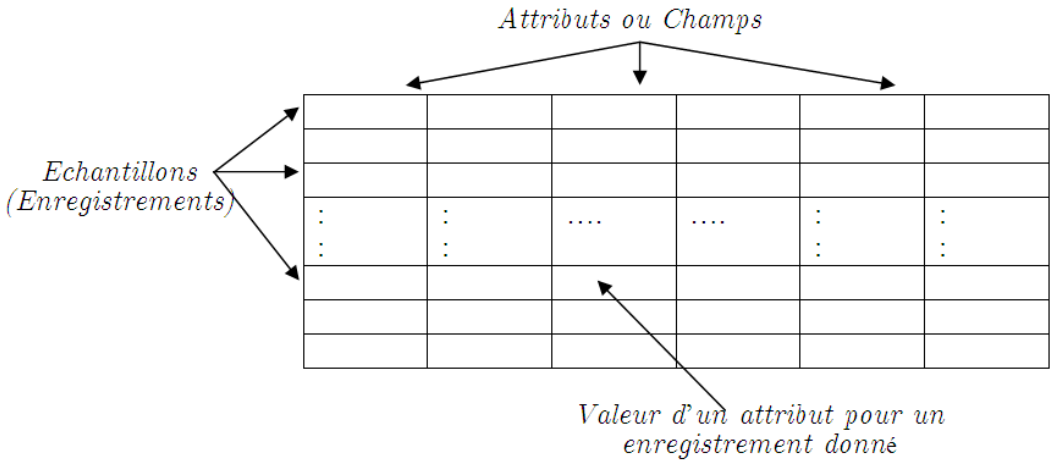
2. Collecte des données : dans cette étape, on s'intéresse à la manière dont les données sont générées et collectées. D'après la définition du problème et des objectifs du data mining, on peut avoir une idée sur les données qui doivent être utilisées. Ces données n'ont pas toujours le même format et la même structure. On peut avoir des textes, des bases de données, des pages web, ...etc. Parfois, on est amené à prendre une copie d'un système d'information en cours d'exécution, puis ramasser les données de sources éventuellement hétérogènes (fichiers, bases de données relationnelles, temporelles, ...). Quelques traitements ne nécessitent qu'une partie des données, on doit alors sélectionner les données adéquates. Généralement les données sont subdivisées en deux parties : une utilisée pour construire un modèle et l'autre pour le tester. On prend par exemple une partie importante (suffisante pour l'analyse) des données (80 %) à partir de laquelle on construit un modèle qui prédit les données futures. Pour valider ce modèle, on le teste sur la partie restante (20 %) dont on connaît le comportement.
3. Prétraitement : Les données collectées doivent être "préparées" [**?**]. Avant tout, elles doivent être nettoyées puisqu'elles peuvent contenir plusieurs types d'anomalies : des données peuvent être omises à cause des erreurs de frappe ou à causes des erreurs dues au système lui-même, dans ce cas il faut remplacer ces données ou éliminer complètement leurs enregistrements. Des données peuvent être incohérentes c-à-d qui sortent des intervalles permis, on doit les écarter où les normaliser. Parfois on est obligé à faire des transformations sur les données pour unifier leur poids. Un exemple de ces transformations est la normalisation des données qui consiste à la projection des données dans un intervalle bien précis [0,1] ou [0,100] par exemple. Un autre exemple est le lissage des données qui considère les échantillons très proches comme étant le même échantillon. Le prétraitement comporte aussi la réduction des données [**?**] qui permet de réduire le nombre d'attributs pour accélérer les calculs et représenter les données sous un format optimal pour l'exploration. Une méthode largement utilisée dans ce contexte, est l'analyse en composantes principales (ACP). Une autre méthode de réduction est celle de la sélection et suppression des attributs dont l'importance dans la caractérisation des données est faible, en mesurant leurs

variances. On peut même réduire le nombre de données utilisées par le data mining en écartant les moins importantes. Dans la majorité des cas, le pré-traitement doit préparer des informations globales sur les données pour les étapes qui suivent tel que la tendance centrale des données (moyenne, médiane, mode), le maximum et le minimum, le rang, les quartiles, la variance, ... etc. Plusieurs techniques de visualisation des données telles que les courbes, les diagrammes, les graphes,... etc, peuvent aider à la sélection et le nettoyage des données. Une fois les données collectées, nettoyées et prétraitées on les appelle entrepôt de données (data warehouse).

4. Estimation du modèle : Dans cette étape, on doit choisir la bonne technique pour extraire les connaissances (exploration) des données. Des techniques telles que les réseaux de neurones, les arbres de décision, les réseaux bayésiens, le clustering, ... sont utilisées. Généralement, l'implémentation se base sur plusieurs de ces techniques, puis on choisit le bon résultat. Dans le reste de ce rapport on va détailler les différentes techniques utilisées dans l'exploration des données et l'estimation du modèle.
5. Interprétation du modèle et établissement des conclusions : généralement, l'objectif du data mining est d'aider à la prise de décision en fournissant des modèles compréhensibles aux utilisateurs. En effet, les utilisateurs ne demandent pas des pages et des pages de chiffres, mais des interprétations des modèles obtenus. Les expériences montrent que les modèles simples sont plus compréhensibles mais moins précis, alors que ceux complexes sont plus précis mais difficiles à interpréter.

1.3 Quel type de données fouiller ?

Le composant de base d'un processus de data mining est l'ensemble d'échantillons représentants les données à explorer. Chaque échantillon est présenté sous forme de ligne caractérisée par un ensemble d'attributs. Dans le cas des bases de données un échantillon est un enregistrement composé d'un ensemble de champs. Généralement, il convient de représenter les enregistrements sous forme de points dans un espace de m dimensions où m est le nombre d'attributs.



Une donnée est, donc, un enregistrement au sens des bases de données, que l'on nomme aussi "individu" (terminologie issue des statistiques) ou "instance" (terminologie orientée objet en informatique) ou même "tuple" (terminologie base de données) et "point" ou "vecteur" parce que finalement, d'un point de vue abstrait, une donnée est un point dans un espace euclidien ou un vecteur dans un espace vectoriel. Une données est caractérisée par un ensemble de "champs", de "caractères", ou encore d' "attributs" (en suivant les 3 terminologies précédemment évoquées : bases de données, statistiques et conception orientée objet).

Les attributs ou les champs sont de deux types : numériques où catégoriels. Les attributs numériques qui comportent les variables réelles ou entières tel que la longueur, le poids, l'âge, ... sont caractérisés par une relation d'ordre ($5 < 7.5$) et une mesure de distance ($D(5, 7.5) = 2.5$). Les attributs catégoriels (appelées aussi symboliques) tel que la couleur, l'adresse ou le groupe sanguin ne possèdent aucune de ces caractéristiques. Deux variables catégorielles ne peuvent être qu'égales ou différentes. Il est clair que la relation d'ordre dans le cas des attributs numériques permet de calculer dans un ensemble d'enregistrements, un max, un min, une moyenne, une distance, ...etc. Alors que dans le cas d'attributs catégoriels ça sera impossible : comment calculer la moyenne, la variance ou la distance entre des adresses ? Dans ce cas, de nouvelles mesures doivent être développées pour chaque technique de fouille de données. Théoriquement, plus le nombre d'échantillons est important, meilleure est la précision de l'analyse. Mais en pratique, beaucoup de difficultés peuvent être rencontrées avec les bases de données gigantesques (des milliards d'enregistrements ou des Gigabytes). En effet, Les bases de données de nos jours sont immenses au point où elles épuisent même les supports de stockage, et nécessitent pour être analysées les machines les plus puissantes et les techniques les plus performantes. Un premier problème avec les

bases de données immenses, est celui de leur préparation à l'analyse, puisque la qualité des données analysées influence directement sur les résultats d'analyse. La préparation doit prendre compte d'un certain nombre de points :

- Les données doivent être précises : les noms doivent être écrits correctement, les valeurs doivent être dans les bons intervalles et doivent être complètes,
- Les données doivent être enregistrées dans les bon formats : une valeur numérique ne doit pas être enregistrée sous format caractère, une valeur entière ne doit pas être réelle,...etc,
- La redondance doit être éliminée ou au moins minimisée,
- ...etc.

Dans le cas d'un nombre limité d'échantillons, la préparation peut être semi-automatique ou même manuelle, mais dans notre cas d'immenses BDD la préparation automatique s'impose et des techniques automatiques de vérification et normalisation doivent intervenir. Le problème majeur des BDD immenses apparaît lors de leur exploration, le temps d'exploration et la précision doivent être pris en compte. Toutes les techniques d'exploration qu'on va voir fixent des critères d'arrêt soit sur le temps d'exécution ou sur la précision des résultats atteints.

Les enregistrements sont regroupés dans des tables et dans des bases de données de différents types et l'analyse effectuée et le choix de ses outils dépendent fortement du type de la base de données à analyser. En fait, les bases de données relationnelles, les bases de données transactionnelles, les systèmes avancés de bases de données, les streams et les bases de données spatiales représentent les types les plus utilisés.

1.4 Les tâches de la fouille de données

Beaucoup de problèmes intellectuels, économiques ou même commerciaux peuvent être exprimés en termes des six tâches suivantes :

- La classification.
- L'estimation.
- Le groupement par similitude (règles d'association).
- L'analyse des clusters.
- La description.

Les trois premières tâches sont des exemples de la fouille supervisée de données dont le but est d'utiliser les données disponibles pour créer un modèle décrivant une variable

particulière prise comme but en termes de ces données. Le groupement par similitude et l'analyse des clusters sont des tâches non-supervisées où le but est d'établir un certain rapport entre toutes La description appartient à ces deux catégories de tâche, elle est vue comme une tâche supervisée et non-supervisée en même temps.

- Classification

  La classification est la tache la plus commune de la fouille de données qui semble être une tâche humaine primordiale. Afin de comprendre notre vie quotidienne, nous sommes constamment obligés à classer, catégoriser et évaluer. La classification consiste à étudier les caractéristiques d'un nouvel objet pour l'attribuer à une classe prédéfinie. Les objets à classifier sont généralement des enregistrements d'une base de données, la classification consiste à mettre à jours chaque enregistrement en déterminant la valeur d'un champ de classe. Le fonctionnement de la classification se décompose en deux phases. La première étant la phase d'apprentissage. Dans cette phase, les approches de classification utilisent un jeu d'apprentissage dans lequel tous les objets sont déjà associés aux classes de références connues. L'algorithme de classification apprend du jeu d'apprentissage et construit un modèle. La seconde phase est la phase de classification proprement dite, dans laquelle le modèle appris est employé pour classifier de nouveaux objets.

- L'estimation

  L'estimation est similaire à la classification à part que la variable de sortie est numérique plutôt que catégorique. En fonction des autres champs de l'enregistrement l'estimation consiste à compléter une valeur manquante dans un champ particulier. Par exemple on cherche à estimer la lecture de tension systolique d'un patient dans un hôpital, en se basant sur l'âge du patient, son genre, son indice de masse corporelle et le niveau de sodium dans son sang. La relation entre la tension systolique et les autres données vont fournir un modèle d'estimation. Et par la suite nous pouvons appliquer ce modèle dans d'autres cas.

- Le groupement par similitude (Analyse des associations et de motifs séquentiels) Le groupement par similitude consiste à déterminer quels attributs "vont ensemble". La tâche la plus répandue dans le monde du business, est celle appelée l'analyse d'affinité ou l'analyse du panier du marché, elle permet de rechercher des associations pour mesurer la relation entre deux ou plusieurs attributs. Les règles d'associations sont, généralement, de la forme "Si <antécédent>, alors <conséquent>".

– L'analyse des clusters

  Le clustering (ou la segmentation) est le regroupement d'enregistrements ou des observations en classes d'objets similaires. Un cluster est une collection d'enregistrements similaires l'un à l'autre, et différents de ceux existants dans les autres clusters. La différence entre le clustering et la classification est que dans le clustering il n'y a pas de variables sortantes. La tâche de clustering ne classifie pas, n'estime pas, ne prévoit pas la valeur d'une variable sortantes. Au lieu de cela, les algorithmes de clustering visent à segmenter la totalité de données en des sous groupes relativement homogènes. Ils maximisent l'homogénéité à l'intérieur de chaque groupe et la minimisent entre les différents groupes.

– La description Parfois le but de la fouille est simplement de décrire se qui se passe sur une Base de Données compliquée en expliquant les relations existantes dans les données pour premier lieu comprendre le mieux possible les individus, les produit et les processus présents dans cette base. Une bonne description d'un comportement implique souvent une bonne explication de celui-ci. Dans la société Algériennes nous pouvons prendre comme exemple comment une simple description, "les femmes supportent le changement plus que les hommes", peut provoquer beaucoup d'intérêt et promouvoir les études de la part des journalistes, sociologues, économistes et les spécialistes en politiques.

Chapitre 2

Recherche des modèles fréquents, corrélations et associations

Les motifs fréquents sont des motifs ou patterns (tel que les ensembles d'items, les sous séquences, ou les sous structures) qui apparaissent fréquemment dans un ensemble de données. Par exemple, un ensemble d'items tel que le lait et le pain qui apparaissent souvent dans une base de transactions dans un supermarché, est un ensemble d'items fréquent. Une sous séquence telle que acheter premièrement un PC puis une caméra numérique ensuite une carte mémoire qui se produit souvent dans la base historique des achats, est une séquence d'items fréquente. Les sous structures peuvent être des sous-graphes ou des sous-arbres qui peuvent être combinés avec des ensembles ou des séquences d'items. Trouver de tels motifs fréquents joue un rôle essentiel dans la fouille des associations et des corrélations, et représente une tâche importante en data mining et constitue toujours un thème qui attire beaucoup de recherches.

L'analyse des motifs fréquents trouve son application dans plusieurs domaines :

- L'analyse du panier du marché, pour comprendre les habitudes des clients afin de mieux organiser les rayons d'articles, organiser les promotions, ...etc.
- L'analyse d'ADN en biologie afin de comprendre les propriétés génétiques des espèces.
- L'analyse du climat en météorologie afin de mieux orienter l'agriculture ou choisir l'orientation des pistes des aérodromes.
- ...

2.1 Concepts de base

2.1.1 Base de données formelle

La version de base de l'extraction de motifs fréquents permet de faire la fouille dans une table d'une base de données relationnelle dont les valeurs sont des booléens indiquant la présence ou l'absence d'une propriété. Une telle base est appelée base de données formelle.

Une base de données formelle est définie par un triplet (O, P, R) où :

- O est un ensemble fini d'objets.
- P est un ensemble fini de propriétés.
- R est une relation sur $O \times P$ qui permet d'indiquer si un objet x a une propriété p (noté xRp) ou non.

Par exemple dans le cas d'analyse du panier dans un supermarché, O est l'ensemble des transactions d'achat, P est l'ensemble d'articles et R est la relation indiquant si un article a est acheté dans la transaction t.

Considérons par exemple la base de données formelle suivante :

R	a	b	c	d	e
x_1	1	0	1	1	0
x_2	0	1	1	0	1
x_3	1	1	1	0	1
x_4	0	1	0	0	1
x_5	1	1	1	0	1
x_6	0	1	1	0	1

- $O = \{x_1, x_2, x_3, x_4, x_5, x_6\}$.
- $P = \{a, b, c, d, e\}$.
- xRp si et seulement si la ligne de x et la colonne de p se croisent sur un 1 (et pas sur un 0), par exemple :x_1Ra, x_1Rc et x_1Rd.

2.1.2 Motif

Un motif d'une base de données formelle (O, P, R) est un sous-ensemble de P. L'ensemble de tous les motifs d'une base est donc l'ensemble des parties de P, noté 2^P. On dira qu'un objet $x \in O$ possède un motif m si $\forall\ p \in m,\ xRp$. Pour la base de données en exemple, on a donc :

- Motif de taille 0 = $\emptyset$ ($C_5^0 = 0$ motifs).
- Motifs de taille 1 = $\{a\}, \{b\}, \{c\}, \{d\}$ et $\{e\}$, qu'on notera, pour simplifier, $\underline{a}, \underline{b}, \underline{c}, \underline{d}$ et $\underline{e}$. ($C_5^1 = 5$ motifs).
- Motifs de taille 2 = $\underline{ab}, \underline{ac}, \underline{ad}, \underline{ae}, \underline{bc}, \underline{bd}, \underline{be}, \underline{cd}, \underline{ce}, \underline{de}$ ($C_5^2 = 10$ motifs)
- Motifs de taille 3 = $\underline{abc}, \underline{abd}, \underline{abe}, \underline{acd}, \underline{ace}, \underline{ade}, \underline{bcd}, \underline{bce}, \underline{bde}, \underline{cde}$ ($C_5^3 = 10$ motifs).
- Motifs de taille 4 = $\underline{abcd}, \underline{abce}, \underline{abde}, \underline{acde}, \underline{bcde}$ ($C_5^4 = 5$ motifs).
- Motifs de taille 5 = $\underline{abcde}$ ($C_5^5 = 1$ motifs).

Dans la base formelle précédente, x_1 possède les motifs :$\emptyset, \underline{a}, \underline{c}, \underline{d}, \underline{ac}, \underline{ad}, \underline{cd}$ et $\underline{acd}$. Parmi l'ensemble global de 2^p motifs, on va chercher ceux qui apparaissent fréquemment. Pour cela, on introduira les notions de connexion de Galois et de support d'un motif.

2.1.3 Connexion de Galois

La connexion de Galois associée à une base de données formelle (O, P, R) est le couple de fonctions (f, g) définies par :

$$\begin{cases} f: & 2^p \to 2^o \\ & m \to f(m) = \{x \in O/\ x \text{ possède } m\} \\ g: & 2^o \to 2^p \\ & X \to g(X) = \{p \in P/\ \forall x \in X\ xRp\} \end{cases}$$

g est dite duale de f et f duale de g. On dit parfois que $f(m)$ est l'image du motif m.

Exemples :

$$\begin{array}{ll} f(\underline{bc}) = \{x_2, x_3, x_5, x_6\} & g(x_2, x_3, x_5, x_6) = \{b, c, e\} \\ g(x_1) = \{a, c, d\} & g(\underline{acd}) = \phi \\ f(\underline{a}) = \{x_1, x_3, x_5\} & g(x_1, x_3, x_5) = \{a, c\} \end{array}$$

2.1.4 Support d'un motif

Soit $m \in 2^P$, un motif. Le support de m est la proportion d'objets dans O qui possèdent le motif :

$$\begin{aligned} Support: \quad & 2^p \to [0, 1] \\ & m \to Support(m) = \frac{|f(m)|}{|O|} \end{aligned}$$

Par exemple dans la base précédente, on a :

$$Support(\underline{a}) = \frac{3}{6},$$

$$Support(\underline{b}) = \frac{5}{6},$$
$$Support(\underline{ab}) = \frac{2}{6},$$
$$Support(\emptyset) = 1,$$
$$Support(P) = 0.$$

Propriété fondamentale : Le support est décroissant de $(2^p, \subseteq)$ dans $([0,1], \leq)$. Autrement dit, si m est un sous-motif de m' ($m \subseteq m'$) alors $Support(m) \geq Support(m')$. Le support mesure la fréquence d'un motif : plus il est élevé, plus le motif est fréquent. On distinguera les motifs fréquents des motifs non fréquents à l'aide d'un seuil σ_s.

2.1.5 Motif fréquent

Soit $\sigma_s \in [0,1]$. Un motif m est fréquent (sous-entendu, relativement au seuil σ_s) si $Support(m) \geq \sigma_s$. Sinon, il est dit non fréquent.

2.2 Méthodes efficaces pour la recherche des modèles fréquents

Une approche naïve pour l'extraction des motifs fréquents consiste à parcourir l'ensemble de tous les motifs, à calculer leurs nombres d'occurrences (support) et à ne garder que les plus fréquents. Malheureusement, cette approche est trop consommatrice en temps et en ressources. En effet, le nombre de motifs est 2^p (p est le nombre de propriétés), et en pratique, on veut manipuler des bases ayant un grand nombre d'attributs. L'algorithme Apriori proposé par Agrawal et ses co-auteurs en 1994 est un algorithme de base qui permet d'extraire des motifs fréquents dans une base ayant plusieurs milliers d'attributs et plusieurs millions d'enregistrements. L'idée est d'effectuer une extraction par niveaux selon le principe suivant :

- On commence par chercher les motifs fréquents de longueur 1 ;
- On combine ces motifs pour obtenir des motifs de longueur 2 et on ne garde que les fréquents parmi eux ;
- On combine ces motifs pour obtenir des motifs de longueur 3 et on ne garde que les fréquents parmi eux ;
- ... continuer jusqu'à la longueur maximale.

Cette approche s'appuie sur les deux principes fondamentaux suivants (qui reposent sur la décroissance du support) :

1. Tout sous-motif d'un motif fréquent est fréquent.
2. Tout sur-motif d'un motif non fréquent est non fréquent.

L'algorithme de référence basé sur cette approche est l'algorithme Apriori. Le pseudo-code suivant décrit l'extraction de motifs fréquents selon ce principe :

Algorithme 1 Apriori

Entrées: Base de données de transactions D, Seuil de support minimum σ

Sorties: Ensemble des items fréquents

$i \leftarrow 1$

$C_1 \leftarrow$ ensemble des motifs de taille 1 (un seul item)

tantque $C_i \neq \phi$ **faire**

Calculer le Support de chaque motif $m \in C_i$ dans la base

$F_i \leftarrow \{m \in C_i | support(m) \geq \sigma\}$

$C_{i+1} \leftarrow$ toutes les combinaisons possibles des motifs de F_i de taille $i+1$

$i \leftarrow i+1$

fin tantque

retourner $\cup_{(i \geq 1)} F_i$

Exemple :

L'application de l'algorithme sur la base donnée en exemple avec $\sigma = 0.25$ se passe comme suit :

1. Génération de candidats de taille 1 :
   - $C_1 = \{a, b, c, d, e\}$
   - Supports : $\frac{3}{6}, \frac{5}{6}, \frac{5}{6}, \frac{1}{6}, \frac{5}{6}$

   D'où $F_1 = \{a, b, c, e\}$ (aucun motif fréquent ne contiendra d).
2. Génération de candidats de taille 2 : Combiner 2 à 2 les candidats de taille 1 de F_1 :
   - $C_2 = \{ab, ac, ae, bc, be, ce\}$
   - Supports : $\frac{2}{6}, \frac{3}{6}, \frac{2}{6}, \frac{4}{6}, \frac{5}{6}, \frac{4}{6}$

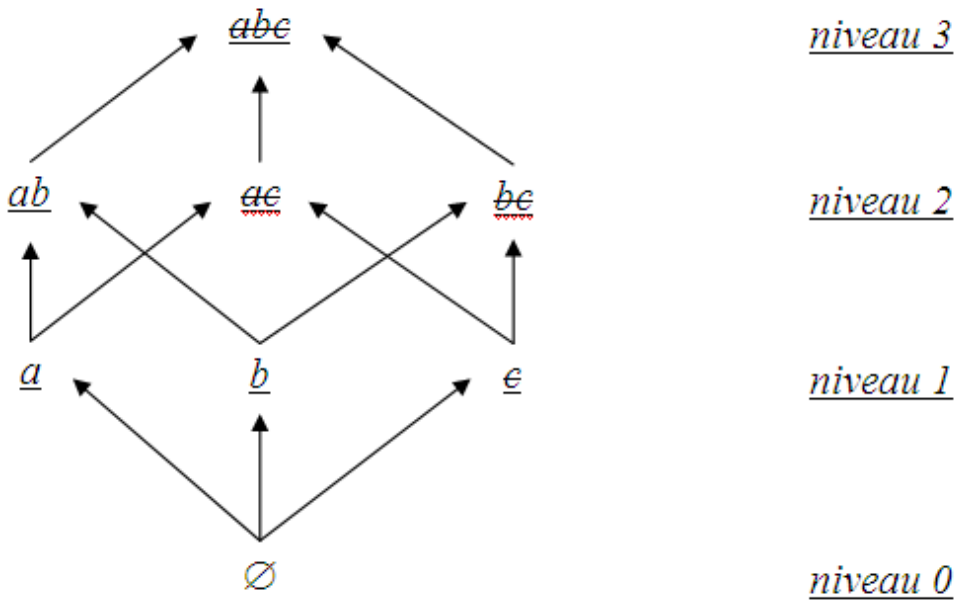
   $F_2 = C_2$: tous les motifs de C_2 sont fréquents.
3. Génération de candidats de taille 3 : Combiner 2 à 2 les candidats de taille 2 de F_2 (et ne considérer que ceux qui donnent des motifs de taille 3) :
   - $C3 = abc, abe, ace, bce$
   - Supports : $\frac{2}{6}, \frac{2}{6}, \frac{2}{6}, \frac{4}{6}$

   $F_3 = C_3$: tous les motifs de C3 sont fréquents.

4. Génération de candidats de taille 4 :
   - $C4 = \{abce\}$
   - Supports : $\frac{2}{6}$
5. Génération de candidats de taille 5 : $C_5 = \emptyset$. Donc, $F_5 = \emptyset$
6. L'algorithme retourne alors l'ensemble des motifs fréquents : $F_1 \cup F_2 \cup F_3 \cup F_4$

Remarque 1 : On peut voir cet algorithme comme le parcours du treillis des parties de P ordonné pour l'inclusion.

Supposons par exemple que $P = \{a, b, c\}$. Le treillis des parties de P est :



Il est parcouru par niveau croissant à partir du niveau $i = 1$. Quand un motif n'est pas fréquent, tous ses sur-motifs sont non fréquents. Dans notre exemple c n'est pas fréquent (il a été barré) et, par conséquent, aucun de ses sur-motifs n'est considéré. On a ainsi élagué le parcours du treillis.

Remarque 2 : Un des objectifs des optimisations de cet algorithme est de diminuer le nombre d'accès à la base de données.

Remarque 3 : Le seuil σ est fixé par l'analyste. Celui-ci peut suivre une approche itérative en fixant un seuil au départ et, en fonction du résultat, changera la valeur du seuil : Si trop de motifs fréquents ont été trouvés, il augmentera le seuil ; dans le cas inverse, il le diminuera. Par ailleurs, on peut constater que le temps de calcul de l'algorithme Apriori décroit avec le seuil. Par conséquent, si l'analyste fixe une valeur de seuil trop grande, cela gaspillera moins de temps que s'il en fixe un trop petit.

2.2.1 Optimisations

2.2.1.1 L'algorithme AprioriTlD

L'algorithme Apriori nécessite N passes sur la base, N étant la taille du plus grand ensemble susceptible d'être fréquent. Une optimisation possible consiste à générer en mémoire, pendant la première passe, les identifiants (TID) des transactions pour chaque 1-itemset (ensemble de motifs de taille 1) fréquent. Dans la suite, les listes de TID correspondant à chaque k-itemset sont gardées. Le calcul d'un k-itemset se fait toujours à partir des deux (k-1)-itemsets contenant un élément de moins, mais le comptage se fait simplement par intersection des deux listes de TID des deux (k-1)-itemsets source. Nous construisons la liste des TID après avoir déterminé les 1-itemsets fréquents, ce qui est plus efficace en taille mémoire mais nécessite deux passes. La première passe permet d'éliminer les produits non fréquents et réduit donc les listes de TID en mémoire construites dans une deuxième passe. Générer les listes de TID en mémoire en parallèle dès la première passe est plus efficace. Cependant, il n'est possible d'éliminer des listes qu'après le comptage total de la base, lorsqu'on sait qu'un 1-itemset ne sera pas fréquent. L'algorithme nécessite alors une taille mémoire de N*P TID, N étant le nombre de transactions et P le nombre de produits. Il devient très inefficace si les listes ne tiennent pas en mémoire.

2.2.1.2 L'algorithme apriori partitionné

L'algorithme proposé par Savasere permet de résoudre le problème de place mémoire de l'algorithme précédent. L'idée est simplement de diviser la base en Q partitions, de sorte que chaque partition tienne en mémoire. Chaque partition est traitée indépendamment et les ensembles fréquents sont découverts pour chaque partition. La validité de l'algorithme est basée sur le lemme suivant : pour être globalement fréquent, un ensemble doit être fréquent dans au moins une partition. Ayant obtenu les ensembles fréquents sur chaque partition, il suffit dans une passe finale d'explorer l'union des ensembles fréquents sur la base. L'opération se fait par un comptage simple sur la base. Les ensembles non globalement fréquents, mais localement fréquents dans une partition, sont ainsi éliminés. L'avantage de cet algorithme est qu'il nécessite deux passes au plus. Il est aussi facilement parallélisable, les partitions pouvant être traitées indépendamment. Dans ce cas, il faut cependant beaucoup de mémoire pour tenir les partitions et listes de TID en mémoire.

2.2.1.3 Algorithme de comptage dynamique

L'algorithme DIC a été proposé par Brin et al. Pour réduire le nombre de parcours de la base de données. DIC partitionne la base de données en blocs de M transactions. Durant le calcul des supports des k-itemsets, après le parcours d'une partition de taille M de D, on vérifie les k-itemsets candidats qui ont déjà atteint le support minimum, DIC les utilise alors pour générer des candidats de taille (k+1), et commence à compter leurs supports. Ainsi les supports de candidats de tailles différentes sont calculés durant les mêmes parcours de D. En contrepartie de la diminution du nombre de balayages de la base de données, DIC considère des itemsets candidats de tailles différentes simultanément. Ceci pose le problème de stockage des itemsets candidats traités simultanément et du coût de calcul des supports des candidats qui est plus important que pour Apriori.

2.3 Types de motifs fréquents

Le nombre de motifs présents dans une base de données est généralement très grand, il est courant d'en obtenir plusieurs millions. Leur utilisation repose classiquement sur une démarche de sélection par la fréquence à fin de ne conserver que les plus représentatifs, mais aussi parce que c'est une façon pragmatique efficace d'élaguer l'espace de recherche. Néanmoins cette approche s'avère insuffisante dans de nombreux cas pratiques et montre ses limites lorsque l'espace de recherche est important ou la base très dense et corrélée.

Les représentations condensées de motifs fournissent une solution au problème de l'extraction de motifs fréquents en proposant un résumé des motifs fréquents, tout en assurant de pouvoir reconstruire l'ensemble des motifs fréquents si nécessaire. En général, on en trouve deux types :

2.3.1 Motif fréquent fermé

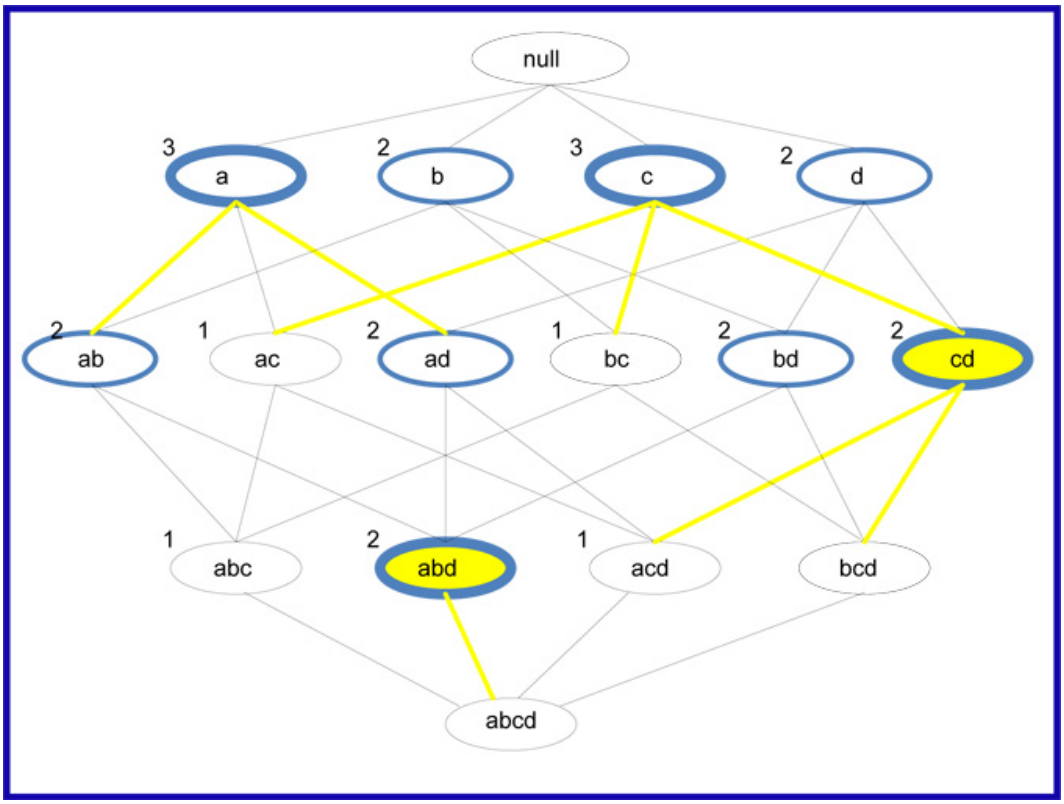
Un motif fréquent est dit fermé s'il ne possède aucun sur-motif qui a le même support.

2.3.2 Motif fréquent maximal

Un motif fréquent est dit Maximal si aucun de ses sur-motifs immédiats n'est fréquent.

Exemple

Le schéma suivant illustre la relation entre le motifs fréquents, fréquents fermés et fréquents maximaux :



– Les motifs encerclés par les lignes minces ne sont pas fréquents, les autres le sont.

– Les motifs encerclés par des lignes plus épais sont fermés.

– Les motifs colorés sont maximaux.

Il est clair que :

$$\text{Les motifs maximaux} \subset \text{Les motifs fermés} \subset \text{Les motifs fréquents}$$

2.4 Passage aux règles d'association

La phase qui suit la phase de recherche des motifs fréquents est la phase de découverte des règles d'association basées sur tous les items fréquents trouvés. Cette phase est relativement simple, elle consiste à trouver toutes les règles qui existent entre les items fréquents. Par exemple pour une règle telle que $\{x_1, x_2, x_3\} \Rightarrow x_4$, il faut premièrement que $\{x_1, x_2, x_3\}$ et x_4 soient fréquents. Ensuite, il faut que la confidence de la règle dépasse un certain seuil. La confidence est calculée comme suit :

$$\begin{aligned} Confidence(\{x_1, x_2, x_3\} \Rightarrow x_4) &= P(x_4 / \{x_1, x_2, x_3\}) \\ &= \frac{Support(\{x_1, x_2, x_3\} \cup x_4)}{Support(\{x_1, x_2, x_3\})} \end{aligned}$$

Où le $Support(x)$ est le nombre d'enregistrements où apparait l'item x,
et le $Support(\{x_1, x_2, x_3\})$ est le nombre d'enregistrement où apparaissent les items x_1, x_2, x_3 ensemble.
L'ensemble des règles d'association peut être trouvé en calculant la confidence de toutes les combinaisons possibles des items fréquents puis prendre celles dont la confidence est importante. Cette opération peut être accélérée en enregistrant la liste des items fréquents avec leurs fréquences dans une table qui peut être accédée rapidement.

Définition d'une règle d'association

Soit m_1 et m_2 deux motifs. Une règle d'association est une implication de la forme :

$$m_1 \Rightarrow m_2$$
$$\text{Où} \quad m_1, m_2 \in 2^P, \quad et \quad m_1 \cap m_2 = \emptyset$$

2.4.0.1 Support d'une règle

La règle $m_1 \Rightarrow m_2$ est vérifiée dans la base de donnée D avec un support s, où s est le pourcentage d'objets dans D contenant $m_1 \cup m_2$:

$$Support(m_1 \Rightarrow m_2) = \frac{Nombre\ de\ transactions\ contenant(m_1 \cup m_2)}{Nombre\ total\ de\ transactions}$$

2.4.0.2 Qualité d'une règle

La qualité de la règle représente une mesure de la cooexistance des deux paries (gauche et droite) de la règle. Plusieurs métriques sont utilisées :

- **Confidence :** La confidence de la règle $m_1 \Rightarrow m_2$ est définie par le pourcentage de transactions qui contiennent $m_1 \cup m_2$ dans les transaction contenant m_1.

$$Conf(m_1 \Rightarrow m_2) = \frac{Support(m_1 \cup m_2)}{Support(m_1)} = \frac{Nombre\ de\ transactions\ contenant(m_1 \cup m_2)}{Nombre\ de\ transactions\ contenant\ m_1}$$

- **Lift :** c'est l'amélioration

$$Lift(m_1 \Rightarrow m_2) = \frac{Support(m_1 \cup m_2)}{Support(m_1) \times Support(m_1)}$$

  C'est le rapport entre le support observé et celui attendu si m_1 et m_2 étaient indépendants.

Si le Lift d'une règle est égal à 1, cela signifie que les occurrences de ses deux parties sont totalement indépendantes. Si par contre le lift est supérieur à 1, il nous informe du degré de dépendance des deux parties et leur candidature pour les futures prédictions.

Le lift considère à la fois la confidence de la règle et la base toute entière.

- **Leverage :** Il représente la différence entre la fréquence observée d'apparition des deux parties ensembles et leur apparition ensemble attendues s'il étaient indépendants.

$$Leverage(m_1 \Rightarrow m_2) = Support(m_1 \cup m_2) - Support(m_1) \times Support(m_2)$$

- **Conviction :**

  La conviction est calculé par la formule :

$$Conv(m_1 \Rightarrow m_2) = \frac{1 - Support(m_2)}{1 - Conf(m_1 \Rightarrow m_2)}$$

  Elle représente le rapport entre la fréquence d'apparition de X sans Y (taux d'erreur de la règle), et le taux des prédictions erronés.

Les règles qui dépassent un minimum de support et un minimum de qualité sont appelées règles **solides**.

Une fois les items fréquents dans une base de donnée sont extraits, il devient simple de générer les règles d'association qui vérifient un minimum de support et un minimum de confidence, comme suit :

- Pour chaque motif fréquent l, générer tous les sous ensembles non vides de l ,
- Pour chaque sous-ensemble non vide s de l, enregistrer la règle (s ⇒ l-s) si :

$$Confidence(s \Rightarrow l - s) \geq Min\_Conf$$

  Où Min_Conf est un seuil minimum de confidence.

Puisque les règles sont générées des motifs fréquents, chacune vérifie automatiquement le support minimum.

2.5 Analyse des corrélation

La plupart des algorithmes de recherche des règles d'association utilise un seuil minimum de confidence. Cependant, plusieurs règles intéressantes peuvent être trouvées en

utilisant un seuil très faible. Bien que l'utilisation d'un support minimum empêche l'exploration d'un grand nombre de règle intéressantes, plusieurs règles ainsi trouvées restent inutiles pour l'utilisateur.

On s'intéresse maintenant à la recherche de paires d'items dont le support est faible mais dont la présence est fortement corrélée. Les données peuvent être vues comme une matrice $X = (x_{i,j})$ dont chaque ligne représente un individu et chaque colonne représente un item. Typiquement la matrice $x_{i,j}$ est très creuse : pour chaque individu, il n'y qu'une faible proportion d'items présents (souvent bien moins qu'1 %). On cherche des paires de produits souvent présents ensemble, *i.e.* des colonnes similaires. Le problème à résoudre est donc de trouver le plus efficacement possibles ces paires de colonnes similaires, en faisant l'hypothèse que les items apparaissent rarement (support faible). On peut essayer d'utiliser l'algorithme A-Priori vu plus haut mais il est mal adapté à la recherche d'items de support faible ; on peut aussi essayer d'imaginer des méthodes plus spécifiques. C'est ce que nous décrivons ci-dessous. On suppose par ailleurs que :

- P est suffisamment petit pour que l'on puisse stocker une information (un nombre) concernant chaque colonne en mémoire centrale, mais on ne peut pas stocker en mémoire centrale pour chaque paire d'attributs ;
- O est tellement grand que l'on ne peut pas stocker toute la matrice en mémoire centrale, même en tenant compte du fait que la matrice est creuse, et même en la compressant ;
- on cherche une alternative à l'utilisation des critères utilisés plus haut (support, confidence) car l'ensemble de paires candidates est trop grand pour être énuméré en un temps raisonnable.

2.5.1 Calcul de la corrélation

On commence par préciser ce que l'on entend par "corrélation" de deux colonnes. Informellement, deux colonnes sont similaires si elles ont généralement des 1 aux mêmes lignes. On mesure la corrélation de deux colonnes $x_{.,j}$ et $x_{.,k}$ par le rapport entre le nombre de lignes où $x_{i,j}$ et $x_{i,k}$ sont égaux à 1 en même temps par le nombre de lignes où l'un des deux seulement vaut 1 :

$$Cor(x_{.j}, x_{.k}) = \frac{|x_{i,j} \wedge x_{i,k}|}{|x_{i,j} \vee x_{i,k}|}$$

La Complexité du calcul de Cor : (O) pour deux colonnes données ; il y a $\mathrm{O}(P^2)$ paires de colonnes ; donc $\mathrm{O}(O \times P^2)$ pour l'ensemble des données.

En prenant $O = 106$ et $P = 105$, $OP^2 = 10^{16}$. Sur un ordinateur capable d'effectuer 1 million de comparaisons par seconde, le calcul de Cor demande 1010 secondes, soit 10 milliards de secondes, soit ... 300 ans !

La signature permet de surmonter ce problème de complexité. On définit le type de deux colonnes pour une ligne donnée par a, b, c ou d comme suit :

Type	$x_{i,j}$	$x_{i,k}$
a	1	1
b	1	0
c	0	1
d	0	0

Pour une paire de colonnes, On note respectivement a, b, c et d le nombre de lignes de type a, b, c et d. On a :

$$Cor(x_{.,j}, x_{.,k}) = \frac{|x_{i,j} \wedge x_{i,k}|}{|x_{i,j} \vee x_{i,k}|} = \frac{a}{a+b+c+d}$$

Les applications de cette recherche de paires fortement corrélées à faible support sont :

- lignes et colonnes sont des pages web et $x_{i,j} = 1$ si la page i pointe sur la page j. Dans ce cas, des colonnes similaires peuvent être des pages traitant d'un même sujet : ces pages sont pointées par les mêmes pages ;
- lignes et colonnes sont des pages web et $x_{i,j} = 1$ si la page j pointe sur la page i. Dans ce cas, des colonnes similaires sont des pages miroirs ;
- les lignes sont des pages web ou des documents, les colonnes sont des mots. Des colonnes similaires indiquent des mots qui apparaissent souvent ensemble dans les mêmes pages ;
- les lignes sont des pages web ou des documents, les colonnes sont des phrases. Les colonnes similaires indiquent des pages miroir ou des plagiats.

2.6 Motifs rares

Les motifs rares représentent les motifs qui apparaissent rarement dans un ensemble de données tel que les symptômes non usuels ou les effets indésirables exceptionnels qui peuvent se déclarer chez un patient pour une pathologie ou un traitement donné.

De même que pour les motifs fréquents, certains phénomènes rares dans les bases de données peuvent également véhiculer des connaissances.

La découverte des motifs rares peut se révéler très intéressante, en particulier en médecine et en biologie. Prenons d'abord un exemple simulé d'une base de données médicale où nous nous intéressons à l'identification de la cause des maladies cardio-vasculaires (MCV). Une règle d'association fréquente telle que "niveau élevé de cholestérol → MCV" peut valider l'hypothèse que les individus qui ont un fort taux de cholestérol ont un risque élevé de MCV. A l'opposé, si notre base de données contient un grand nombre de végétariens, une règle d'association rare "végétarien → MCV" peut valider l'hypothèse que les végétariens ont un risque faible de contracter une MCV. Dans ce cas, les motifs végétarien et MCV sont tous deux fréquents, mais le motif végétarien, MCV est rare. Un autre exemple est en rapport avec la pharmacovigilance, qui est une partie de la pharmacologie dédiée à la détection et l'étude des effets indésirables des médicaments. L'utilisation de l'extraction des motifs rares dans une base de données des effets indésirables des médicaments pourrait contribuer à un suivi plus efficace des effets indésirables graves et ensuite à prévenir les accidents fatals qui aboutissent au retrait de certains médicaments.

2.6.1 Définitions

Un motif est dit rare ou infréquent si son support est inférieur ou égal à un support maximum (noté max_supp).

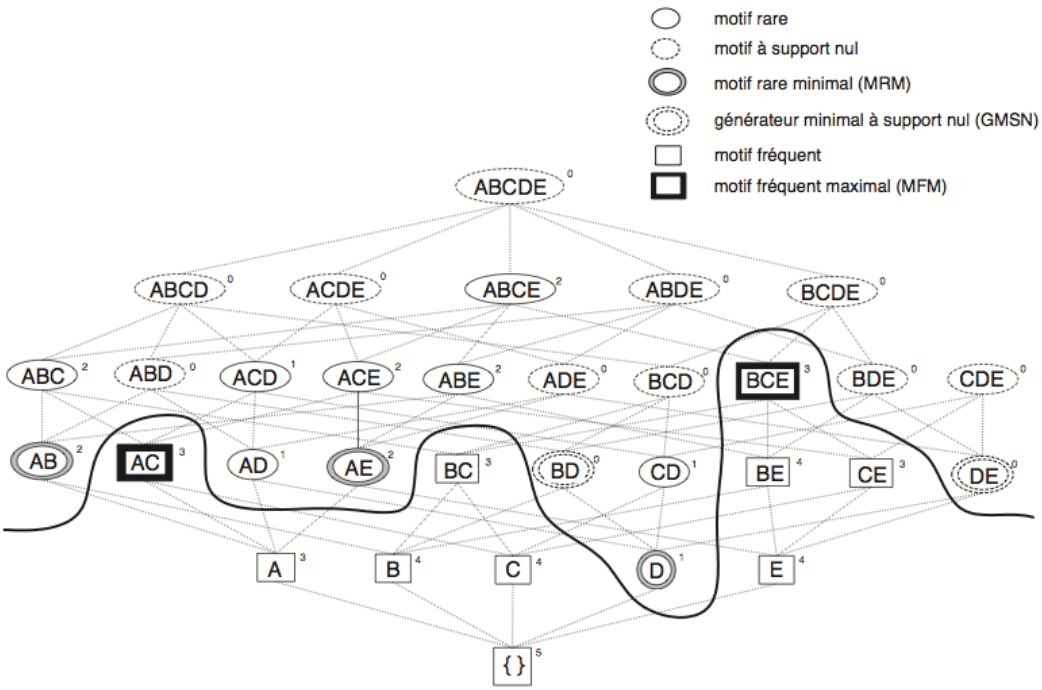
Généralement, on considère que $max\_supp = min\_supp - 1$, c'est-à-dire qu'un motif est rare s'il n'est pas fréquent. Cela implique l'existence d'une seule frontière entre motifs rares et fréquents.

2.6.2 Recherche des motifs rares

Une première approche pour la recherche des motifs rares est celle de treillis. Prenons l'exemple de la base suivante :

	A	B	C	D	E
1	X		X	X	
2		X	X		X
3	X	X	X		X
4	X				X
5	X	X	X		X

En fixant min_supp à 3 ($max\_supp = 2$), On obtient le treillis suivant :



Les motifs peuvent être séparés en deux ensembles formant une partition : les motifs rares et les motifs fréquents. Une frontière peut être dessinée entre ces deux ensembles. L'ensemble des motifs rares et l'ensemble des motifs fréquents ont tous deux un sous-ensemble minimal générateur. Dans le cas des motifs fréquents, ce sous-ensemble est appelé ensemble des motifs fréquents maximaux (MFM).

Ces motifs sont dits maximaux, parce qu'ils n'ont pas de sur-motifs fréquents. Du point de vue du nombre de ces motifs ils sont minimaux, c'est-à-dire qu'ils forment un ensemble générateur minimal à partir duquel tous les motifs fréquents peuvent être retrouvés.

Les motifs rares minimaux (MRM) sont définies en tant que complémentaires des MFMs. Un motif est un MRM s'il est rare (et ainsi tous ses sur-motifs sont rares) et si tous ses sous-motifs ne sont pas rares. Ces motifs forment un ensemble générateur minimal à partir duquel tous les motifs rares peuvent être retrouvés.

Ces motifs forment un ensemble générateur minimal à partir duquel on peut retrouver les motifs rares. Nous devons d'abord générer tous les sur-motifs possibles des motifs rares minimaux, puis calculer le support des motifs rares grâce à un passage sur la base de données.

2.6.3 Apriori-Rare

De manière surprenante, les motifs rares minimaux peuvent être trouvés simplement à l'aide de l'algorithme bien connu Apriori. Il est conçu pour trouver les motifs fréquents,

mais, puisque nous sommes dans le cas où non fréquent signifie rare, cela a pour "effet collatéral" d'explorer également les motifs rares minimaux. Quand Apriori trouve un motif rare, il ne générera plus tard aucun de ses sur-motifs car ils sont de manière sûre rares. Puisque Apriori explore le treillis des motifs niveau par niveau du bas vers le haut, il comptera le support des motifs rares minimaux. Ces motifs seront élagués et plus tard l'algorithme peut remarquer qu'un candidat a un sous-motif rare. En fait Apriori vérifie si tous les (k-1)-sous-motifs d'un k-candidat sont fréquents. Si l'un d'entre eux n'est pas fréquent, alors le candidat est rare. Autrement dit, cela signifie que le candidat a un sous-motif rare minimal. Grâce à cette technique d'élagage, Apriori peut réduire significativement l'espace de recherche dans le treillis des motifs. Une légère modification d'Apriori suffit pour conserver les MRM. Si le support d'un candidat est inférieur au support minimum, alors à la place de l'effacer nous l'enregistrons dans l'ensemble des motifs rares minimaux

Tous les motifs rares sont retrouvés à partir des motifs rares minimaux. Pour cela nous avons besoin de générer tous les sur-motifs possibles des MRM.

Chapitre 3

Classification

3.1 Concepts de base

3.1.1 Définition

La classification est une tâche très importante dans le data mining, et qui consomme beaucoup de recherches pour son optimisation. La classification supervisée est l'une des techniques les plus utilisées dans l'analyse des bases de données. Elle permet d'apprendre des modèles de décision qui permettent de prédire le comportement des exemples futurs.

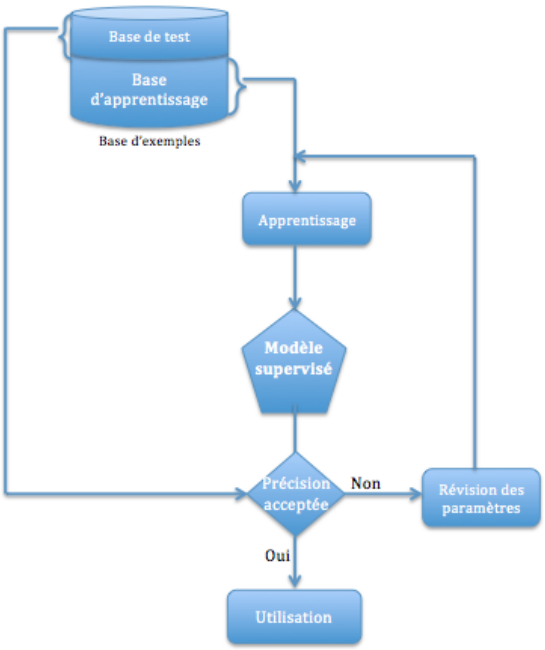
La classification supervisée consiste à inférer à partir d'un échantillon d'exemples classés une procédure de classification. Un système d'apprentissage effectue la recherche d'une telle procédure selon un modèle. Les systèmes d'apprentissage peuvent être basés sur des hypothèses probabilistes (classifieur naïf de Bayes, méthodes paramétriques) ; sur des notions de proximité (plus proches voisins) ; sur des recherches dans des espaces d'hypothèses (arbres de décision, réseaux de neurones).

3.1.2 Organisation

La classification est un processus à deux étapes : une étape d'apprentissage (entrainement) et une étape de classification (utilisation).

Dans l'étape d'apprentissage, un classiffieur (une fonction, un ensemble de règles, ...) est construit en analysant (ou en apprenant de) une base de données d'exemples d'entrainement avec leurs classes respectives. Un exemple $X = (x_1, x_2, .., x_m)$ est représenté par un vecteur d'attributs de dimension m. Chaque exemple est supposé appartenir à une classe prédéfinie représentée dans un attribut particulier de la base de donnée appelé attribut de classe. Puisque la classe de chaque exemple est donnée, cette étape est aussi connue par

l'apprentissage supervisé.



Dans l'étape de classification, le modèle construit dans la première étape est utilisé pour classer les nouvelles données. Mais avant de passer à l'utilisation, le modèle doit être testé pour s'assurer de sa capacité de généralisation sur les données non utilisées dans la phase d'entrainement. Le modèle obtenu peut être testé sur les données d'entrainement elles-mêmes, la précision (le taux de reconnaissance) est généralement élevée mais ne garantit pas automatiquement une bonne précision sur les nouvelles données. En effet, les données d'entrainement peuvent contenir des données bruitées ou erronées (outliers) qui ne représentent pas le cas général et qui tire le modèle vers leurs caractéristiques. Ce cas est appelée le sur-apprentissage ou en anglais *"over fitting"* et qui peut être évité en testant le modèle sur une base de données différente appelée base de test. La base de test est un ensemble d'exemples ayant les mêmes caractéristiques que ceux de la base d'entrainement et qui sont écartés au départ de l'entrainement pour effectuer les tests.
La méthode de prédiction utilisée dépend essentiellement du type d'information prédite c-à-d le type de l'attribut de classe. Si l'attribut est catégoriel ou symbolique (appartient à un ensemble fini), il s'agit de classification. par contre si cet attribut est continu (numérique) il s'agit d'un problème de régression.

Dans le cas de classification, on dispose d'un ensemble X de N données étiquetées représentant un sous ensemble de l'ensemble D de toutes les données possibles. Chaque

donnée x_i est caractérisée par P attributs et par sa classe $y_i \in Y$. Dans un problème de classification, la classe prend sa valeur parmi un ensemble fini. Le problème consiste alors, en s'appuyant sur l'ensemble d'exemples $X = \{(x_i, y_i)/i \in \{1, .., N\}\}$, à prédire la classe de toute nouvelle donnée $x \in D$. On parle de classification binaire quand le nombre de classes $|Y|$ est 2 ; il peut naturellement être quelconque. Dans tous les cas, il s'agit d'un attribut qualitatif pouvant prendre un nombre fini de valeurs. Dans l'absolu, une donnée peut appartenir à plusieurs classes : c'est alors un problème multi-classes. Ici, on considère que chaque donnée appartient à une et une seule classe. Souvent, on utilise le mot "étiquette" comme synonyme de "classe". Un exemple est donc une donnée dont on dispose de sa classe. On utilise donc un ensemble d'exemples classés pour prédire les classes des nouvelles données ; c'est une tâche d'"apprentissage à partir d'exemples", ou "apprentissage supervisé".

3.1.3 Evaluation du modèle

L'apprentissage supervisé utilise une partie des données pour calculer un modèle de décision qui sera généralisé sur l'ensemble du reste de l'espace. Il est très important d'avoir des mesures permettant de qualifier le comportement du modèle appris sur les données non utilisées lors de l'apprentissage. Ces métriques sont calculées soit sur les exemples d'entrainement eux mêmes ou sur des exemples réservés d'avance pour les tests.

La métrique intuitive utilisée est la précision du modèle appelée aussi le taux de reconnaissance. Elle représente le rapport entre le nombre de donnée correctement classées et le nombre total des données testées. L'équation suivante donne la formule utilisée.

$$P = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i))$$

avec :

$$L = \begin{cases} 1 & si \quad y_i = \hat{f}(x_i) \\ 0 & sinon \end{cases}$$

Généralement, la précision est donnée sous forme de pourcentage ce qui nécessite de multiplier la précision de l'équation précédente par 100.

3.1.3.1 Matrice de confusion

La mesure précédente donne le taux d'erreurs commises par le modèle appris ($100 - prcision$) mais ne donne aucune information sur la nature de ces erreurs. Dans la plus part des cas d'application, il est très important de connaître la nature des erreurs commises :

quelle classe est considérée comme quelle classe par le modèle ? Par exemple dans un modèle appris pour des objectifs médicaux, considérer un échantillon non cancéreux alors qu'il l'est, est beaucoup plus grave de considérer un échantillon cancéreux alors qu'il ne l'est pas. Dans le cas de classification binaire, le résultat de test d'un modèle peut être une possibilité parmi quatre :

$$\begin{cases} \hat{f}(x_i) = +1 \quad et \quad y_i = +1 \quad correcte \;\; positive \\ \hat{f}(x_i) = +1 \quad et \quad y_i = -1 \quad fausse \;\; positive \\ \hat{f}(x_i) = -1 \quad et \quad y_i = -1 \quad correcte \;\; négative \\ \hat{f}(x_i) = -1 \quad et \quad y_i = +1 \quad fausse \;\; négative \end{cases}$$

Si le modèle donne une classe positive pour un exemple d'une classe positive, on dit que c'est une classe correcte positive (CP). Si par contre l'exemple appartient à la classe négative on dit que c'est une classe fausse positive (FP). Si le modèle donne une classe négative pour un exemple d'une classe négative, le résultat est une classe correcte négative (CN), si, par contre, la classe de l'exemple est positive le résultat est qualifié de classe fausse négative (FN).

La matrice de confusion est une matrice qui rassemble en lignes les observations (y) et en colonnes les prédictions $\hat{f}(x)$. Les éléments de la matrice représentent le nombre d'exemples correspondants à chaque cas.

TABLE 3.1 – Matrice de confusion pour la classification binaire

Observations (y)	Prédictions ($\hat{f}$)	
	+1	-1
+1	CP	FN
-1	FP	CN

Un modèle sans erreurs aura ses résultats rassemblés sur la diagonale de sa matrice de confusion (CP et CN). Dans le cas multiclasses la matrice sera plus large avec les classes possibles au lieu des deux classes +1 et -1. La précision P du modèle peut être calculée à partir de la matrice de confusion comme suit :

$$P = \frac{CP + CN}{CP + FP + CN + FN}$$

Deux autre mesures sont utilisées dans la littérature : la sensitivité Sv et la spécificité Sp. La sensitivité représente le rapport entre les observations positives correctement prédites et le nombre des observations positives, et la spécificité représente le rapport entre les observations négatives correctement prédites et le nombre total des observations négatives.

$$\begin{cases} Sv = & \frac{CP}{CP+FN} \\ Sp = & \frac{CN}{CN+FP} \end{cases}$$

Une autre métrique calculée à base de la sensitivité et la spécificité est utilisée. C'est la moyenne harmonique.

$$Moyenne\ harmonique = \frac{2 \times Sv \times Sp}{Sv + Sp}$$

3.1.3.2 Évaluation

L'apprentissage d'un modèle de décision se fait à base de plusieurs paramètres, à savoir les exemples d'entrainement et leur nombre, les paramètres de la méthodes utilisée, ...etc. Le choix des valeurs de ces paramètres se fait à travers plusieurs essais et évaluation pour atteindre des performances satisfaisantes du modèle. Les paramètres optimaux pour un modèle donné sont les paramètres qui lui permettent de donner une précision de 100%. Cette situation serait idéale si l'ensemble des exemples représentait parfaitement l'ensemble de tous les exemples possibles. Le modèle appris peut donner une très grande précision face aux exemples d'entrainement, mais se comporte très mal avec les nouveaux exemples. Cela représente un phénomène très connu en apprentissage qui est le sur-apprentissage ou l'apprentissage par coeur. Le sur-apprentissage donne, généralement, des modèles à faible capacité de généralisation, et par conséquent la mesure de précision n'est pas suffisante pour qualifier les performances d'un modèle. Les méthodes d'évaluation permettent de tirer des conclusion sur le comportement d'un modèle face à tout l'espace d'exemples en limitant l'influence des exemples d'entrainement et du bruit qui peut y exister (erreurs d'étiquetage, erreurs d'acquisition, ...) et leur ordre sur le modèle appris.

3.1.3.2.1 Méthode HoldOut La méthode HoldOut suppose que les données d'entrainement sont tout l'espace d'exemples. Elle consiste à diviser l'ensemble des données en deux parties, la première partie est utilisée pour l'entrainement et la deuxième pour les tests. Le test du modèle appris sur la partie de test permet de donner une idée sur son comportement en dehors des exemples d'entrainement et éviter le phénomène de sur-apprentissage. Le modèle qui maximise la précision pour tout l'espace d'exemple est donc celui qui la maximise pour la partie de test du fait que cette partie représente la majorité de l'espace.

Une question importante qui se pose pour cette méthode est comment choisir les deux

parties puisque ce choix a une grande influence sur la qualité du modèle. Le pire est de mettre les exemples positifs dans une partie et les exemples négatifs dans l'autre. La méthode qui suit répond à cette question.

3.1.3.2.2 Validation croisée Pour minimiser l'influence du choix du partitionnement de l'ensemble des exemples, la validation croisée subdivise l'ensemble d'entrainement initial en k sous ensemble disjoints $D_1, D_2, .., D_k$ de même taille. L'entrainement et le test sont effectués k fois. A l'itération i le sous-ensemble D_i est réservé pour le test et le reste des exemples sont utilisés pour entrainer le modèle. La précision finale du modèle est égale à la moyenne des k précisions de test.
La méthode Leave-One-Out est un cas particulier de la validation croisée où $k = N$. Á chaque itération, le modèle est entrainé sur $N - 1$ exemples et testé sur l'exemple exclu de l'entrainement. On obtient à la fin N précisions, la précision du modèle est égale à leur moyenne.

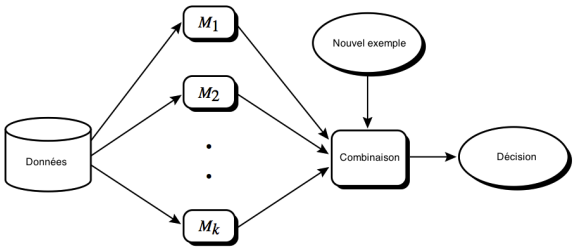
3.1.3.2.3 Le Bootstrap La méthode de Bootstrap, appelée aussi échantillonnage par remplacement, entraine le modèle sur un ensemble de N exemples choisis aléatoirement de l'ensemble des exemples, des exemples peuvent être choisis plus d'une fois et d'autre ne se seront pas choisis du tout. Les exemples non choisis pour l'entrainement sont utilisés pour le test. Cette opération peut être répétée plusieurs fois pour obtenir une précision moyenne du modèle.
Parmi les méthodes de Bootstrap les plus utilisées, la méthode Bootstrap ".632" qui tire son nom du fait que 63.2 % des exemples contribuent à l'entrainement et les restants (36.8%) contribuent aux tests. A chaque prélèvement, un exemple a une probabilité $1/N$ d'être choisi et $(1-1/N)$ de ne pas l'être, et puisqu'on répète le prélèvement N fois, chaque exemple aura une probabilité de $(1-1/N)^N$ de ne pas être choisi du tout dans un ensemble d'entrainement. Si N est grand cette probabilité approche de $e^{-1} = 0.368$. La méthode répète le processus k fois et la précision finale P est donnée par :

$$P = \sum_{i=1}^{k} (0.632 \times Pi_{test} + 0.368 \times Pi_{entr})$$

Où Pi_{test} est la précision du modèle entrainé sur les exemples choisis dans l'itération i, appliqué sur les exemples de test dans la même itération. Pi_{entr} est la précision du même modèle appliqué sur les données d'entrainement.

3.2 Combinaison de modèles

Pour augmenter la précision des modèles obtenus, certaines méthodes combinent plusieurs modèles pour obtenir les décisions.



Deux méthodes sont particulièrement utilisées : Bagging and Boousing.

3.2.1 Bagging

Cette méthode se base sur le Bootstrap. Elle subdivise l'ensemble D d'exemples en n sous-ensembles. À partir de chaque sous-ensemble D_i, on apprend un modèle M_i en utilisant la méthode Bootstrap. L'ensemble de ces modèles forme un modèle composé M_*. Pour classiffier un nouvel exemple, il est exposé à chaque modèle M_i pour obtenir une classe c_{M_i}. Chaque décision est considérée comme un vote. La classe de décision est prise comme la classe la plus votée.

3.2.2 Boosting

Dans la méthode boosting, on associe des poids aux exemples. Une série de k modèles est itérativement apprise. Après qu'un modèle M_i est construit, les poids des exemples sont mis à jour de telle sorte à attirer l'attention du modèle M_{i+1} aux exemples mal-classées par le modèle M_i. Le Modèle final M_* combine les voltes des k modèles pondérés par leur précisions.

3.3 Classification par analyse des règles d'association

Dans la classification associative (par règles d'association), les règles d'association sont générées et analysées pour les utiliser en classification. L'idée est de rechercher les règles solides contenant dans leur partie droite l'attribut classe, c-à-d de la forme :

$Attribut_1 = v_{att1} \wedge Attribut_2 = v_{att2} \wedge ... \wedge Attribut_n = v_{attn} \Rightarrow Classe = v_{classe}$

Plusieurs études ont montré que cette technique est plus précise que certaines méthodes traditionnelles tel que les arbres de décision.

3.3.1 Algorithme ZeroR

ZeroR est le classifieur le plus simple, il se base sur la classe et ignore tous les attribut. Il prédit tout simplement la classe majoritaire. Il n'a aucune capacité de prédiction mais représente une base de comparaison pour les autres méthodes.

Exemple : "Play Golf = Yes" est le modèle ZeroR pour la base suivante avec une précision de 64%.

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

3.3.2 Algorithme OneR

OneR (One Rule) est un algorithme simple et précis de classification. Il génère une règle pour chaque attribut dans les données puis sélectionne la règle d'erreur minimale comme la seule règle de décision. Pour créer une règle pour un attribut, on construit une table de fréquence pour chaque attribut par rapport à la classe. L'expérience a montré que cet algorithme produit des règles un peu moins précises que plusieurs algorithmes classiques de classification. Cependant, les règles produites sont facilement interprétables par les êtres humains.

Algorithme 2 OneR

pour chaque attribut **faire**

pour chaque valeur de cet attribut **faire** Créer une règle comme suit :

Calculer le nombre d'apparition de chaque valeur de la classe ;

Trouver la classe la plus fréquente ;

Créer une règle assignant cette classe à cette valeur de l'attribut

fin pour

fin pour

Calculer l'erreur totale de chaque règle pour chaque attribut

Choisir l'attribut de l'erreur minimale

Exemple : trouver l'attribut d'erreur minimale en utilisant l'algorithme OneR de la table weather.

Frequences

★		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3

IF Outlook = Sunny THEN PlayGolf = Yes
IF Outlook = Overcast THEN PlayGolf = Yes
IF Outlook = Rainy THEN PlayGolf = No

3.3.3 Algorithme CBA

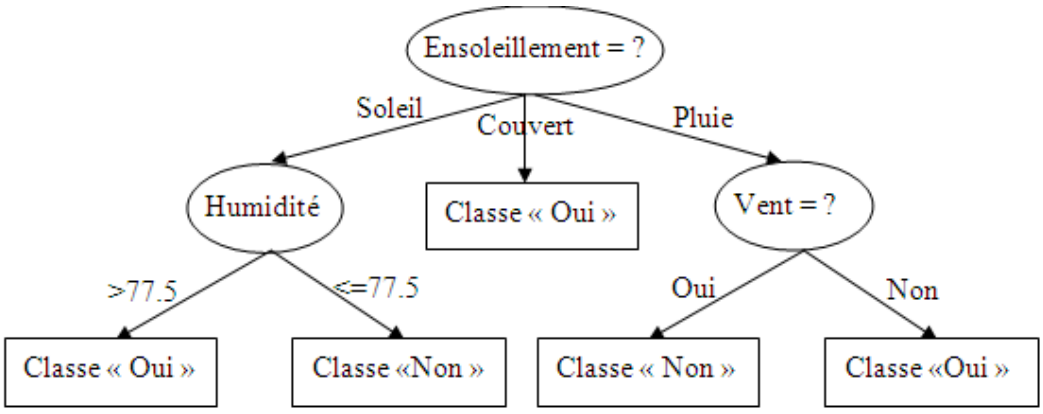
L'un des premiers algorithmes de classification associative est l'algorithme CBA (Classification-Based Association). Il utilise l'algorithme Apriori pour générer les règles d'association puis utilise une heuristique pour construire le classiffieur. Les règles sont ordonnées selon leurs supports et confidences. Si plusieurs règles ont la même partie gauche, la règle de la confidence la plus élevée est utilisée dans le classifieur. Pour classer un nouveau tuplet, la première règle le satisfaisant est utilisée. Le classifieur contient aussi une règle par défaut pour classer les tuplet dont une règles satisfaisante n'existe pas.

3.4 Arbres de décision

Les arbres de décision représentent une méthode très efficace d'apprentissage supervisé. Il s'agit de partitionner un ensemble de données en des groupes les plus homogènes possible du point de vue de la variable à prédire. On prend en entrée un ensemble de données classées, et on fournit en sortie un arbre qui ressemble beaucoup à un diagramme d'orientation où chaque nœud final (feuille) représente une décision (une classe) et chaque nœud non final (interne) représente un test. Chaque feuille représente la décision d'appartenance à une classe des données vérifiant tous les tests du chemin menant de la racine à cette feuille. L'exemple suivant montre un ensemble de données avec quatre attributs : Ensoleillement, Température, Humidité, Vent et l'attribut à prédire Jouer.

N°	Ensoleillement	Température	Humidité	Vent	Jouer
1	Soleil	75	70	Oui	Oui
2	Soleil	80	90	Oui	Non
3	Soleil	85	85	Non	Non
4	Soleil	72	95	Non	Non
5	Soleil	69	70	Non	Oui
6	Couvert	72	90	Oui	Oui
7	Couvert	83	78	Non	Oui
8	Couvert	64	65	Oui	Oui
9	Couvert	81	75	Non	Oui
10	Pluie	71	80	Oui	Non
11	Pluie	65	70	Oui	Non
12	Pluie	75	80	Non	Oui
13	Pluie	68	80	Non	Oui
14	Pluie	70	96	Non	Oui

L'arbre appris à partir de cet ensemble de donnée est le suivant :



En effet, toutes les données ayant l'attribut Ensoleillement="Soleil" et l'attribut Humidité>77.5 appartiennent à la classe 1 ("oui"). Toute nouvelle donnée peut être classée en testant ses valeurs d'attributs l'un après l'autre en commençant de la racine jusqu'à atteindre une feuille c'est-à-dire une décision. Pour construire un tel arbre, plusieurs algorithmes existent : ID3, CART, C4.5,...etc. On commence généralement par le choix d'un attribut puis le choix d'un nombre de critères pour son nœud. On crée pour chaque critère un nœud concernant les données vérifiant ce critère. L'algorithme continue d'une façon récursive jusqu'à obtenir des nœuds concernant les données de chaque même classe.

Algorithme 3 CONSTRUIRE-ARBRE(D : ensemble de données)

1: Créer nœud N
2: **Si** tous les exemples de D sont de la même classe C alors
3: Retourner N comme une feuille étiquetée par C ;
4: **Si** la liste des attributs est vide alors
5: Retourner N Comme une feuille étiquetée de la classe de la majorité dans D ;
6: Sélectionner l'attribut A du meilleur Gain dans D ;
7: Etiqueter N par l'attribut sélectionné ;
8: Liste d'attributs ← Liste d'attributs - A ;
9: **Pour** chaque valeur V_i de A
10: Soit D_i l'ensemble d'exemples de D ayant la valeur de $A = V_i$;
11: Attacher à N le sous arbre généré par l'ensemble Di et la liste d'attributs
12: **FinPour ;**
13: **Fin ;**

En réalité ce n'est pas si simple, plusieurs problèmes doivent être résolus :

- Comment choisir l'attribut qui sépare le mieux l'ensemble de données ? On parle souvent de la variable de segmentation.
- Comment choisir les critères de séparation d'un ensemble selon l'attribut choisi, et comment ces critères varient selon que l'attribut soit numérique ou symbolique ?
- Quel est le nombre optimal du nombre de critères qui minimise la taille de l'arbre et maximise la précision ?
- Quels sont les critères d'arrêt de ce partitionnement, sachant que souvent l'arbre et d'une taille gigantesque ?

3.4.1 Choix de la variable de segmentation

Il s'agit de choisir parmi les attributs des données, celui qui les sépare le mieux du point de vue de leurs classes déjà connues. Pour choisir le meilleur attribut, on calcule pour chacun une valeur appelée "Gain" qui dépend des différentes valeurs prises par cet attribut. Cette mesure est basée sur les recherches en théorie d'informations menées par C.Shannon.

Par exemple, l'algorithme ID3 utilise le concept d'entropie introduite initialement par Shannon en 1948.

Soit un ensemble X d'exemples dont une proportion p_+ sont positifs et une proportion

p_- sont négatifs. (Bien entendu, $p_+ + p_- = 1$) L'entropie de X est :

$$H(X) = \text{-}p_+ log_2(p_+) \text{-} p_- log_2(p_-)$$

Biensur

$$0 \leq H(X) \leq 1$$

Si $p_+ = 0$ ou $p_- = 0$, alors $H(X) = 0$. Ainsi, si tous exemples sont soit tous positifs, soit tous négatifs, l'entropie de la population est nulle. Si $p_+ = p_- = 0.5$, alors $H(X) = 1$. Ainsi, s'il y a autant de positifs que de négatifs, l'entropie est maximale.

$$Gain(X, a_j) = H(X) \text{-} \sum_{v \in valeurs(a_j)} \frac{|X_{a_j=v}|}{|X|} H(X_{a_j=v})$$

Où $X_{a_j=v}$, est l'ensemble des exemples dont l'attribut considéré a_j prend la valeur v, et la notation $|X|$ indique le cardinal de l'ensemble X.

Exemple

Le Gain du champs "Vent" de la table précédente est calculé comme suit :

$$\begin{aligned}
Gain(X, vent) &= H(X) - \tfrac{9}{14} H(X_{a=oui}) - \tfrac{5}{14} H(X_{a=non}) \\
On\ a : & \\
H(X) &= -\tfrac{5}{14} ln_2 \tfrac{5}{14} - \tfrac{9}{14} ln_2 \tfrac{9}{14} = 0.940 \\
H(X_{a=non}) &= -(\tfrac{6}{8} ln_2 \tfrac{6}{8} + \tfrac{2}{8} ln_2 \tfrac{2}{8}) = 0.811 \\
Et & \\
H(X_{a=oui}) &= -(\tfrac{3}{6} ln_2 \tfrac{3}{6} + \tfrac{3}{6} ln_2 \tfrac{3}{6}) = 1.0 \\
D'où : & \\
Gain(X, vent) &= 0.940 - \tfrac{9}{14} * 0.811 - \tfrac{5}{14} * 1.0 \\
&= 0.048
\end{aligned}$$

Le principe de l'algorithme ID3 pour déterminer la variable de segmentation est de prendre la variable du gain d'information maximum.

3.4.2 Choix de la bonne taille de l'arbre

Une fois l'arbre de décision construit, il peut contenir plusieurs anomalies qui peuvent être dues au bruit ou aux valeurs extrêmes, et qui peuvent conduire au problème de sur-apprentissage (overfitting). Ce problème est la déduction d'informations plus que supporte l'ensemble de données d'apprentissage. L'arbre peut être aussi d'une taille très importante qui peut épuiser les ressources de calcul et de stockage. Pour surmonter ce problème, on effectue des opérations d'élagage qui consistent à éliminer de l'arbre les branches les moins

significatives (qui déduisent d'un nombre réduit d'enregistrements ou de ceux qui appartiennent à diverses classes). L'élagage peut être effectué avant ou après l'apprentissage, on parle souvent de pré et post-élagage :

- Pré-élagage : effectué lors de la construction de l'arbre, lorsqu'on calcule les caractéristiques statistiques d'une partie des données tel que le gain, on peut décider de l'importance ou non de sa subdivision, et ainsi on coupe complètement des branches qui peuvent être générée.
- Post-élagage : effectué après la construction de l'arbre en coupant des sous arbres entiers et en les remplaçant par des feuilles représentant la classe la plus fréquente dans l'ensemble des données de cet arbre. On commence de la racine et on descend, pour chaque nœud interne (non feuille), on mesure sa complexité avant et après sa coupure (son remplacement par une feuille), si la différence est peu importante, on coupe le sous arbre et on le remplace par une feuille.

3.4.3 Algorithmes de construction d'arbres de décision

3.4.3.1 Algorithme ID3

ID3 construit l'arbre de décision récursivement. A chaque étape de la récursion, il calcule parmi les attributs restant pour la branche en cours, celui qui maximisera le gain d'information. C'est-à-dire l'attribut qui permettra le plus facilement de classer les exemples à ce niveau de cette branche de l'arbre. Le calcul ce fait à base de l'entropie de Shanon déjà présentée. L'algorithme suppose que tous les attributs sont catégoriels ; si des attributs sont numériques, ils doivent être descritisés pour pouvoir l'appliquer. ID3 utilise l'algorithmes *Construire_ arbre* précédent.

3.4.3.2 Algorithme C4.5 (J48)

C'est une amélioration de l'algorithme ID3, il prend en compte les attributs numérique ainsi que les valeurs manquantes. L'algorithme utilise la fonction du gain d'entropie combiné avec une fonction *SplitInfo* pour évaluer les attributs à chaque itération.

3.4.3.2.1 Attributs discrets Pour les attributs discrets possédant un grand nombre de valeurs, nous avons vu que la fonction GainRatio permettait d'éviter de privilégier ces attributs. Il existe, de plus, une option de C4.5 qui permet le regroupement des valeurs. Par exemple, si on dispose d'un attribut A prenant les valeurs a, b, c et d, en standard le

test considéré serait 4-aire. Si on active l'option regroupement, seront également considéré des tests de la forme : le test binaire A∈{a,b} et A∈{c,d} ; le test ternaire A=a , A=c et A∈{b,d} ; ...

3.4.3.2.2 Attributs continus Pour les attributs continus, la discrétisation peut être laissée à un expert du domaine d'application. Par exemple, en médecine, l'expérience du domaine peut avoir permis la mise en évidence l'existence de valeurs seuil pour un attribut correspond à une mesure médicale. Sinon, l'algorithme gère les attributs continus de la façon suivante : les exemples sont triés dans l'ordre croissant pour l'attribut continu A considéré, on considère alors tous les tests de la forme A>a_i + $a_{i+1}/2$ où a_i et a_{i+1} sont deux valeurs consécutives de l'attribut A. Par exemple, supposons que A prenne les valeurs 1 ; 3 ; 6 ; 10 ; 12, alors on considère les tests A>2 ; A>4,5 ; A>8 et A>11, ces tests participent alors à la compétition dans la recherche du test apportant le meilleur gain (fonction Gain ou GainRatio, selon l'option choisie).

3.4.3.2.3 Attributs à valeurs manquantes Dans de nombreux problèmes concrets, il existe certains attributs dont les valeurs ne sont pas renseignées. Par exemple, si on dispose du descriptif de patients, il est très probable que toutes les mesures ne soient pas disponibles car elles n'ont pas pu être faites pour tous les patients. Pour classifier un exemple possédant des valeurs manquantes à l'aide d'arbres de décision, on procède comme dans le cas standard, lorsque l'on rencontre un test et que la valeur de l'attribut est manquante, on considère la branche majoritaire. Pour la phase d'apprentissage, on suppose que la valeur de cet attribut suit la distribution des valeurs connues.

3.4.3.3 Algorithme CART

L'lgorithme CART dont l'acronyme signifie "Classification And Regression Trees", construit un arbre de décision d'une manière analogue à l'algorithme ID3. Contrairement à ce dernier, l'arbre de décision généré par CART est binaire et le critère de segmentation est l'indice de Gini.

À un attribut binaire correspond un test binaire. À un attribut qualitatif ayant n modalités, on peut associer autant de tests qu'il y a de partitions en deux classes, soit 2n-1 tests binaires possibles. Enfin, dans le cas d'attributs continus, il y a une infinité de tests envisageables. Dans ce cas, on découpe l'ensemble des valeurs possibles en segments, ce découpage peut être fait par un expert ou fait de façon automatique.

3.4.3.4 Forêts aléatoires

Les forêts aléatoires ont été inventées par Breiman en 2001 ([Bre01]). Elles sont en général plus efficaces que les simples arbres de décision mais possède l'inconvénient d'être plus difficilement interprétables. Leur construction se base sur le bootstrap (ou le bagging). On subdivise l'ensemble de données en plusieurs parties par le bootstrap puis on apprend un arbre de décision à partir de chaque partie. Un nouvel exemple est testé par tous les arbres construits et sa classe est la classe majoritaire.

3.5 Réseaux bayésiens

Les techniques se basant sur les lois statistiques sont les premières qui ont été utilisées pour l'analyse de données. Elles consistent à prendre un sous ensemble d'une population et essayer d'arriver à des conclusions concernant toute la population. Ce sont des méthodes qui reposent sur la théorie de Bayes représentant une référence théorique pour les approches statistiques de résolution des problèmes de classification. Le principe de cette théorie est le suivant : Soit X un échantillon de données dont la classe est inconnue et qu'on veut la déterminer, et soit H une hypothèse (X appartient à la classe C par exemple). On cherche à déterminer $P(H/X)$ la probabilité de vérification de H après l'observation de X. $P(H/X)$ est la probabilité postérieure c'est-à-dire après la connaissance de X tandis que $P(H)$ est la probabilité à priori représentant la probabilité de vérification de H pour n'importe quel exemple de données. Le théorème de Bayes propose une méthode de calcul de $P(H/X)$ en utilisant les probabilités $P(H)$, $P(X)$ et $P(X/H)$:

$$P(H/X) = [P(X/H).P(H)]\,/P(X)$$

$P(H/X)$ est donc la probabilité d'appartenance de X à la classe C, $P(H)$ la probabilité d'apparition de la classe C dans la population et qui peut être calculée comme le rapport entre le nombre d'échantillons appartenant à la classe C et le nombre total d'échantillons. $P(X/H)$ peut être considérée comme la probabilité d'apparence de chaque valeur des attributs de X dans les attributs des échantillons appartenant à la classe C :

$$P(X/H) = \prod P(a_i = v_i/H)$$

Où a_i est le $i^{ème}$ attribut de X et v_i sa valeur. Cette astuce de calcul de $P(X/H)$ est basée sur la supposition d'indépendance entre les attributs. Malgré que cette supposition soit rarement vérifiée, sa considération facilite le calcul et donne une idée approximative sur

la probabilité. Finalement $P(X)$ est constante pour toute la population et indépendante des classes. Il ne reste donc que considérer la classe de X, celle maximisant le produit $P(X/H) \bullet P(H)$. Cette application est l'application la plus simple de la théorie de Bayes, elle s'appelle la classification naïve de Bayes.

En pratique, on peut vouloir trouver la classe d'un enregistrement dont la valeur d'un attribut n'existe pas dans la table. Dans ce cas, une méthode dite "Estimateur de Laplace" est utilisée : on ajoute 1 à tous les numérateurs des probabilités et on ajoute le nombre de valeurs distinctes de cet attribut au dénominateur. Par exemple au lieu d'avoir les probabilités $\frac{2}{9}$, $\frac{4}{9}$ et $\frac{3}{9}$, on utilise les probabilités $\frac{3}{12}$, $\frac{5}{12}$ et $\frac{4}{12}$ si l'attribut n'a que 3 valeurs distinctes. Comme ça on minimise la probabilité sans l'annuler et par conséquent annuler toute la probabilité.
Un autre problème que l'algorithme ne prend pas en compte, bien comme il faut, est celui des valeurs numériques continues, puisqu'il se base uniquement sur les égalités des valeurs. En effet, on ne peut pas dire que la probabilité qu'une variable continue soit égale à 12.36 est égale à 0 par exemple, seulement car la valeur 12.36 n'appartient pas aux valeurs de cet attribut. Pour surmonter le problème, on suppose que la distribution des valeurs de l'attribut est normale, et on calcule sa moyenne et sont écart type et la probabilité peut être calculée selon la loi normale :

$$P(X = x) = \frac{1}{\sqrt{2\pi\rho^2}} e^{-\frac{(x-\overline{x})^2}{2\rho^2}} dx$$

La méthode naïve de Bayes est applicable uniquement en cas de vérification de l'indépendance entre les attributs, ce qui peut être contrôlé par la matrice de corrélation et ses valeurs propres. Aussi, les valeurs des attributs numériques doivent avoir une distribution normale. Cette méthode reste une méthode simple et moins coûteuse en temps de calcul. Elle est aussi incrémentale c'est-à-dire que l'arrivée d'une nouvelle information (classe d'un nouvel enregistrement) ne nécessite pas de refaire tous les calculs pour la prendre en considération. Les connaissances apprises peuvent être renforcées sans avoir besoin de refaire tous les calculs.

Les réseaux Bayésiens (ou réseaux de croyance) prennent en considération les dépendances éventuelles entre les attributs contrairement à la technique précédente. Un réseau Bayésien est représenté sous forme d'un graphe orienté acyclique, où les nœuds représentent les attributs et les arcs représentent les liaisons entre ces attributs (des probabilités conditionnelles). Deux attributs sont reliés par un arc si l'un cause ou influe sur l'autre : le pré-

décesseur est la cause et le successeur est l'effet. Prenons l'exemple suivant : Un médecin reçoit un patient qui souffre d'un problème de respiration (symptôme) appelé "dyspnoea", et qui a peur d'avoir un cancer de poumon. Le médecin sait que d'autres causes sont possibles tel que la tuberculose et les bronchites. Il sait aussi que d'autres informations peuvent augmenter la probabilité du cancer tel que si le patient est fumeur ou non, et la pollution de l'air où il vie. Mais une image rayon X positive confirmera le cancer ou la tuberculose. Le tableau suivant résume les attributs qu'on a avec leurs valeurs possibles.

TABLE 3.2 – Attributs utilisés pour un exemple d'un réseau Bayésien

Attribut	type	valeur
Pollution	Binaire	{Basse, Haute}
Fumeur	Booléen	{V, F}
Cancer	Booléen	{V, F}
Dyspnoea	Booléen	{V, F}
X-Ray	Binaire	{Positif, Négatif}

Le réseau Bayésien représentant de telles connaissances est le suivant :

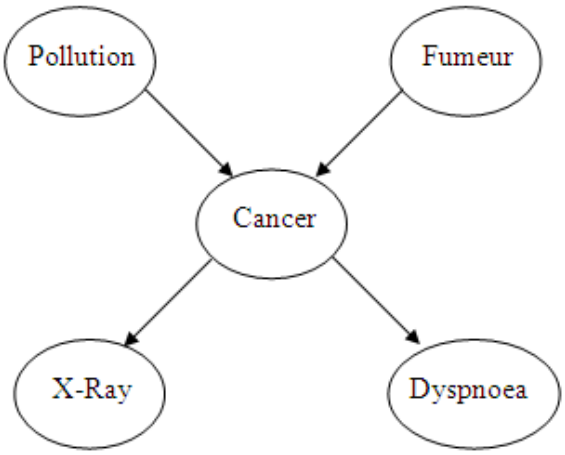


FIGURE 3.1 – Exemple d'un réseau Bayésien

En effet, la pollution et fumer causent le cancer, et le cancer cause des rayons X positifs et le symptôme Dyspnoea. Les qualités des relations entre ces nœuds sont représentées dans des tables appelées CPT (Conditional Probability Table) en fonction des valeurs possibles des attributs. Pour chaque valeur possible des pères, on établit une table représentant les probabilités d'avoir les différentes valeurs possibles du fils : P(Cancer=V|Pollution, Fumeur).

TABLE 3.3 – Table des probabilités conditionnelles pour un réseau Bayésien

Polution	Fumeur	P(Cancer=V|Pollution, Fumeur)
Haute	V	0.05
Haute	F	0.02
Basse	V	0.03
Basse	F	0.001

Le réseau Bayésien peut être construit à partir de la base de données d'apprentissage en calculant la corrélation entre les attributs. On commence par ajouter au réseau les nœuds (attributs) indépendants et à chaque fois, on ajoute des arcs à partir des nœuds existants dans le réseau desquels dépend le nœud ajouté. Les CPTs peuvent être aussi calculées facilement à partir de la base de données en se basant sur la fréquence d'apparition des valeurs.

Une fois le réseau Bayésien établi avec les CPTs, il peut être utilisé pour raisonner dans deux sens :

- Au sens des arcs : appelé "Prédiction" où on possède des causes et on cherche les probabilités des différents effets possibles, par exemple, on connaît qu'un patient est fumeur et on cherche la probabilité d'avoir un cancer, on multiplie simplement les probabilités du chemin entre la cause et l'effet final.
- Au sens contraire des arcs : appelé "Diagnostic" où on connaît des effets et on cherche les probabilités de certaines causes, par exemple, on connaît qu'un patient a un cancer, et on cherche la probabilité qu'il soit un fumeur. Dans ce cas, on multiplie aussi les probabilités du chemin inversé de l'effet à la cause.

Les réseaux Bayesiens sont beaucoup plus précis que d'autres techniques d'apprentissage, puisqu'ils, d'un côté, prennent en considération les dépendances entre les attributs, et d'un autre côté, peuvent intégrer des connaissances humaines au préalable. On peut par exemple introduire directement la topologie du réseau et le faire entraîner pour construire les CPTs. Ils sont aussi incrémentaux puisque les croyances peuvent être modifiées à chaque arrivée d'une nouvelle information et cela par propagation directe sur le réseau. Malheureusement, ces réseaux sont très coûteux en temps de calcul pour l'apprentissage surtout pour le calcul des CPts puisqu'il faut calculer une probabilité pour chaque valeur possible d'un fils pour chaque valeur possible de chacun de ses pères. L'espace nécessaire pour stocker les CPTs est aussi exhaustif.

La plupart des travaux récents sur les réseaux Bayésiens visent à optimiser la tâche complexe d'apprentissage en optimisant le temps de calcul tout en gardant la précision. Dans des travaux récents, on essaye d'hybrider les réseaux Bayésiens avec les machines à vecteurs supports(SVM) pour estimer les paramètres d'apprentissage. Une combinaison du raisonnement Bayesiaen avec les méthodes à noyaux a permis selon certaines recherches d'utiliser plusieurs hyperplans, pour séparer les données, ensuite utiliser ces hyperplans pour produire un seul critère de classification plus précis.

3.6 Réseaux de neurones

Les réseaux de neurones artificiels (RNA) sont inspirés de la méthode de travail du cerveau humain qui est totalement différente de celle d'un ordinateur. Le cerveau humain se base sur un système de traitement d'information parallèle et non linéaire, très compliqué, ce qui lui permet d'organiser ses composants pour traiter, d'une façon très performante et très rapide, des problèmes très compliqués tel que la reconnaissance des formes.

3.6.1 Physiologie du cerveau humain

La physiologie du cerveau montre que celui-ci est constitué de cellules (les neurones) interconnectées.

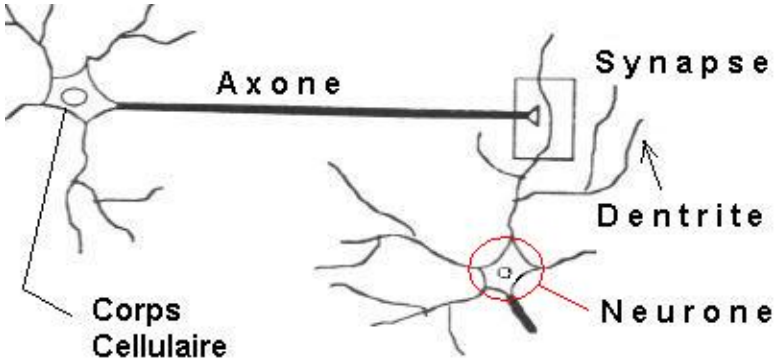


FIGURE 3.2 – Modèle d'un neurone artificiel

Les neurones reçoivent les signaux (impulsions électriques) par des extensions très ramifiées de leur corps cellulaire (les dendrites) et envoient l'information par de longs prolongements (les axones). Les impulsions électriques sont régénérées pendant le parcours le long de l'axone. La durée de chaque impulsion est de l'ordre d'1 ms et son amplitude d'environ 100 mV.

Les contacts entre deux neurones, de l'axone à une dendrite, se font par l'intermédiaire des synapses

Voici quelques informations à propos des neurones du cerveau humain :

- le cerveau contient environ 100 milliards de neurones.
- on ne dénombre que quelques dizaines de catégories distinctes de neurones.
- aucune catégorie de neurones n'est propre à l'homme.
- la vitesse de propagation des influx nerveux est de l'ordre de 100m/s, c'est à dire bien inférieure à la vitesse de transmission de l'information dans un circuit électronique.

– on compte de quelques centaines à plusieurs dizaines de milliers de contacts synaptiques par neurone. Le nombre total de connexions est estimé à environ 10^{15}.
– la connectique du cerveau ne peut pas être codée dans un "document biologique" tel l'ADN pour de simples raisons combinatoires. La structure du cerveau provient donc en partie des contacts avec l'environnement. L'apprentissage est donc indispensable à son développement.
– le nombre de neurones décroit après la naissance. Cependant, cette affirmation semble remise en question.
– les synapses entre des neurones qui ne sont pas simultanément actifs sont affaiblis puis éliminés.
– il semble que l'apprentissage se fasse par un double mécanisme : des connections sont établies de manière redondantes et aléatoires puis seules les connexions entre des neurones simultanément actifs sont conservés (phase de sélection) tandis que les autres sont éliminés. On parle de stabilisation sélective.

3.6.2 Structure d'un réseau de neurones artificiels

3.6.2.1 Un neurone seul

Un neurone est l'unité élémentaire de traitement d'un réseau de neurones. Il est connecté à des sources d'information en entrée (d'autres neurones par exemple) et renvoie une information en sortie.

Un neurone artificiel reçoit en entrée des entrées numériques x_i ($1 \leq i \leq N$ où N est le nombre de données qu va recevoir le neurone) valorisée chacune par un coefficient w_i.

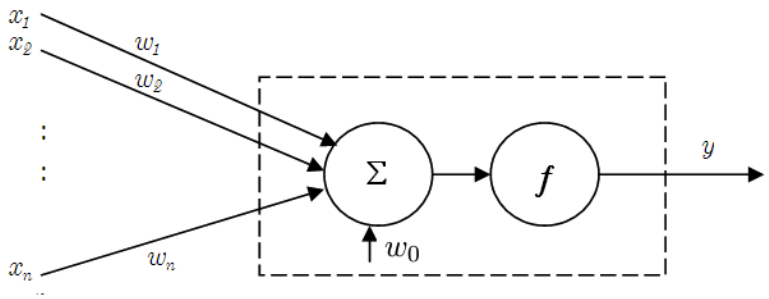
Le neurone artificiel (qui est une modélisation des neurones du cerveau) effectue alors une somme pondérée de ses entrées et lui ajoute un coefficient w_0 dit de biais supposé lié à une donnée $x_0 = -1$.

$$S = w_1 \times x_1 + \cdots + w_N \times x_N - w_0 = \sum_{i=1}^{N} w_i \times w_i - w_0$$

Cette donnée est passée à une fonction f dite d'activation qui représente un filtre permettant d'adapter la valeur de la somme précédente aux caractéristiques de la sortie désirée. C'est une fonction qui, généralement, doit renvoyer un réel proche de 1 quand les "bonnes" informations d'entrée sont données et un réel proche de 0 quand elles sont "mauvaises". La valeur de la fonction d'activation est la sortie y du neurone.

$$y = f(\sum_{i=1}^{N} w_i \times x_i - w_0)$$

La figure suivante résume la composition d'un neurone artificiel :



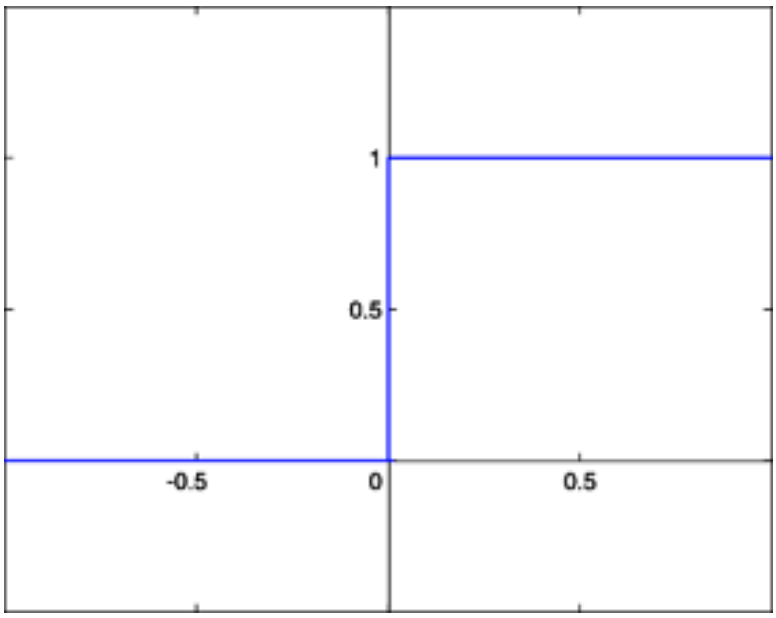
La figure montre qu'un neurone artificiel se constitue de trois éléments basiques :

- Un ensemble de connexions avec les différentes entrées x_i, pondérée chacune par un poids w_i,
- Un additionneur permettant de calculer une combinaison linéaire des entrées x_i pondérées par les coefficients w_i,
- Un biais w_0 qui permet de contrôler l'entrée de la fonction d'activation,
- Une fonction d'activation f permettant de délimiter la sortie y du neurone.

Il existe beaucoup de fonctions d'activations possibles, toutefois dans la pratique il y en a principalement deux qui sont utilisées :

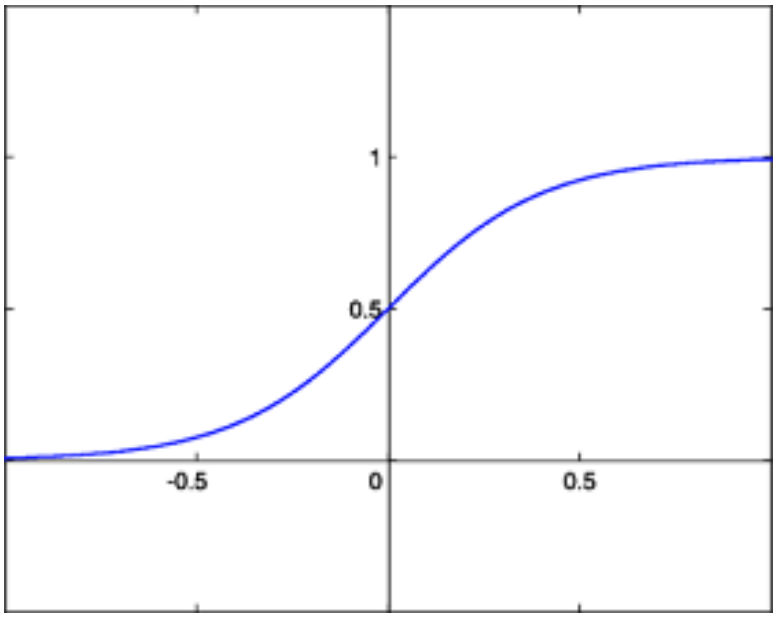
- **La fonction de Heaviside**

  La fonction de Heaviside est définie par : $\forall\ x \in \Re\ f(x) = 1\ six \geq 0,\ 0\ sinon$



– **La fonction sigmoïde**

La fonction de Heaviside est définie par : $\forall\ x \in \Re\ f(x) = \frac{1}{1+e^{-x}}$



3.6.2.2 Réseau de neurone

L'architecture d'un réseau de neurones artificiel est définie par la structure de ses neurones et leur connectivité. Elle est spécifiée par le nombre d'entrées, de sorties, de nœuds et la façon selon laquelle sont interconnectés et organisés les nœuds. Une fameuse architecture des réseaux de neurones est celle basée sur des couches où les nœuds de chaque couche n'ont aucune connexion entre eux. Cette architecture est utilisée dans presque 90 % des applications commerciales et industrielles.

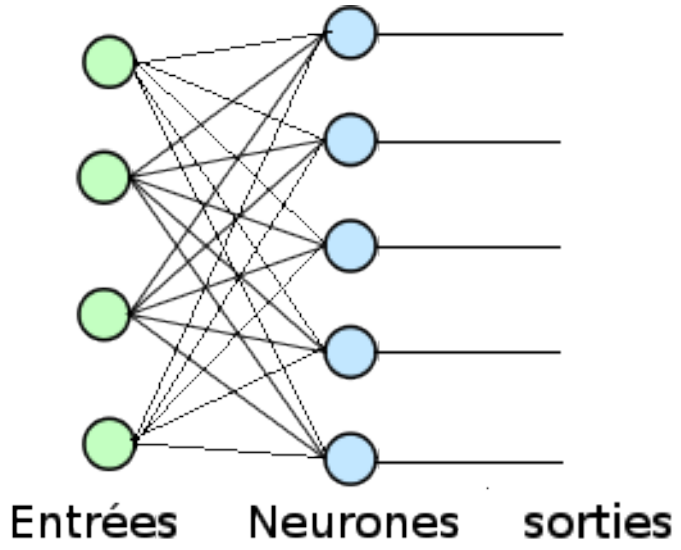
3.6.2.2.1 Réseau de neurone monocouche (Perceptron) Un réseau de neurones est une structure de réseau constituée d'un nombre de nœuds interconnectés par des liaisons directionnelles. Chaque nœud représente une unité de traitement et les liaisons représentent les relations causales entre les nœuds.

Le réseau le plus simple est celui monocouche appelé *le perceptron* et caractérisé de la manière suivante :

– Il possède N informations en entrée ;

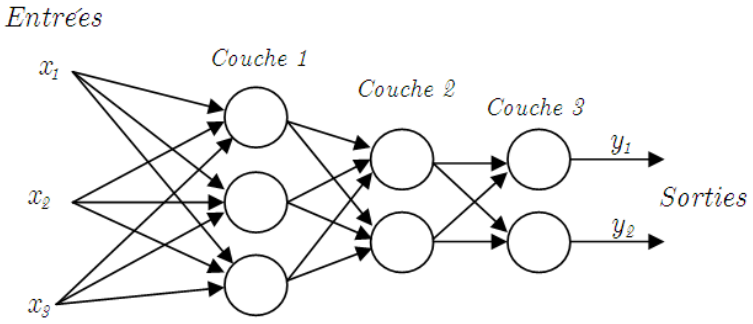
– Il est composé de p neurones, que l'on représente généralement alignés verticalement. Chacun peut en théorie avoir une fonction d'activation différente. En pratique, ce n'est généralement pas le cas ;

– Chacun des p neurones est connecté aux N informations d'entrée.



Le réseau de neurones possède ainsi N informations en entrée et P sorties, chaque neurone renvoyant sa sortie. Une utilisation courante est que chaque neurone de la couche représente une classe. Pour un exemple X donné, on obtient la classe de cet exemple en prenant la plus grande des p sorties.

3.6.2.2.2 Réseau de neurone multicouche Dans ce réseau, les neurones de la première couche reçoivent toutes les informations entrées, ceux de la deuxième reçoivent toutes les sorties des neurones de la première couche, et ainsi de suite jusqu'au neurone de sortie qui reçoit celles de la dernière couche.



Les couches 1 et 2 s'appellent des couches cachées tandis que la couche 3 est la couche de sortie.

3.6.2.3 Apprentissage des RNA

La tâche principale des réseaux de neurones artificiels est l'apprentissage pour la classification, qui est réalisée par un processus itératif d'adaptation des poids w_i pour arriver

à la meilleure fonction permettant d'avoir $f(x_i) = y_i, \forall\ i = 1..N$. Les valeurs des w_i sont initialisées aléatoirement, et corrigées selon les erreurs entre les y_i obtenus et attendus. Dans un réseau de neurones multicouches, la correction se fait dans le sens inverse du sens de propagation des données ce qui est appelé la *"backpropagation"*. À chaque présentation d'un exemple d'apprentissage au réseau, on passe par deux étapes :

1. Dans l'étape de propagation, les valeurs du vecteur d'entrée (l'exemple) sont reçues dans la couche d'entrée et propagées d'une couche à l'autre jusqu'à la sortie où un vecteur de sortie (les y_i) est obtenu.
2. Dans la phase de backpropagation, les w_i sont ajustés de la dernière couche jusqu'à la première de manière à rapprocher les y_i obtenus de ceux attendus.

Ces deux étapes sont répétées avec chaque exemple d'apprentissage pour obtenir à la fin un réseau de neurones artificiel entrainé. L'utilisation d'un RNA entrainé, se fait par l'injection des valeurs du vecteur de l'exemple à classifier, dans l'entrée et recevoir sa classe à la sortie par propagation.

Comme exemple des méthodes d'apprentissage prenons le cas de la méthode simple de décente par gradient dans un perceptron monocouche utilisant l'erreur quadratique. La méthode suit l'algorithme suivant :

Algorithme 4 ApprretissageRNA par décente de gradient

1: Créer n variables dw_i, pour $1 \leq i \leq n$, égales à 0
2: Prendre un exemple e_k, pour $1 \leq k \leq N$
3: Calculer la sortie obtenue avec les poids actuels, notée s_k
4: Rajouter à dw_i, pour tout $1 \leq i \leq n$, le nombre $\alpha(y_k - s_k)x_i$
5: Réitérer (3) et (4) sur chacun des exemples
6: pour $1 \leq i \leq n$, remplacer w_i par $w_i + dw_i$

α est un nombre réel appelé taux (ou vitesse) d'apprentissage qui doit être choisi minutieusement. Une valeur trop grande risque de faire osciller le réseau autour d'un minimum local et une valeur trop petite augmente le nombre d'itérations. En pratique, on diminue graduellement α au fur et à mesure des itérations.

Afin d'obtenir de bons résultats, il faudra passer plusieurs fois les exemples à chaque neurone, de sorte que les poids convergent vers des poids "idéaux".

Le problème avec cette méthode est que l'on corrige sur la globalité des exemples, ce qui fait que le réseau ne s'adaptera aux exemples qu'après un certain moment. Il y a une

autre méthode qui permet de corriger sur chacun des exemples, et qui se nomme méthode d'apprentissage de Widrow-Hoff.

Algorithme 5 ApprretissageRNA par Widrow-Hoff

Entrée :

- n poids reliant les n informations à notre neurone ayant des valeurs quelconques
- N exemples (X_k, y_k) où X_k est un vecteur à n composantes x_i, chacune représentant une information de cet exemple
- Le taux d'apprentissage α

Sortie : les n poids modifiés

pour Tout exemple = (X_k, y_k) **faire**

Calculer la sortie s_k du neurone

pour $1 \leq i \leq n$ **faire**

$w_i = w_i + \alpha(y_k - s_k)x_i$

fin pour

fin pour

Cette méthode est génélaisée pour faire entrainer un réseau de neurone multicouche. L'algorithme suivant illustre l'entrainent d'un RNA multicouche.

Algorithme 6 Apprentissage RNA multicouche

Initialiser aléatoirement les coefficients w_{ij} dans [-0.5 ; 0.5]

Répéter

Prendre un exemple (x, y)

Calculer la sortie s

pour toute cellule de sortie i **faire**

$\delta_i = s_i \times (1 - s_i) \times (y_i - s_i)$

fin pour

pour chaque couche de $q - 1$ à 1 **faire**

pour chaque cellule i de la couche courante **faire**

$\delta_i = s_i \times (1 - s_i) \times [\sum_{(k \in Succ(i))} (\delta_k \times w_{ki})]$

fin pour

fin pour

pour Pour tout poids w_{ij} **faire**

$w_{ij} = w_{ij} + \alpha \times \delta_i \times x_{ij}$

fin pour

Fin Répéter.

Les réseaux de neurones sont utilisés pour la classification ou la régression. Pour la régression les valeurs des y_i représentent la réponse de la fonction à estimer. Dans le cas de classification, si le cas est binaire, une seule sortie (0 ou 1) suffit. Si la classification est multi-classes, on utilise généralement une sortie pour chaque classe.
Plusieurs types de réseaux de neurones existent, ils se diffèrent dans la manière selon laquelle sont interconnectés les nœuds. Les réseaux récurrents, par exemple, consistent à propager les résultats au sens inverse de la propagation dans le calcul des w_i. Un autre type est celui des cartes auto-organisatrices de Kohonen qui utilise un principe de compétition pour ne prendre que les résultats des meilleurs nœuds dans les calculs. Ce type de réseaux de neurones est utilisé généralement dans l'apprentissage non supervisé.

Certes, les réseaux de neurones artificiels permettent de surmonter le problème d'analyse d'un système donné pour le modéliser. On peut simuler son comportement uniquement à partir d'un certain nombre d'exemples observés. Mais, par contre, ils représentent des problèmes remarquables qui ont limité leur évolution en face d'autres techniques tel que les SVMs.

– Un réseau de neurones artificiel représente une boîte noire, et il est très difficile voire impossible d'analyser et comprendre son fonctionnement en face d'un problème donné, ce qui empêche de choisir la structure (type, nombre de nœuds, organisation, connexions,...etc) la mieux adaptée à ce problème.
– L'ordre de présentation des exemples d'entrainement au réseau influe directement sur les résultats obtenus. Pour surmonter ce problème, il est nécessaire de répéter au moins la phase d'entrainement avec des ordres différents des exemples ce qui augmente considérablement le temps d'apprentissage.
– Dans le cas des bases de données, les réseaux de neurones artificiels ne permettent pas de traiter des exemples avec des attributs symboliques (catégoriels) qu'après un encodage adapté, à l'inverse de plusieurs autres techniques d'apprentissages tel que les SVMs et les arbres de décision.
– Les RNAs représentent un inconvénient majeur qui est leur sensibilité aux minimas locaux et la possibilité de leur divergence. L'ambiguïté de leur fonctionnement empêche d'éviter de tels cas.

Pour toutes ces raisons beaucoup de travaux récents de comparaison, favorisent les SVMs par rapport aux RNAs dans plusieurs applications.

3.7 Machines à vecteur support

Les machines à vecteur support se situent sur l'axe de développement de la recherche humaine des techniques d'apprentissage. Les SVMs sont une classe de techniques d'apprentissage introduite par Vladimir Vapnik au début des années 90, elles reposent sur une théorie mathématique solide à l'inverse des méthodes de réseaux de neurones. Elles ont été développées au sens inverse du développement des réseaux de neurones : ces derniers ont suivi un chemin heuristique de l'application et l'expérimentation vers la théorie ; alors que les SVMs sont venues de la théorie du son vers l'application.

3.7.1 SVMs binaires

Le cas le plus simple est celui où les données d'entrainement viennent uniquement de deux classes différentes (+1 ou -1), on parle alors de classification binaire. L'idée des SVMs est de rechercher un hyperplan (droite dans le cas de deux dimensions) qui sépare le mieux ces deux classes. Si un tel hyperplan existe, c'est-à-dire si les données sont linéairement séparables, on parle d'une machine à vecteur support à marge dure (Hard margin).

L'hyperplan séparateur est représenté par l'équation suivante :

$$H(x) = w^T x + b$$

Où w est un vecteur de m dimensions et b est un terme. La fonction de décision, pour un exemple x, peut être exprimée comme suit :

$$\begin{cases} \text{Classe} = 1 & \text{Si} \quad H(x) > 0 \\ \text{Classe} = -1 & \text{Si} \quad H(x) < 0 \end{cases}$$

Puisque les deux classes sont linéairement séparables, il n'existe aucun exemple qui se situe sur l'hyperplan, c-à-d qui satisfait $H(x) = 0$. Il convient alors d'utiliser la fonction de décisions suivante :

$$\begin{cases} \text{Classe} = 1 & \text{Si} \quad H(x) > 1 \\ \text{Classe} = -1 & \text{Si} \quad H(x) < -1 \end{cases}$$

Les valeurs +1 et -1 à droite des inégalités peuvent être des constantes quelconques +a et -a, mais en divisant les deux parties des inégalités par a, on trouve les inégalités précédentes qui sont équivalentes à l'équation suivante :

$$y_i(w^T x_i + b) \geq 1, i = 1..n$$

L'hyperplan $w^T x + b = 0$ représente un hyperplan séparateur des deux classes, et la distance entre cet hyperplan et l'exemple le plus proche s'appelle la marge. La région qui se trouve

entre les deux hyperplans $w^T x + b = -1$ et $w^T x + b = +1$ est appelée la région de généralisation de la machine d'apprentissage. Plus cette région est importante, plus est la capacité de généralisation de la machine. La maximisation de cette région est l'objectif de la phase d'entrainement qui consiste, pour la méthode SVM, à rechercher l'hyperplan qui maximise la région de généralisation c-à-d la marge. Un tel hyperplan est appelé *"hyperplan de séparation optimale"*. En supposant que les données d'apprentissage ne contiennent pas des données bruitées (mal-étiquetées) et que les données de test suivent la même probabilité que celle des données d'entraînement, l'hyperplan de marge maximale va certainement maximiser la capacité de généralisation de la machine d'apprentissage.

La détermination de l'hyperplan optimal passe par la détermination de la distance euclidienne minimale entre l'hyperplan et l'exemple le plus proche des deux classes. Puisque le vecteur w est orthogonal sur l'hyperplan séparateur, la droite parallèle à w et reliant un exemple x à l'hyperplan est donnée par la formule :

$$\frac{aw}{\|w\|} + x = 0$$

Où a représente la distance entre x et l'hyperplan. La résolution de cette équation, donne :

$$a = -\frac{w^T x + b}{\|w\|}$$

La distance de tout exemple de l'hyperplan doit être supérieure ou égale à la marge δ :

$$\frac{y_i(w^T x_i + b)}{\|w\|} \geq \delta$$

Si une paire (w, b) est une solution alors (aw, ab) est une solution aussi où a est un scalaire. On impose alors la contrainte suivante :

$$\|w\| \, \delta \geq 1$$

Pour trouver l'hyperplan séparateur qui maximise la marge, on doit déterminer, à partir des deux dernières inégalités, le vecteur w qui possède la norme euclidienne minimale et qui vérifie la contrainte de l'équation, de bonne classification des exemples d'entrainement. L'hyperplan séparateur optimal peut être obtenu en résolvant le problème de l'équation :

$$\begin{cases} \text{Minimiser} & \frac{1}{2}\|w\|^2 \\ \text{sous} \quad \text{contraintes} & \\ y_i(w^T x_i + b) \geq 1 & \forall i = 1..n \end{cases}$$

Remarquons que nous pouvons obtenir le même hyperplan même en supprimant toutes les données qui vérifient l'inégalité de la contrainte. Les données qui vérifient l'égalité de la

contrainte s'appellent les vecteurs supports, et ce sont ces données seules qui contribuent à la détermination de l'hyperplan. Dans la figure, les données qui se trouvent sur les deux droites $+1$ et -1 représentent les vecteurs supports.

Le problème de l'équation est un problème de programmation quadratique avec contraintes linéaires. Dans ce problème, les variables sont w et b, c-à-d que le nombre de variables est égal à $m+1$. Généralement, le nombre de variables est important ce qui ne permet pas d'utiliser les techniques classiques de programmation quadratique. Dans ce cas le problème est convertit en un problème dual équivalent sans contraintes de l'équation suivante qui introduit les multiplicateurs de Lagrange :

$$Q(w,b,\alpha) = \frac{1}{2}w^T w - \sum_{i=1}^{n} \alpha_i \left\{ y_i(w^T x_i + b) - 1 \right\}$$

Où les α_i sont les multiplicateurs non négatifs de Lagrange. L'optimum de la fonction objective Q peut être obtenu en la minimisant par rapport à w et b et en la maximisant par rapport aux α_i. À l'optimum de la fonction objective, ses dérivées par rapports aux variables w et b s'annulent ainsi que le produit des α_i aux contraintes :

$$\begin{cases} \frac{\partial Q(w,b,\alpha)}{\partial w} = 0 & (a) \\ \frac{\partial Q(w,b,\alpha)}{\partial b} = 0 & (b) \\ \alpha_i \left\{ y_i(w^T x_i + b) - 1 \right\} = 0 & (c) \\ \alpha_i \geq 0 & (d) \end{cases}$$

De (a) on déduit :

$$\begin{cases} w = \sum\limits_{i=1}^{n} \alpha_i y_i x_i \\ \sum\limits_{i=1}^{n} \alpha_i y_i = 0 \end{cases}$$

En remplaçant dans la fonction objective, on obtient le problème dual à maximiser suivant :

$$\begin{cases} \text{Maximiser} & Q(\alpha) = \sum\limits_{i=1}^{n} \alpha_i - \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{Sous contraintes} & \\ & \sum\limits_{i=1}^{n} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \end{cases}$$

Si le problème de classification est linéairement séparable, une solution optimale pour les α_i existe. Les exemples ayant des $\alpha_i \neq 0$ représentent les vecteurs supports appartenant aux deux classes. La fonction de décision est donnée par :

$$H(x) = \sum_{S} \alpha_i y_i x^T x_i + b$$

Où S représente l'ensemble des vecteurs supports. b peut être calculé à partir de n'importe quel vecteur support par l'équation :

$$b = y_i - w^T x_i$$

D'un point de vue précision, on prend la moyenne de b pour tous les vecteurs supports :

$$b = \frac{1}{|S|} \sum_{i \in S} y_i - w^T x_i$$

La fonction de décision H peut être calculée, donc, pour chaque nouvel exemple x par la fonction $H(x)$ et la décision peut être prise comme suit :

$$\begin{cases} x \in Classe + 1 & si \quad H(x) > 0 \\ x \in Classe - 1 & si \quad H(x) < 0 \\ x \quad \text{inclassifiable} & si \quad H(x) = 0 \end{cases}$$

La zone $-1 < H(x) < 1$ est appelée la zone de généralisation.

Si on prend un exemple x_k de l'ensemble d'entrainement appartenant à la classe y_k et on calcule sa fonction de décision $H(x_k)$, on peut se trouver dans l'un des cas suivants :

1. $y_k * H(x_k) > 1$: dans ce cas l'exemple est bien classé et ne se situe pas dans la zone de la marge. Il ne représente pas un vecteur support.
2. $y_k * H(x_k) = 1$: dans ce cas l'exemple est bien classé et se situe aux frontières de la zone de la marge. Il représente un vecteur support.
3. $0 < y_k * H(x_k) < 1$: dans ce cas l'exemple est bien classé et se situe dans de la zone de la marge. Il ne représente pas un vecteur support.
4. $y_k * H(x_k) < 0$: dans ce cas l'exemple se situe dans le mauvais coté, il est mal classé et ne représente pas un vecteur support.

3.7.1.1 Cas non linéairement séparable

En réalité, un hyperplan séparateur n'existe pas toujours, et même s'il existe, il ne représente pas généralement la meilleure solution pour la classification. En plus une erreur d'étiquetage dans les données d'entrainement (un exemple étiqueté $+1$ au lieu de -1 par exemple) affectera crucialement l'hyperplan.

Dans le cas où les données ne sont pas linéairement séparables, ou contiennent du bruit (outliers : données mal étiquetées) les contraintes ne peuvent être vérifiées, et il y a nécessité de les relaxer un peu. Ceci peut être fait en admettant une certaine erreur de classification des données ce qui est appelé "SVM à marge souple (Soft Margin)".

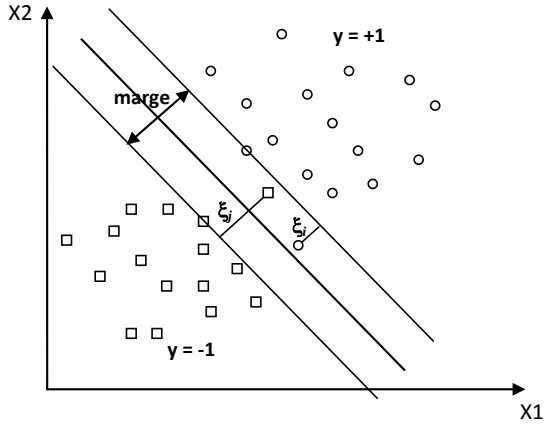


FIGURE 3.4 – SVM binaire à marge souple

On introduit alors sur les contraintes des variables ξ_i dites de relaxation pour obtenir la contrainte de l'équation :

$$y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1..n$$

Grâce aux variables de relaxation non négatives ξ_i, un hyperplan séparateur existera toujours.

Si $\xi_i < 1$, x_i ne respecte pas la marge mais reste bien classé, sinon x_i est mal classé par l'hyperplan. Dans ce cas, au lieu de rechercher uniquement un hyperplan séparateur qui maximise la marge, on recherche un hyperplan qui minimise aussi la somme des erreurs permises c-à-d minimiser $Q(w) = \sum_{i=1}^{n} \xi_i$.

Le problème dual devient donc :

$$\begin{cases} \text{Minimiser} & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{sous contraintes} & \\ y_i(w^T x_i + b) \geq 1 - \xi_i & i = 1..n \\ \xi_i \geq 0 & \end{cases}$$

Où C est un paramètre positif libre (mais fixe) qui représente une balance entre les deux termes de la fonction objective (la marge et les erreurs permises) c-à-d entre la maximisation de la marge et la minimisation de l'erreur de classification. On obtient le problème dual de l'équation suivante où on introduit les multiplicateurs de Lagrange α_i et β_i :

$$Q(w, b, \alpha, \xi, \beta) = \frac{1}{2}w^T w + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i y_i(w^T x_i + b) - 1 + \xi_i - \sum_{i=1}^{n}\beta_i\xi_i$$

À la solution optimale, les dérivées par rapport aux variables w, b, α, β s'annulent ainsi que le produit des contraintes aux multiplicateurs. Les conditions suivantes sont alors vérifiées :

$$\begin{cases} \frac{\partial Q(w,b,\xi,\alpha,\beta)}{\partial w} = 0 & (a) \\ \frac{\partial Q(w,b,\xi,\alpha,\beta)}{\partial b} = 0 & (b) \\ \alpha_i \left\{ y_i(w^T x_i + b) - 1 + \xi_i \right\} = 0 & (c) \\ \beta_i \xi_i = 0 & (d) \\ \alpha_i \geq 0 \,;\, \beta_i \geq 0 \,;\, \xi_i \geq 0 & (e) \end{cases}$$

On déduit :

$$\begin{cases} w = \sum_{i=1}^{n} \alpha_i y_i x_i \\ \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \alpha_i + \beta_i = C \end{cases}$$

En remplaçant cette équation dans la fonction objective, on obtient le problème dual suivant :

$$\begin{cases} \text{Maximiser} & Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{sous contraintes} & \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \end{cases}$$

La seule différence avec la SVM à marge dure est que les α_i ne peuvent pas dépasser C, ils peuvent être dans l'un des trois cas suivants :

1. $\alpha_i = 0 \Rightarrow \beta_i = C \Rightarrow \xi_i = 0$: x_i est bien classé,
2. $0 < \alpha_i < C \Rightarrow \beta_i > 0 \Rightarrow \xi_i = 0 \Rightarrow y_i(w^T x_i + b) = 1$: x_i est un vecteur support et est appelé dans ce cas vecteur support non borné (unbounded),
3. $\alpha_i = C \Rightarrow \beta_i = 0 \Rightarrow \xi_i \geq 0 \Rightarrow y_i(w^T x_i + b) = 1 - \xi_i$: x_i est un vecteur support appelé dans ce cas vecteur support borné (bounded). Si $0 \leq \xi_i < 1$, x_i est bien classé, sinon x_i est mal classé.

Ces conditions sur les α_i sont appelées les conditions de Karush-Kuhn-Tucker (KKT), elles sont très utilisées par les algorithmes d'optimisation pour rechercher les α_i optimums et par conséquent l'hyperplan optimal.

La fonction de décision est alors calculée de la même manière que dans le cas des SVMs à marge dure mais uniquement à base des vecteurs supports non bornés par :

$$H(x) = \sum_{i \in U} \alpha_i y_i x_i^T x + b$$

Pour les vecteurs supports non bornés, nous avons :

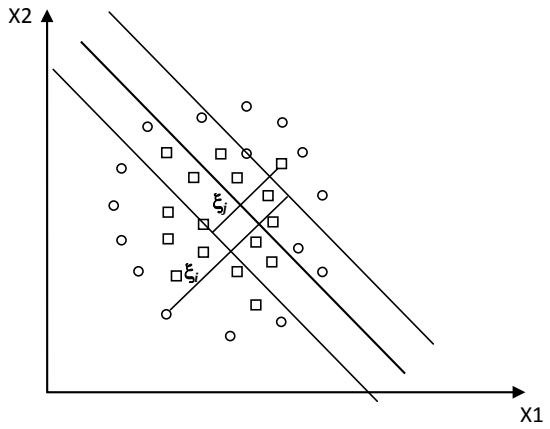
$$b = y_i - w^T x_i$$

Pour garantir une bonne précision, on prend la moyenne de b pour tous les vecteurs supports non bornés :

$$b = \frac{1}{|U|} \sum_{i \in U} y_i - w^T x_i$$

3.7.2 Utilisation des noyaux

Le fait d'admettre la mal-classification de certains exemples, ne peut pas toujours donner une bonne généralisation pour un hyperplan même si ce dernier est optimisé.



Plutôt qu'une droite, la représentation idéale de la fonction de décision serait une représentation qui colle le mieux aux données d'entrainement.

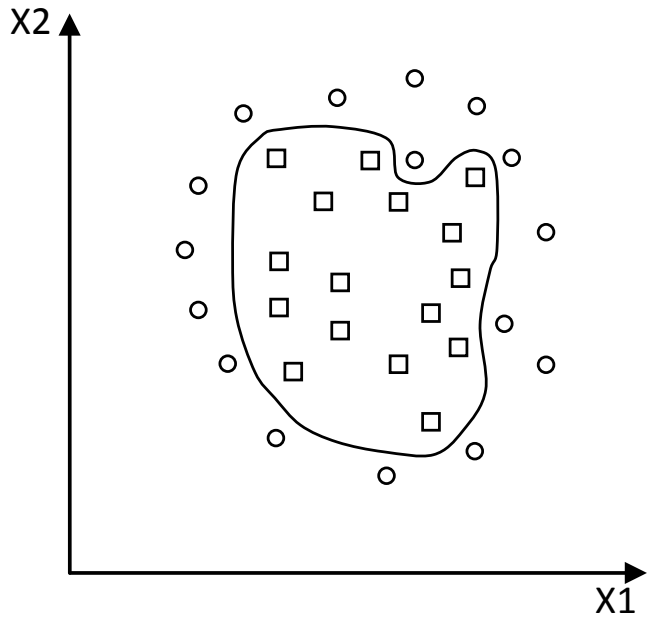


FIGURE 3.5 – Représentation idéale de la fonction de décision

La détermination d'une telle fonction non linéaire est très difficile voire impossible. Pour cela les données sont amenées dans un espace où cette fonction devient linéaire, cette astuce permet de garder les mêmes modèles de problèmes d'optimisation vus dans les sections précédentes, utilisant les SVMs basées essentiellement sur le principe de séparation linéaire. Cette transformation d'espace est réalisée souvent à l'aide d'une fonction $F = \{\phi(x)|x \in X\}$ appelé *"Mapping function"* et le nouvel espace est appelé espace de caractéristiques *"Features space"*.

Dans ce nouvel espace de caractéristiques, la fonction objective à optimiser devient :

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

Où $\langle \phi(x_i), \phi(x_j) \rangle$ est le produit scalaire des deux images des vecteurs x_i et x_j dans le nouvel espace et dont le résultat est un scalaire.

Dans le calcul de l'optimum de la fonction, on utilise une astuce appelée *"Noyau" ("Kernel")*, au lieu de calculer $\phi(x_i), \phi(x_j)$ et leur produit scalaire, on calcule plutôt une fonction $K(x_i, x_j)$ qui représente à la fois les deux transformations (qui peuvent être inconnues) et leur produit scalaire. Cette fonction permet de surmonter le problème de détermination de la transformation ϕ et permet d'apprendre des relations non linéaires par des machines linéaires. En pratique, il existe certains noyaux qui sont très utilisés et qui sont considérés comme standards. Une fois le noyau choisi, la fonction objective peut être calculée comme suit :

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

Et la fonction de décision devient :

$$H(x) = \sum_{i \in S} \alpha_i y_i K(x_i, x) + b$$

Où S représente l'ensemble des vecteurs supports.

Exemples de noyaux

- Noyau linéaire : Si les données sont linéairement séparables, on n'a pas besoin de changer d'espace, et le produit scalaire suffit pour définir la fonction de décision :

$$K(x_i, x_j) = x_i^T x_j$$

- Noyau polynomial : Le noyau polynomial élève le produit scalaire à une puissance naturelle d :

$$K(x_i, x_j) = (x_i^T x_j)^d$$

  Si $d = 1$ le noyau devient linéaire. Le noyau polynomial dit non homogène $K(x_i, x_j) = (x_i^T x_j + C)^d$ est aussi utilisé.
- Noyau RBF : Les noyaux RBF (Radial Basis functions) sont des noyaux qui peuvent être écrits sous la forme : $K(x_i, x_j) = f(d(x_i, x_j))$ où d est une métrique sur X et f est une fonction dans $\Re$. Un exemple des noyaux RBF est le noyau Gaussien :

$$K(x_i, x_j) = e^{\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)}$$

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

3.7.3 Architecture générale d'une machine à vecteur support

Une machine à vecteur support, recherche à l'aide d'une méthode d'optimisation, dans un ensemble d'exemples d'entrainement, des exemples, appelés vecteurs support, qui caractérisent la fonction de séparation. La machine calcule également des multiplicateurs associés à ces vecteurs. Les vecteurs supports et leurs multiplicateurs sont utilisés pour calculer la fonction de décision pour un nouvel exemple. Le schéma de la figure suivante résume l'architecture générale d'une SVM dans le cas de la reconnaissance des chiffres manuscrits.

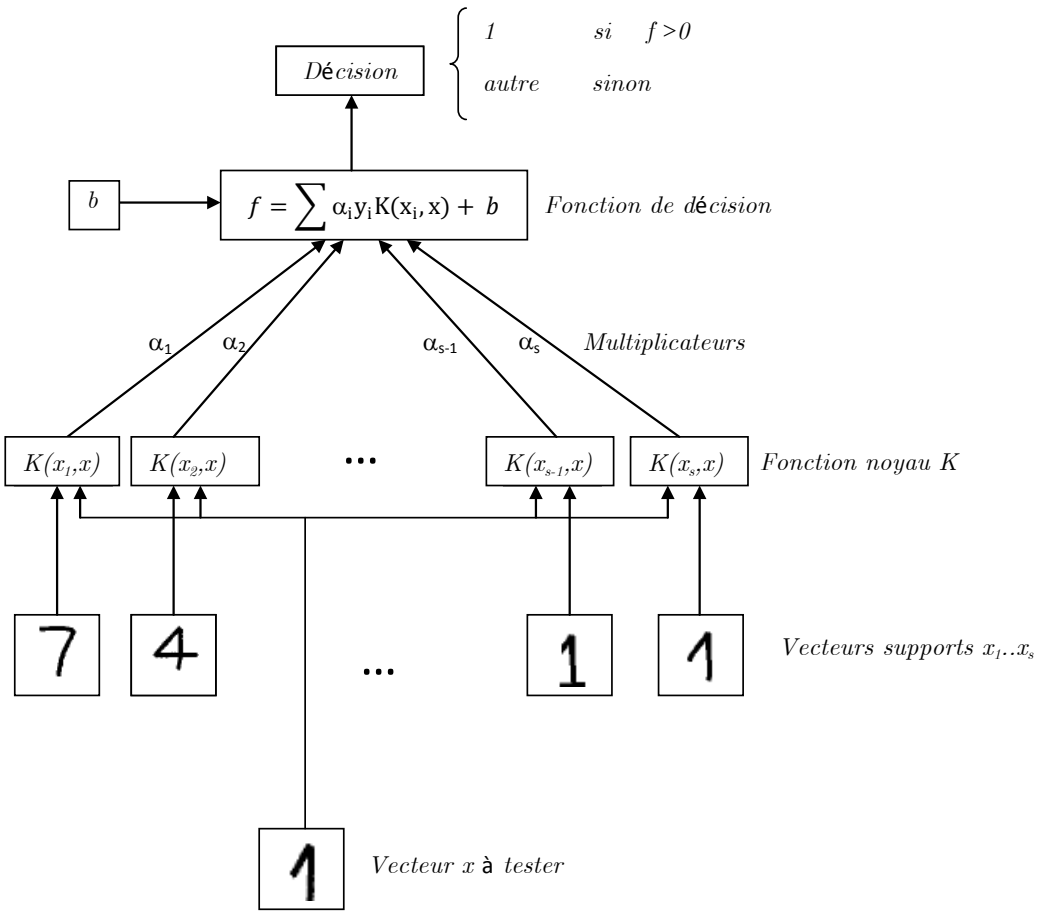


FIGURE 3.7 – Architecture d'une machine à vecteur support

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

3.7.4 SVMs multiclasse

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

Les méthodes des machines à vecteur support multiclasse, réduisent le problème mul-

ticlasse à une composition de plusieurs hyperplans biclasses permettant de tracer les frontières de décision entre les différentes classes. Ces méthodes décomposent l'ensemble d'exemples en plusieurs sous ensembles représentant chacun un problème de classification binaire. Pour chaque problème un hyperplan de séparation est déterminé par la méthode SVM binaire. On construit lors de la classification une hiérarchie des hyperplans binaires qui est parcourue de la racine jusqu'à une feuille pour décider de la classe d'un nouvel exemple. On trouve dans la littérature plusieurs méthodes de décomposition :

3.7.5 Une-contre-reste (1vsR)

C'est la méthode la plus simple et la plus ancienne. Selon la formulation de Vapnik, elle consiste à déterminer pour chaque classe k un hyperplan $H_k(w_k, b_k)$ la séparant de toutes les autres classes. Cette classe k est considérée comme étant la classe positive $(+1)$ et les autres classes comme étant la classe négative (-1), ce qui résulte, pour un problème à K classes, en K SVM binaires. Un hyperplan H_k est défini pour chaque classe k par la fonction de décision suivante :

$$\begin{aligned} H_k(x) &= \text{signe}(\langle w_k, x\rangle + b_k) \\ &= \begin{cases} +1 & \text{si} \quad f_k(x) > 0; \\ 0 & \text{sinon} \end{cases} \end{aligned}$$

La valeur retournée de l'hyperplan permet de savoir si x appartient à la classe k ou non. Dans le cas où il n'appartient pas à k $(H_k(x) = 0)$, nous n'avons aucune information sur l'appartenance de x aux autres classes. Pour le savoir, on présente x à tous les hyperplans, ce qui donne la fonction de décision de l'équation suivante :

$$k^* = \underbrace{Arg}_{(1 \leq k \leq K)} \; Max(H_k(x))$$

Si une seule valeur $H_k(x)$ est égale à 1 et toutes les autres sont égales à 0, on conclut que x appartient à la classe k. Le problème est que l'équation peut être vérifiée pour plus d'une classe, ce qui produit des régions d'ambiguïté, et l'exemple x est dit non classifiable. La figure suivante représente un cas de séparation de 3 classes.

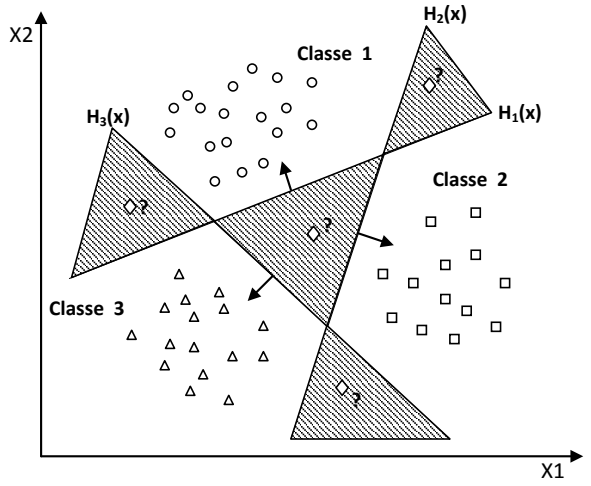


FIGURE 3.8 – Approche une-contre-reste avec des zones d'indécision

Pour surmonter cette situation, la méthode 1vsR utilise le principe de "le gagnant prend tout" ("winner-takes-all") : la classe k retenue est celle qui maximise $f_k(x) = \langle w_k, x \rangle + b_k$ de l'équation :

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

Géométriquement interprétée, tout nouvel exemple x est affecté à la classe dont l'hyperplan est le plus loin de x, parmi les classes ayant $H(x) = 1$.

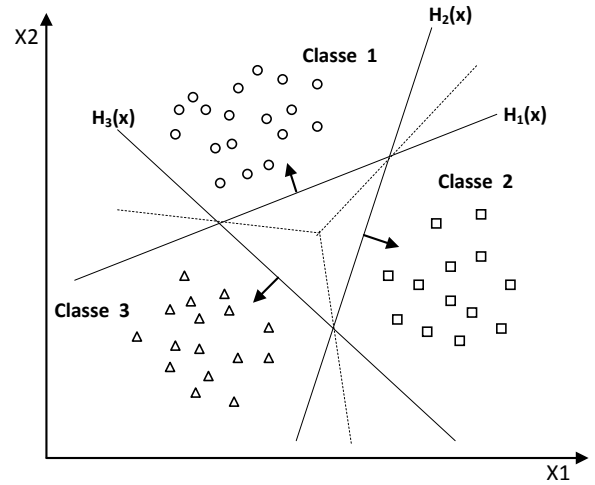


FIGURE 3.9 – Résolution des cas d'indécision dans la méthode 1vsR

La méthode 1vsR peut être utilisée pour découvrir même les cas de rejet où un exemple

n'appartient à aucune des K classes. Pour cela, on prend les deux fonctions de décision les plus élevées, puis on calcule leur différence, si elle est au dessous d'un certain seuil, l'exemple est rejeté.

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

3.7.6 Une-contre-une (1vs1)

Cette méthode, appelée aussi *"pairwise"*, revient à Kner et ses co-auteurs qui l'ont proposée pour les réseaux de neurones. Elle consiste à utiliser un classifieur pour chaque paire de classes. Au lieu d'apprendre K fonctions de décisions, la méthode 1vs1 discrimine chaque classe de chaque autre classe, ainsi $K(K-1)/2$ fonctions de décisions sont apprises.

Pour chaque paire de classes (k, s), la méthode 1vs1 définit une fonction de décision binaire $h_{ks} : \Re \rightarrow \{-1, +1\}$. L'affectation d'un nouvel exemple se fait par liste de vote. On teste un exemple par le calcul de sa fonction de décision pour chaque hyperplan. Pour chaque test, on vote pour la classe à laquelle appartient l'exemple (classe gagnante). On définit pour le faire la fonction de décision binaire $H_{ks}(x)$ de l'équation suivante.

$$\begin{aligned} H_{ks}(x) &= \text{signe}(f_{ks}(x)) \\ &= \begin{cases} +1 & \text{si} f_{ks}(x) > 0; \\ 0 & \text{sinon} \end{cases} \end{aligned}$$

Sur la base des $K(K-1)/2$ fonctions de décision binaires, on définit K autres fonctions de décision :

$$H_k(x) = \sum_{s=1}^{m} H_{ks}(x)$$

Un nouvel exemple est affecté à la classe la plus votée. La règle de classification d'un nouvel exemple x est donnée par l'équation :

$$k^* = \underbrace{Arg}_{(1 \leq k \leq K)} (MaxH_k(x))$$

Malheureusement, cette fonction peut être vérifiée pour plusieurs classes, ce qui produit des zones d'indécisions. La méthode de vote affecte dans ce cas, un exemple aléatoirement à l'une des classes les plus votées.

La Figure suivante représente un exemple de classification de trois classes avec la zone d'indécision.

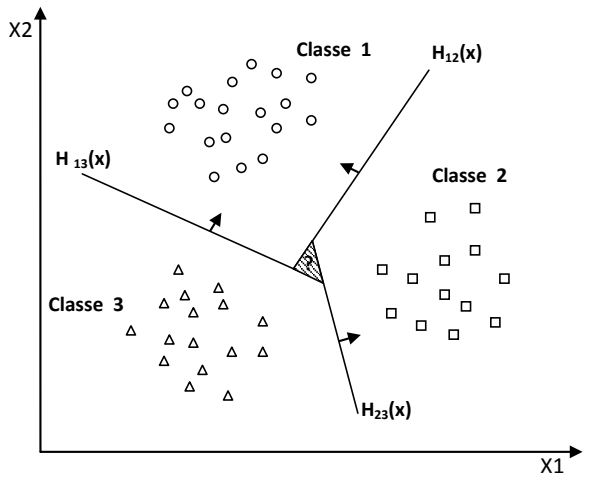


FIGURE 3.10 – Approche une-contre-une

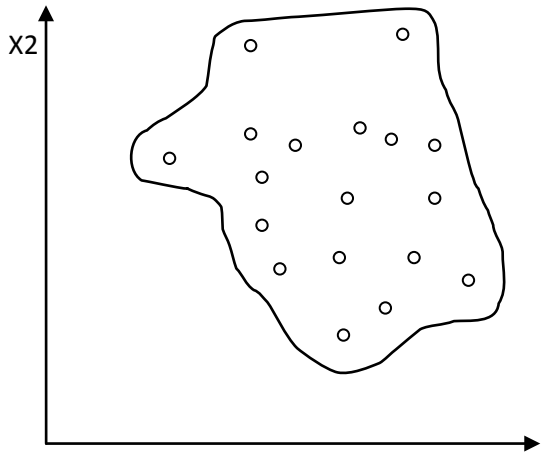
Bien que La méthode 1vs1 utilise, pour l'entrainement, un nombre plus important d'hyperplans que la méthode 1vsR, elle est souvent plus rapide. Cela est du, d'une part, au nombre limité d'exemples utilisés pour entrainer chaque hyperplan, et d'autre part, à la simplicité des problèmes à résoudre. En effet, chaque deux classes prises à part sont moins chevauchées que toutes les classes.

3.7.7 SVM monoclasse (Novelty detection)

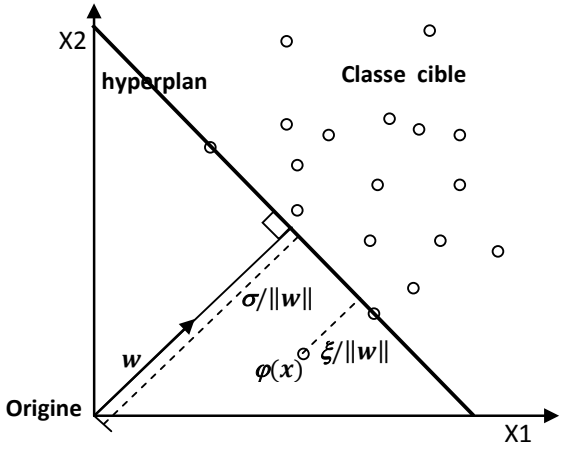
Dans les machines à vecteur support binaires et multiclasse précédentes, nous avons toujours des exemples positifs et d'autres négatifs c-à-d des exemples et des contre-exemples. De telles informations ne sont pas disponibles dans tous les cas d'application. Parfois, il est très coûteux, voire impossible, de trouver des contre-exemples qui représentent réellement la classe négative. Prenons l'exemple de reconnaissance d'une catégorie particulière de pièces par un robot dans une usine, il est facile d'avoir des exemples suffisants de cette pièce, mais il est difficile d'avoir des exemples de toutes les pièces différentes. Il est souhaitable, dans de tels cas, d'avoir un modèle de décision permettant de reconnaître autant d'exemples possibles de cette catégorie et de rejeter tous les autres. Ce problème est sou-

vent appelé *"Novelty detection"* ou détection des nouveautés, puisque le modèle de décision connaît un ensemble d'exemples et détecte tous ce qui est nouveau (étranger ou outlier).

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



Pour résoudre ce cas de problèmes, la technique SVM monoclasse utilise le même modèle binaire avec une astuce en plus ; l'origine de l'espace est considérée comme étant la seule instance de la classe négative. Le problème revient, donc, à trouver un hyperplan qui sépare les exemples de la classe cible de l'origine, et qui maximise la marge entre les deux.



Le problème est modélisé par le problème primal de programmation quadratique de l'équation suivante, dont l'objectif est de maximiser la marge et minimiser les erreurs de classification. La contrainte est la bonne classification des exemples d'entrainement.

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

Où N est le nombre d'exemples de la classe cible, (w, ρ) les paramètres permettant de localiser l'hyperplan, ξ_i représentent les erreurs permises sur les exemples, pénalisées par le paramètre v et ϕ est une transformation d'espace semblable à celle du cas binaire.
Une fois (w, ρ) déterminés, tout nouvel exemple pourra être classé par la fonction de décision de l'équation suivante :

$$f(x) =< w, \phi(x_i) > -\rho$$

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

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

$$\begin{cases} \textit{Minimiser}_{\alpha} & \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \\ \textit{sous contraintes} & \\ & \sum_{i=1}^{n} \alpha_i = 1 \\ & 0 \leq \alpha_i \leq \frac{1}{vN} \end{cases}$$

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

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

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

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

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

Où ρ peut être déterminé à partir d'un exemple x_i d'apprentissage dont $\alpha_i \neq 0$ par l'équation :

$$\rho = \sum_j \alpha_j K(x_j, x_i)$$

3.7.8 Implémentation des SVMs

L'implémentation des SVMs pour la classification binaire consiste à la résolution du problème dual de programmation quadratique de l'équation suivante pour déterminer l'hyperplan de marge maximale.

$$\begin{cases} \text{Maximiser} & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{sous contraintes} & \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \end{cases}$$

Où les x_i sont les exemples d'entrainement, n leur nombre et $y_i = \pm 1$ leur classes respectives, les α_i les multiplicateurs de Lagrange à déterminer et K est le noyau utilisé.

La résolution de ce problème consiste à déterminer les α_i optimaux. Si le nombre d'exemples est modeste (de l'ordre de 1000), les méthodes classiques de programmation quadratique tel que les méthodes quasi-Newton ou les méthodes du point intérieur, peuvent être utilisées. Si par contre le nombre d'exemples est important (le cas habituel), des méthodes d'optimisation sont indispensables pour résoudre le problème en un temps acceptable.

En pratique, quand n est élevé, deux problèmes se posent : premièrement la taille de la matrice du noyau qui devient insupportable par la mémoire principale, deuxièmement le temps de recherche des α_i optimaux est exhaustif.

Pour résoudre ces problèmes, plusieurs méthodes ont été développées. La méthode de *shnuking*, effectue l'entrainement sur un nombre limité d'exemples, choisis par une heuristique, puis ajoute itérativement les autres jusqu'à atteindre l'hyperplan optimal. Parmi les implémentations classiques de cette méthode on trouve le SVMlight.

La méthode SMO (sequential minimal optimisation), est la méthode la plus utilisée actuellement, elle consiste à optimiser à chaque itération, deux α_i conjointement.

Optimisation séquentielle minimale

L'algorithme d'optimisation séquentielle optimale (SMO) a été proposé premièrement par Platt et all en 1999, c'est l'algorithme le plus utilisé actuellement pour les problèmes de grande taille. Cet algorithme pousse la décomposition des α_i à son maximum : à chaque itération, uniquement deux multiplicateurs de Lagrange α_i du problème dual sont optimisés. En effet, la première contrainte du problème de l'équation précédente implique que le plus petit nombre de α_i qui peuvent être optimisés conjointement est de 2. À Chaque fois qu'un multiplicateur est mis à jour, un autre multiplicateur au moins doit être ajusté afin de maintenir la contrainte satisfaite.

À chaque itération, l'algorithme choisi à l'aide d'une heuristique deux α_i et les optimise conjointement tout en gardant les valeurs des autres multiplicateurs inchangées.

Le point fort de l'algorithme est que l'optimisation des deux multiplicateurs choisis se fait analytiquement, ce qui réduit considérablement le temps d'entrainement. En plus, l'algorithme ne fait aucune opération matricielle et n'a pas besoin de maintenir la matrice du noyau en mémoire.

Plusieurs optimisations peuvent être ajoutées à l'algorithme pour l'accélérer davantage. Premièrement, la matrice du noyau K peut être gérée par une méthode de cache tel que LRU (least recently used), pour garder une simple partie de la matrice en mémoire et mettre à jour uniquement les entrées les plus anciennes. Cette technique peut garantir de trouver jusqu'à 80 % des éléments dans une cache d'une taille de 10 % de la matrice K.

Même pour la fonction $f(x_i)$ calculée plusieurs fois, elle peut être mise en cache aussi par un traitement pareil.

Certaines optimisations proposent de larguer les exemples dont les multiplicateurs correspondants atteignent leurs limites supérieures ou inférieures (0 ou C), au cours de progression de l'algorithme. L'idée consiste à écarter les exemples x_i dont les $\alpha_i = 0$, au cours de progression de l'algorithme puisque ce sont uniquement les exemples avec $\alpha_i \neq 0$ qui influencent la solution finale.

Divers packages d'implémentation du SMO peuvent être trouvés dans la littérature, particulièrement LIBSVM et SVMTORCH.

Chapitre 4

Régression

4.1 Définition

La régression est la méthode utilisée pour l'estimation des valeurs continues. Son objectif est de trouver le meilleur modèle qui décrit la relation entre une variable continue de sortie et une ou plusieurs variables d'entrée. Il s'agit de trouver une fonction f qui se rapproche le plus possible d'un scénario donné d'entrées et de sorties.

4.2 Régression linéaire simple

Le modèle le plus utilisée actuellement est le modèle linéaire qui est utilisé pour décrire la relation entre une seule variable de sortie et une ou plusieurs variables d'entrée. Ce modèle est appelé la régression linéaire.

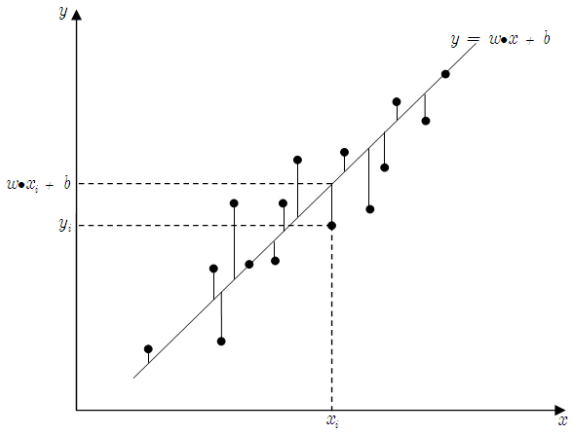


FIGURE 4.1 – Régression linéaire simple

Dans les problèmes de régression, les exemples d'entrainement sont associés à des va-

leurs numériques plutôt qu'à des étiquettes discrètes . Le problème est formulé comme suit : Soit D l'ensemble d'entrainement définit par :

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\} \subset \Re^m \times \Re \tag{4.1}$$

Le problème consiste à trouver, en utilisant D, une fonction $\hat{f} : \Re^m \rightarrow \Re$ qui vérifie $\hat{f}(x_i) \approx y_i; \quad \forall i = 1..n$. En pratique une telle fonction est difficile à trouver, on recherche plutôt une fonction qui rapproche des y_i, en d'autre terme qui minimise la différence entre les $\hat{f}(x_i)$ et les y_i :

$$Min \quad \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2 \tag{4.2}$$

Souvent, $\hat{f}$ est considérée comme fonction linéaire : $\hat{f} = \langle w, x \rangle + b$, où w est un vecteur et b est un scalaire. Le problème revient donc à trouver un hyperplan caractérisé par w^* et b^* qui minimise l'écart global entre $\hat{f}$ et les y_i (équation 4.4).

$$(w^*, b^*) = argmin_{w,b} \sum_{i=1}^{n} (y_i - \langle w, x_i \rangle - b)^2 \tag{4.3}$$

Dans le cas où les exemples d'entrainement sont réels ($m = 1$), on parle de régression linéaire simple, sinon on parle de régression linéaire multiple. La figure 4.1 représente un exemple de régression linéaire simple. l'équation à minimiser est :

$$(w, b) = argmin_{w,b} \sum_{i=1}^{n} (y_i - wx_i - b)^2 \tag{4.4}$$

Pour minimiser, on dérive par rapport à w et b puis on annule les dérivées, on obtient alors :

$$\begin{cases} w & = \overline{Y} - b\overline{X} \\ b & = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})}{\sum_{i=1}^{n} (X_i - \overline{X})} \end{cases} \tag{4.5}$$

Pour chaque nouvelle entrée X, on calcule l'estimation de la sortie correspondante Y comme suit :

$$Y = w + bX \tag{4.6}$$

4.3 Régression linéaire multiple

La régression linéaire multiple utilisée pour estimer une sortie en fonction de plusieurs entrées n'est qu'une extension de la précédente, on a donc :

$$Y_i = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_n X_{ni} + \epsilon_i \tag{4.7}$$

Pour résoudre une telle équation, on utilise la représentation matricielle :

$$[Y] = \beta[X] + \epsilon \tag{4.8}$$

Où $[Y]$ et $[X]$ sont les matrices respectivement $1 \times n$ de sortie et $N \times n$ d'entrée d'entraînement, β est la matrice $1 \times N$ de coefficients et ϵ la matrice $1 \times n$ d'erreurs d'estimation, rappelons que n est le nombre d'exemples et N est le nombre d'attributs. Le calcul de la SSE (Error Sum of Squares) peut se faire de la même manière :

En dérivant et annulant on obtient :

$$\beta = ([X]' \cdot [X])^{-1}([X]' \cdot [Y]) \tag{4.9}$$

Il est clair que les tailles des matrices $[X]$ et $[Y]$ dépendent du nombre de données d'entraînement, si n est dans l'ordre de quelques centaines le problème peut être traité, mais si n atteint plusieurs millions, le calcul de la matrice β devient pénible et des solutions d'optimisation doivent être recherchés. Les cas de régression non linéaire sont généralement transformés en des problèmes de régression linéaire à l'aide de transformation de variables. Par exemple, le problème non linéaire suivant :

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_3 + \beta_4 X_2 X_3 \tag{4.10}$$

peut être transformé en un modèle linéaire en posons des nouvelles variables $X_4 = X_1 X_3$ et $X_5 = X_2 X_3$. La régression peut être même utilisée pour la classification c'est-à-dire l'estimation des variables catégorielles, mais les binaires seulement (deux classe) en estimant la probabilité d'appartenance à l'une des deux classes. Ceci peut être fait en utilisant une fonction linéaire se basant sur les variables d'entrée et qui fournit une probabilité. Ce type de régression s'appelle la régression logistique.

4.4 Régression par arbres de décision

Les arbres peuvent être utilisées pour résoudre le problème de régression, elles sont appelées dans ce cas les arbres de régression. L'algorithme CART (Classification And Regression Tree) en est un exemple. Son idée est de construire un arbre binaire où les nœds internes représentent les variables de prédiction et les feuilles représentent les valeurs continue de la classe à prédire. La figure suivante représente un exemple d'un arbre de régression.

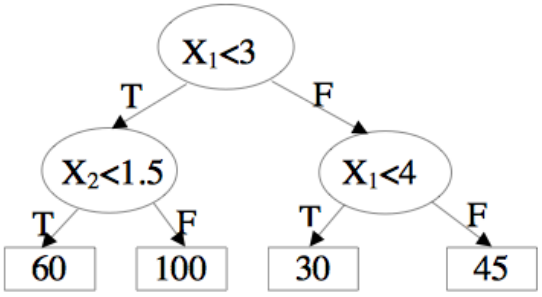


FIGURE 4.2 – Exemple d'un arbre de régression

Pour la construction d'un arbre de décision on a besoin de répondre aux questions suivantes :

- Comment choisir la variable de subdivision ?
- Quel est le critère d'arrêt ?

4.4.1 Choix de la variable de subdivision

Généralement, on choisit la subdivision, au sens de la variable et son partitionnement, qui minimise l'erreur quadratique (*WSS : within sum of squares*) de prédiction donnée par la formule suivante.

$$WSS = \sum_{j=1}^{g} \sum_{i=1}^{n_j} (y_{ij} - \overline{y}_j)^2 \tag{4.11}$$

Où :

- g :le nombre de groupes résultant de la subdivision choisie (la variable et le nombre de ses partitions)
- n_j : nombre d'exemples dans le groupe j
- y_{ij} : valeur de la classe de l'exemple numéro i dans le groupe numéro j
- $\overline{y}_j$: la moyenne de la variable de la classe du groupe j

g est le L'algorithme CART considère des variables de deux partitions uniquement c'est à dire $g = 2$.

4.4.2 Critère d'arrêt

Ce critère représente la condition d'arrêt du développement d'une branche. On trouve deux approche :

1. Pré-élagage : On fixe à l'avance des conditions sur la continuité de développement

d'une branche tel qu'un minimum d'exemples, un maximum de profondeur, une variance minimale, ...etc.

2. Post-élagage : c'est l'approche la plus utilisée. On construit une arbre maximum puis on le réduit à un arbre minimisant l'erreur de prédiction.

4.4.3 Algorithme de construction d'un arbre de régression

Un algorithme de construction d'un arbre de régression suit généralement les étapes suivantes :

Algorithme 7 Arbre-Régression(D : ensemble de données)

Créer nœud N
Si tous les exemples de D ont de la même valeur de la classe y alors
    Retourner N comme une feuille étiquetée par N ;
Si la liste des attributs est vide alors
  Retourner N Comme une feuille étiquetée de la classe de la moyenne dans $\overline{y}$;
Chercher la meilleure subdivision (parmi toutes les variables) qui minimise WSS
Etiqueter N par la variable sélectionnée ;
Liste de variables ← Liste d'attributs - la variable sélectionnée ;
Pour chaque intervalle Int_i (partition) de la variable sélectionnée
   Soit D_i l'ensemble d'exemples de D appartenant à Int_i ;
   Attacher à N le sous arbre généré par l'appel Arbre-Régression(D_i)
FinPour ;
Fin ;

La subdivision des variables dépend de leur type. Pour les variables continues, on peur partitionner par fréquence ou par intervalles et pour les variables discrètes, on peut partitionner selon toutes les valeurs disponibles ou selon des ensembles de ces valeurs.

4.5 SVM pour la régression (SVR)

Dans leur origine, les SVMs ont été développées pour des problèmes de classification. Cependant, leur nature leur permet de résoudre également des problèmes de régression. La régression est un cas particulier de classification où les classes des exemples ne sont pas dénombrables c'est à dire continues.

Pour résoudre le problème de régression, SVM utilise une astuce semblable a celle utilisée en classification. On propose de modéliser la fonction de régression par un hyperplan qui se situe au centre d'un hyper-tube de largeur 2ϵ contenant tous les exemples d'entrainement (cf. Figure 4.3.a).

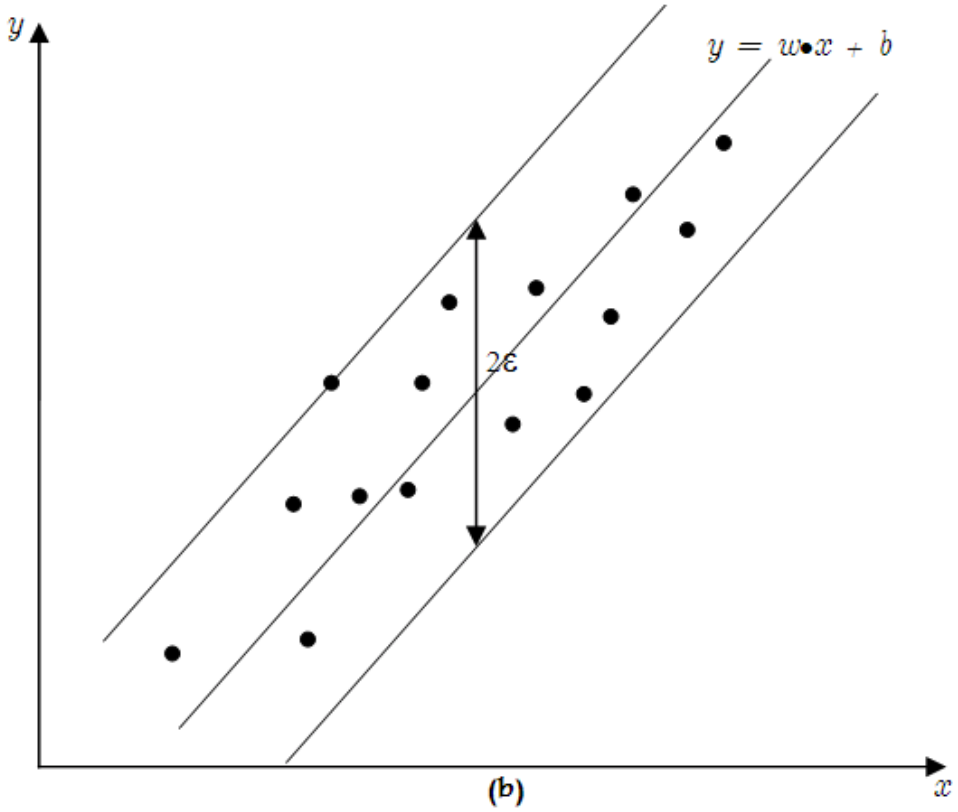


FIGURE 4.3 – Hyper-tube modélisant la fonction de régression

Plusieurs hyperp-tubes, de largeur 2ϵ contenant tous les exemples d'entrainement, peuvent exister. L'hyper-tube optimum est celui qui minimise la distances entre les exemples d'entrainement et ses frontières, autrement dit, qui maximise la distance des exemples de l'hyperplan du centre (cf. Figure 4.3.b). La détermination de l'hyper-tube optimum est semblable à la détermination de l'hyperplan de marge maximale optimum dans le cas de classification. On doit donc rechercher un hyper-tube de marge maximale avec tous les exemples d'entrainement à l'intérieur. En d'autre terme, ajuster l'hyper-tube par rotation et décalage jusqu'à maximiser la distance des exemples de l'hyperplan du centre, tout en gardant tous les exemples à l'intérieur de l'hyper-tube. L'équation 4.12 modélise le problème sous forme d'un problème de programmation quadratique. La fonction objective

maximise la marge et les contraintes gardent les exemples dans l'hyper-tube de largeur 2ϵ.

$$\begin{cases} Minimiser_{w,b} & \frac{1}{2}\langle w, w\rangle \\ sous\ contraintes & \\ & \left|y_i - \hat{f}(x_i)\right| \leq \epsilon; \quad \forall \ \ i = 1..n \end{cases} \tag{4.12}$$

En pratique, où il est difficile de garder tous les exemples dans un hyper-tube de largeur 2ϵ, on relaxe un peut les contraintes comme dans le cas des SVM à marge souple. On admet alors à l'hyper-tube de laisser des exemples à l'extérieur en introduisant des variables de relaxation ξ (cf. Figure 4.4). Les variables de relaxation ξ_i représentent les erreurs des

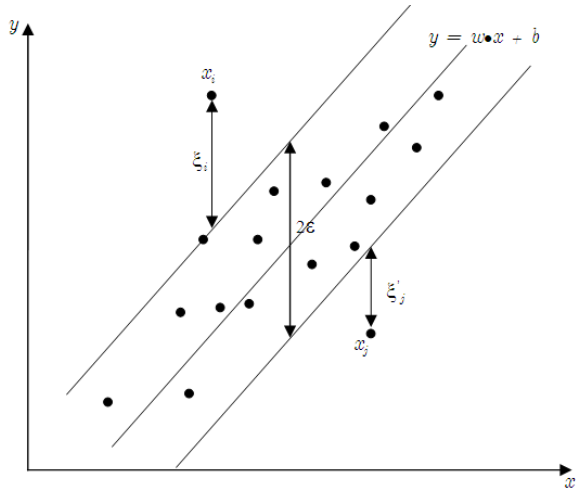


FIGURE 4.4 – Régression à marge maximale avec des variables de relaxation

exemples au dessus de l'hyper-tube tandis que les ξ_i' représentent les erreurs des exemples au dessous. Toutes ces variables sont nulles pour les exemples à l'intérieur de l'hyper-tube et égales aux distances de l'hyper-tube pour les exemples à l'extérieur (équations 4.13,4.14).

$$\xi_i = \begin{cases} 0 & si \ \ y_i - \hat{f}(x_i) \leq \epsilon \\ \left|y_i - \hat{f}(x_i)\right| - \epsilon & sinon \end{cases} \tag{4.13}$$

$$\xi_i' = \begin{cases} 0 & si \ \ \hat{f}(x_i) - y_i \leq \epsilon \\ \left|y_i - \hat{f}(x_i)\right| - \epsilon & sinon \end{cases} \tag{4.14}$$

Le problème de détermination de l'hyperplan devient donc un compromis entre la maximisation de la marge et la minimisation des erreurs ξ_i. En ajoutant les variables de relaxation, le problème de l'équation 4.12 devient :

$$\begin{cases} \textit{Minimiser}_{w,b,\xi} & \frac{1}{2}\langle w,w\rangle + C\sum_{i=1}^{n}(\xi_i+\xi_i') \\ \textit{sous contraintes} & \\ & y_i - \langle w,x_i\rangle - b \leq \epsilon + \xi_i \\ & \langle w,x_i\rangle + b - y_i \leq \epsilon + \xi_i' \\ & \xi_i,\xi_i' \geq 0; \quad \forall \;\; i = 1..n \end{cases} \tag{4.15}$$

Pour résoudre le problème de l'équation 4.15, on utilise la méthode de Lagrange pour passer au problème dual de l'équation 4.16 introduisant les multiplicateurs de Lagrange α_i et α_i'.

$$\begin{cases} \textit{Maximiser}_{\alpha,\alpha'} & Q = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i+\xi_i') \\ & -\sum_{i=1}^{n}\alpha_i(\epsilon+\xi_i - y_i + \langle w,x_i\rangle + b) \\ & -\sum_{i=1}^{n}\alpha_i'(\epsilon+\xi_i + y_i - \langle w,x_i\rangle - b) \\ & -\sum_{i=1}^{n}(\beta_i\xi_i + \beta_i'\xi_i') \\ \textit{avec} & \\ & \alpha_i,\alpha_i',\beta_i,\beta_i' \geq 0; \quad \forall \;\; i = 1..n \end{cases} \tag{4.16}$$

A la solution optimale, la dérivée de Q par rapport aux variables primales (w,b,ξ,ξ') s'annule.

$$\begin{cases} \frac{\partial Q}{\partial w} = w - \sum_{i=1}^{n}(\alpha_i' - \alpha_i)x_i = 0 & (a) \\ \frac{\partial Q}{\partial b} = \sum_{i=1}^{n}(\alpha_i' - \alpha_i) = 0 & (b) \\ \frac{\partial Q}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 & (c) \\ \frac{\partial Q}{\partial \xi_i'} = C - \alpha_i' - \beta_i' = 0 & (d) \end{cases} \tag{4.17}$$

En substituant 4.17.a,b,c,d dans l'équation 4.16, on obtient le problème dual de l'équation 4.18.

$$\begin{cases} \textit{Maximiser}_{\alpha,\alpha'} & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')\langle x_i,x_j\rangle \\ & -\epsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i') + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i') \\ \textit{sous contraintes} & \\ & \sum_{i=1}^{n}(\alpha_i - \alpha_i') = 0 \\ & 0 \leq \alpha_i,\alpha_i' \leq C \end{cases} \tag{4.18}$$

La résolution du problème de l'équation 4.18 consiste à trouver les α_i et les α_i'. La fonction de décision $\hat{f}(x)$ peut être donnée par l'équation 4.19.

$$\hat{f}(x) = \sum_{i=1}^{n} (\alpha_i - \alpha_i') \langle x_i, x \rangle + b \tag{4.19}$$

Où b peut être calculé en utilisant les conditions KKT qui montrent que le produit entre les variables duales et les contraintes s'annule à la solution optimale. Cela signifie que les α_i et les α_i' sont nuls uniquement pour tous les exemples à l'intérieur de l'hyper-tube. Les α_i et les α_i' non nuls correspondent aux exemples qui se trouvent aux frontières ou à l'extérieur de l'hyper-tube c-à-d les vecteurs supports (équation 4.20).

$$\begin{aligned} &\alpha_i(\epsilon + \xi_i - y_i + \langle w, x_i \rangle + b) = 0 \\ &\alpha_i'(\epsilon + \xi_i + y_i - \langle w, x_i \rangle - b) = 0 \\ &(C - \alpha_i)\xi_i = 0 \\ &(C - \alpha_i')\xi_i' = 0 \end{aligned} \tag{4.20}$$

b est calculé, donc, à partir d'un exemple dont $0 < \alpha_i < 0$ (vecteur support) par l'équation 4.21.

$$b = y_i - \langle w, x_i \rangle - \epsilon \tag{4.21}$$

4.5.1 Utilisation des noyaux

Parmi les motivations les plus solides du développement des machines à vecteurs supports pour la régression est leur extension simple aux cas non linéaire grâce à l'utilisation des noyaux. En effet, d'une manière similaire au cas de classification, on fait une transformation d'espace ϕ pour se trouver toujours face à une régression linéaire. La transformation d'espace inverse ϕ^{-1}, permet de retourner à l'espace d'origine après la résolution dans le nouvel espace (cf Figure 4.5).

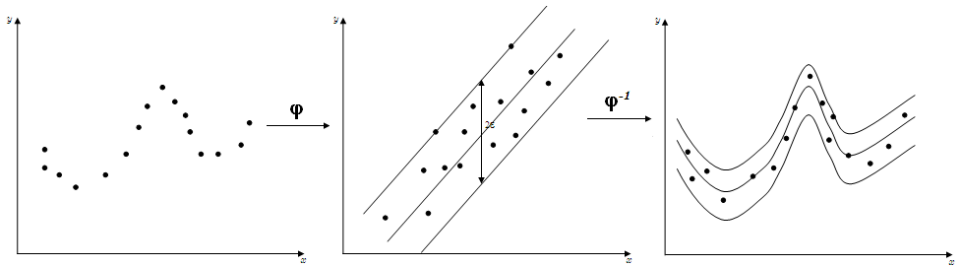


FIGURE 4.5 – Utilisation des noyaux pour la résolution de la régression non linéaire

Comme dans le cas de classification, la transformation ϕ et son inverse est réalisé grâce à une fonction réelle $K(x_i, x_j)$ appelée *Noyau (Kernel)*. Le problème de l'équation 4.18 devient alors :

$$\begin{cases} \textit{Maximiser}_{\alpha,\alpha'} & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i-\alpha'_i)(\alpha_j-\alpha'_j)K(x_i,x_j)-\epsilon\sum_{i=1}^{n}(\alpha_i+\alpha'_i)+\sum_{i=1}^{n}y_i(\alpha_i-\alpha'_i) \\ \textit{sous contraintes} & \\ & \sum_{i=1}^{n}(\alpha_i-\alpha'_i)=0 \\ & 0\leq\alpha_i,\alpha'_i\leq C \end{cases} \tag{4.22}$$

Et la fonction de décision devient :

$$\hat{f}(x)=\sum_{i=1}^{n}(\alpha_i-\alpha'_i)K(x_i,x)+b \tag{4.23}$$

En pratique, le paramètre ϵ est difficile à choisir et il est plus judicieux de le calculer automatiquement. La variante v-SVM introduit ϵ dans les paramètres à optimiser dans la fonction objective du problème, et pour contrôler le taux d'erreur, on spécifie au lieu de ϵ une limite $0<v<1$ sur la fraction des exemples qui peuvent être à l'extérieur de l'hyper-tube. Cela peut être atteint en modifiant la fonction objective de l'équation 4.15 comme suit :

$$\textit{Minimiser}_{w,b,\xi,\epsilon}\quad \frac{1}{2}\langle w,w\rangle+C(vn\epsilon\sum_{i=1}^{n}(\xi_i+\xi'_i)) \tag{4.24}$$

Chapitre 5

Clustering

5.1 Introduction et rappels

Le clustering regroupe un ensemble de techniques qui visent à regrouper les enregistrements d'une base de données en des groupes selon leur rapprochement les uns des autres en ne se basant sur aucune information antérieure, c'est un apprentissage non supervisé. Un système d'analyse en clusters prend en entrée un ensemble de données et une mesure de similarité entre ces données, et produit en sortie un ensemble de partitions décrivant la structure générale de l'ensemble de données. Plus formellement, un système de clustering prend un tuplet (D, s) où D représente l'ensemble de données et s la mesure de similarité, et retourne une partition $P = (G_1, G_2, ..., G_m)$ tel que les $Gi(i = 1..m)$ sont des sous ensembles de D qui vérifient :

$$\begin{cases} G_1 \cup G_2 \cup .. \cup G_m = D \\ G_i \cap G_j = \Phi, \quad i \neq j \end{cases} \tag{5.1}$$

Chaque G_i est appelé un cluster qui représente une ou plusieurs caractéristiques de l'ensemble D, mais les choses ne sont pas ci simples, on a beaucoup de problèmes à traiter. Premièrement, le nombre de clusters à chercher. Dans certains cas c'est un expert du domaine d'application qui fournit le nombre de clusters qui forment l'ensemble de données. Mais dans la plupart des cas, même l'expert ne connaît pas le nombre de clusters et cherche à le savoir et le comprendre. Dans la majorité des cas, on définit une mesure de stabilité du processus d'analyse à base de laquelle on peut atteindre le meilleur nombre de clusters qui décrit mieux les données.

5.2 Mesures de similarités

Une bonne méthode de clustering est une méthode qui maximise la ressemblance entre les données à l'intérieur de chaque cluster, et minimise la ressemblance entre les données des clusters différents. C'est pourquoi les résultats d'une technique de clustering dépendent fortement de la mesure de similarité choisie par son concepteur, qui doit la choisir avec prudence. En effet la mesure de similarité repose sur le calcul de la distance entre deux données, sachant que chaque donnée est composée d'un ensemble d'attributs numériques et/ou catégoriels. Plus la distance est importante, moins similaires sont les données et vice versa. Soit x_i et x_j deux données différentes dont on veut calculer la distance. Cette distance est composée d'une part de la distance entre les valeurs des attributs numériques et d'une autre part de la distance entre les valeurs des attributs catégoriels ou symboliques, en prenant bien sur en considération le poids (le nombre) de chaque type d'attributs.

5.2.1 Attributs numériques

Pour mesurer la distance entre les données à attributs numériques, plusieurs formules existent :

– La distance Euclidienne :

$$D_n(x_i, x_j) = \sqrt{\sum_{k=1}^{n_n} (x_{ik} - x_{jk})^2} \tag{5.2}$$

– La distance City blocs :

$$D_n(x_i, x_j) = \sum_{k=1}^{n_n} |x_{ik} - x_{jk}| \tag{5.3}$$

– La distance de Minkowksi :

$$D_{np}(x_i, x_j) = \left(\sum_{k=1}^{n_n} (x_{ik} - x_{jk})^2 \right)^2 \tag{5.4}$$

Il faut faire attention lors du calcul de ces distances à la normalisation des attributs, puisque les intervalles de variances des attributs peuvent être très différents, ce qui peut entraîner la dominance d'un ou de quelques attributs sur le résultat. Il est conseillé donc, de normaliser tous les attributs sur le même intervalle puis calculer la distance.

5.2.2 Attributs catégoriels

Le problème qui se pose lors du calcul de la distance entre les attributs catégoriels, c'est qu'on ne dispose pas d'une mesure de différence. La seule mesure qui existe, dans

l'absence de toute information sur la signification des valeurs, c'est l'égalité ou l'inégalité. La distance utilisée est alors :

$$\begin{cases} D_c(x_i, x_j) = \frac{1}{n_c} \sum_{k=1}^{n_c} f(x_{ik}, x_{jk}) \\ f(x_{ik}, x_{jk}) = 1 \quad si \quad x_{ik} = x_{jk}, \quad 0 \quad sinon \end{cases} \tag{5.5}$$

Il faut en fin la normaliser avec les attributs numériques et le nombre d'attributs catégoriels.

La distance entre deux données x_i et x_j, composées d'attributs numériques et catégoriels, est donc :

$$D(x_i, x_j) = D_n(x_i, x_j) + D_c(x_i, x_j) \tag{5.6}$$

En se basant sur la distance entre deux attributs, plusieurs distances peuvent être calculées :

- Distance entre deux clusters : permet de mesurer la distance entre deux clusters pour une éventuelle fusion en cas où ils soient trop proches. Cette distance peut être prise entres les centres des deux clusters, entre les deux données les plus éloignées (ou plus proches) des deux clusters ou la distance moyenne de leurs données.
- Distance intracluster : c'est la distance moyenne entre les données à l'intérieur d'un cluster, elle peut être utile pour maintenir un seuil d'éloignement maximum dans le cluster au dessus duquel on doit scinder ce cluster.
- Distance intercluster : c'est la distance moyenne entre les clusters, elle permet de mesurer l'éloignement moyen entre les différents clusters.
- Distance intraclusters moyenne : permet avec la distance interclusters de mesurer la qualité du clustering.

La mesure de similarité peut être utilisée par un algorithme de clustring pour trouver le partitionnement optimal des données. Parmi ces algorithmes on peut citer :

5.3 Clustering hiérarchique

Dans ce type de clustering le nombre de clusters ne peut être connu à l'avance. Le système prend en entrée l'ensemble de données et fournit en sortie une arborescence de clusters. Il existe deux classe de ce type d'algorithmes : les algorithmes divisibles qui commencent à partir d'un ensemble de données et le subdivisent en sous ensembles puis subdivisent chaque sous ensemble en d'autres plus petits, et ainsi de suite, pour générer en fin une séquence de clusters ordonnée du plus général au plus fin. La deuxième classe est celle des algorithmes agglomiratifs qui considèrent chaque enregistrement comme étant un

cluster indépendant puis rassemblent les plus proches en des clusters plus importants, et ainsi de suite jusqu'à atteindre un seul cluster contenant toutes les données. Un algorithme agglomiratif suit généralement les étapes suivantes :

Algorithme 8 Clustering hiérarchique

1: Placer chaque enregistrement dans son propre cluster ;
2: Calculer une liste des distances interclusters et la trier dans l'ordre croissant ;
3: Pour chaque seuil de niveau de similitude préfixé d_k
4: Relier tous les clusters dont la distance est inférieure à d_k par des arrêtes à un nouveau cluster ;
5: FPour
6: Si tous les enregistrements sont membres d'un graphe connecté alors fin sinon aller à 3 ;
7: le résultat est un graphe qui peut être coupé selon le niveau de similarité désiré ;

Exemple : Soient les données suivantes : $X_1(0,2), X_2(0,0), X_3(1.5,0), X_4(5,0), X_5(5,2)$

On utilise la distance euclidienne pour mesurer la distance entre les données :

$D(X_1, X_2) = \sqrt{(0-0)\times 2 + (2-0)\times 2} = 2,$

$D(X_1, X_3) = 2.5,$

:

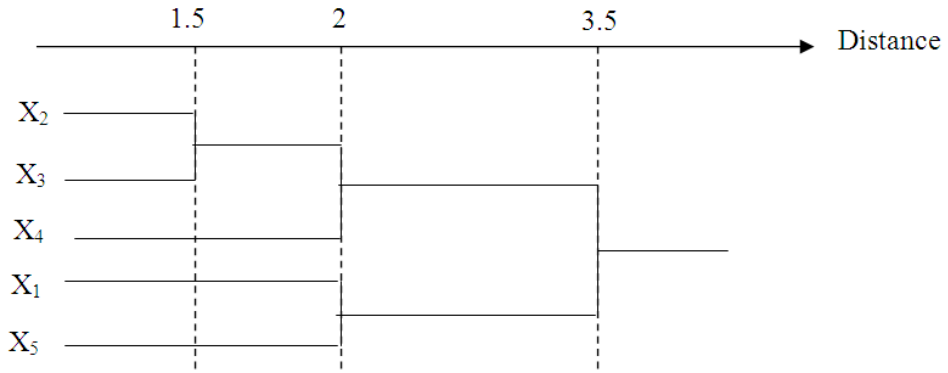
$D(X_2, X_3) = 1.5,$

:

On trie ces distances, on trouve que X_2 et X_3 sont les plus proches, on les rassemble dans le même cluster et on calcule son centre (0.75, 0),

On refait le calcul des distances en remplaçant X_2, X_3 par le nouveau cluster, on trie puis on choisit la plus courte distance et ainsi de suite.

On obtient à la fin le dendrogramme suivant représentant le rassemblement hiérarchique des données :



La complexité de cet algorithme est $O(N^2 Log(n))$, puisqu'il traite les données en paires pour le tri et la création des partitions ce qui le rend un peu lourd vis-à-vis les grandes bases de données. Son avantage est qu'il produit une vue en plusieurs niveaux des données. Il est aussi indépendant de l'ordre des données introduites.

5.4 Clustering partitionnel

Un algorithme partitionnel de clustring obtient à sa fin une seule partition des données au lieu de plusieurs tel que dans le cas précédent. Pour ce faire, il essaye d'optimiser la distance moyenne entre les données de chaque cluster et son centre, on utilise généralement l'erreur quadratique qui calcule l'erreur entre les données et le centre :

$$e_k^2 = \sum_{i=1}^{N_k} (x_{ik} - \overline{x_k})^2 \tag{5.7}$$

Où e_k^2 est l'erreur quadratique moyenne du cluster k, N_k est le nombre d'enregistrements dans le cluster, x_{ik} est l'enregistrement numéro i du cluster k et $\overline{x_k}$ est la moyenne du cluster k. L'objectif d'un algorithme partitionnel est de trouver une partition qui minimise E_k^2 tel que :

$$e_K^2 = \sum_{k=1}^{K} e_k^2 \tag{5.8}$$

Le plus simple et le plus utilisé des algorithmes partitionnels est l'algorithme K-means, il démarre d'une partition arbitraire des enregistrements sur les k clusters, à chaque itération il calcule les centres de ces clusters puis il effectue une nouvelle affectation des enregistrements aux plus proches centres. Il s'arrête dès qu'un critère d'arrêt est satisfait, généralement on s'arrête si aucun enregistrement ne change de cluster. L'algorithme est le suivant :

Algorithme 9 Clustering partitionnel

1: Sélectionner une partition initiale contenant des enregistrements choisis arbitrairement, puis calculer les centres des clusters ;
2: Calculer une liste des distances interclusters et la trier dans l'ordre croissant ;
3: Générer une nouvelle partition en affectant chaque enregistrement au cluster du centre le plus proche ;
4: Calculer les nouveaux centres des clusters ;
5: Répéter 2 et 3 jusqu'à ce que les enregistrements se stabilisent dans leurs clusters ;

Exemple : Prenons le même ensemble de données précédent :

$X_1(0,2), X_2(0,0), X_3(1.5,0), X_4(5,0), X_5(5,2),$

On commence par choisir une affectation arbitraire des données

$C_1 = X_1, X_2, X_4$ et $C_2 = X_3, X_5$.

On calcule les deux centres :

$M_1 = ((0+0+5)/3,\ (2+0+0)/3) = (1.66, 0.66)$

$M_2 = ((1.5+5)/2,\ (0+2)/2) = (3.25, 1)$

On calcule la distance entre chaque donnée Xi et les centres M_1 et M_2, puis on affecte chaque données au cluster le plus proche.

La nouvelle affectation est $C_1 = \{X_1, X_2, X_3\}$, $C_2 = \{X_4, X_5\}$.

On recalcule les nouveaux centres, puis on réaffecte jusqu'à ce qu'aucune donnée ne change de cluster.

Cet algorithme est le plus utilisé pour le clustering des bases de données immenses. Sa complexité est $O(nlk)$ où n est le nombre d'enregistrements, l est le nombre d'itérations de l'algorithme, et k est le nombre de clusters. Pratiquement, k et l sont fixés à l'avance, ce qui veut dire que l'algorithme est linéaire par rapport aux données. Il est aussi indépendant de l'ordre des données introduites.

5.5 Clustering incrémental

Les deux algorithmes présentés ci-dessus nécessitent la présence de tout l'ensemble de données analysées en mémoire, ce qui est pratiquement avec de larges bases de données avec des millions d'enregistrements. Pour palier ce problème le clustering incrémental traite une partie des données (selon la mémoire disponible) puis ajoute itérativement des données en modifiant chaque fois si nécessaire le partitionnement obtenu. L'algorithme suivant résume les étapes suivies :

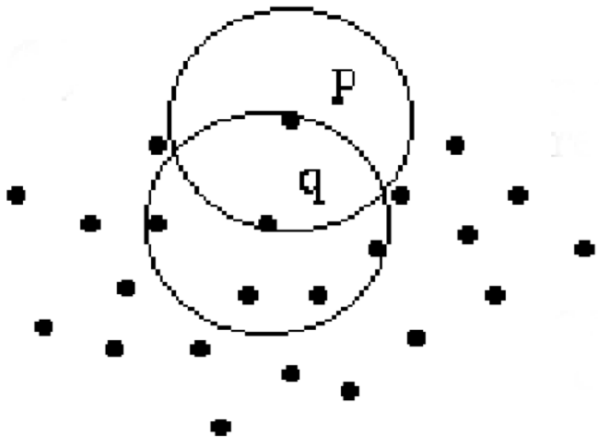
Algorithme 10 Clustering incrémental

1: Affecter le premier enregistrement au premier cluster ;
2: Pour un nouvel enregistrement : soit l'affecter un un cluster existant, soit l'affecter à un nouveau cluster. Cette affectation est effectué selon un certain critère. Par exemple la distance du nouvel enregistrement des centres des clusters existants. Dans ce cas à chaque affectation les centres des clusters sont recalculés ;
3: Répéter l'étape 2 jusqu'à ce que tous les enregistrements soient clusterés ;

Le besoin en terme de mémoire des algorithmes de clustering incrémental est très réduit, ils nécessitent généralement uniquement les centres des clusters. Un problème sérieux duquel souffrent les algorithmes de clustering incrémental est leur dépendance de l'ordre de présentation des enregistrements. Avec les mêmes données ordonnées différemment, ils fournissent des résultats totalement différents.

5.6 Clustering basé densité

Ce type de clustering se base sur l'utilisation de la densité à la place de la distance. On dit qu'n point est dense si le nombre de ses voisins dépasse un certain seuil. Un point est voisin d'un autre point s'il est à une distance inférieure à une valeur fixée. Dans la figure suivante q est dense mais pas p :



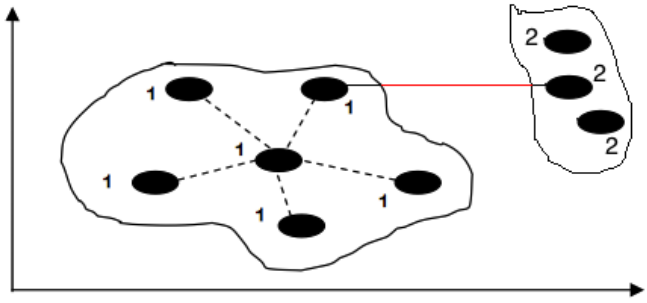
L'algorithme DBSCAN (Density-Based Spatial Clustering of Applications with Noise) est un exemple des algorithmes à base de densité.

Il utilise deux paramètres : la distance ϵ et le nombre minimum de points MinPts devant se trouver dans un rayon ϵ pour que ces points soient considérés comme un cluster. Les

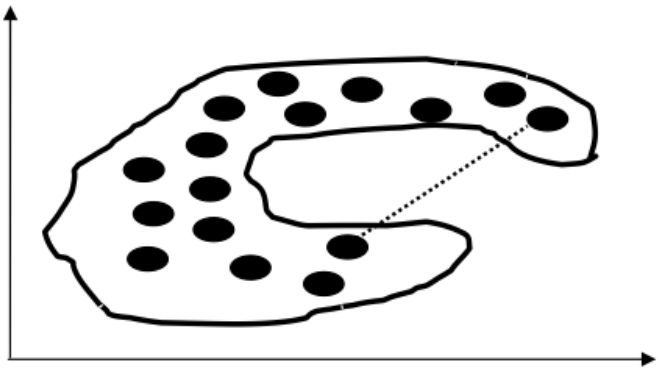
paramètres d'entrées sont donc une estimation de la densité de points des clusters. L'idée de base de l'algorithme est ensuite, pour un point donné, de récupérer son ϵ-voisinage et de vérifier qu'il contient bien MinPts points ou plus. Ce point est alors considéré comme faisant partie d'un cluster. On parcourt ensuite l'ϵ-voisinage de proche en proche afin de trouver l'ensemble des points du cluster.

5.7 Support vector clustering

Le SVC (Support vector clustering) est une méthode de clustering dérivée de la méthode des machines à vecteurs supports.Elle utilise la méthode SVM mono-classe pour regrouper les données dans une zone connue par la fonction de décision. Pour détecter les clusters (c-à-d étiqueter les exemples), un segment, est tracé entre chaque paire d'exemples dans l'espace d'origine, et ses points sont examinés par la fonction de décision ; si tous les points du segment appartiennent à la zone positive, les deux exemples sont considérés appartenant au même cluster.



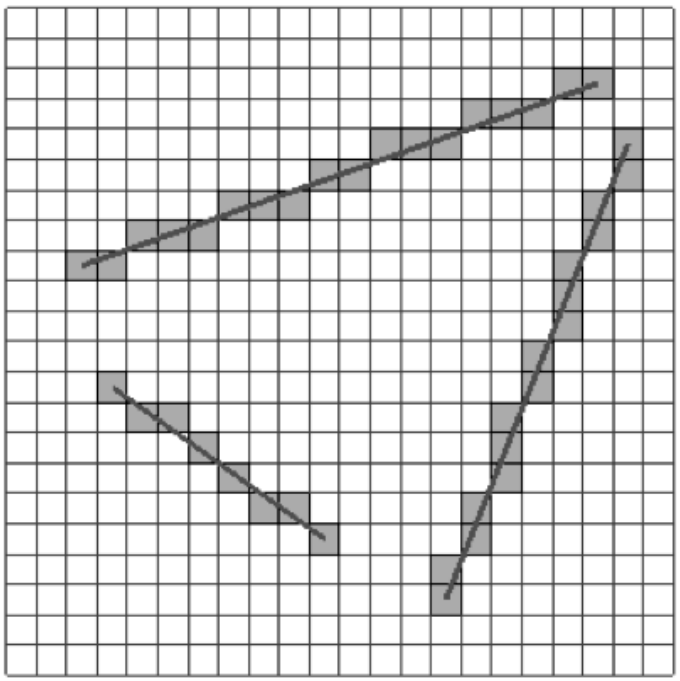
Le problème qu'on peut rencontrer dans cette solution c'est le cas où des points du segment reliant deux exemples du même cluster retournent des valeurs négatives par la fonction de décision mono-classe comme illustré dans la figure suivante :



La solution à ce problème est de construire un graphe non orienté, où les nœuds sont les

exemples et les arrêtes sont les segments de points positifs, puis rechercher les composantes connexes. Chaque composante connexe représente un cluster.

Le tracé de segment est largement étudié dans la littérature pour le traçage des lignes sur les écrans. Parmi les algorithmes qui ont prouvé leurs efficacités on trouve l'algorithme de Bresenham, qui prend en entrée un point de départ et un point d'arrivée et fournit l'ensemble des points du segment reliant ces deux points. L'espace est supposé discret c-à-d un écran formé de pixels.

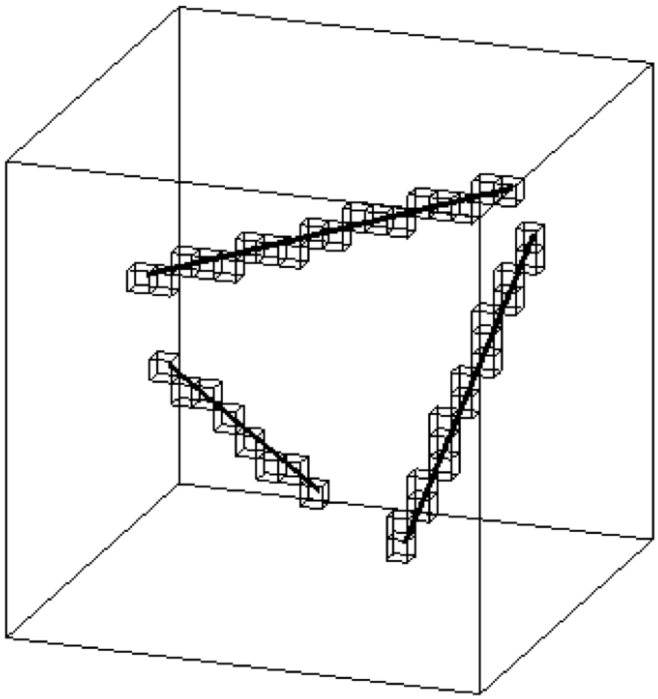


Le pseudo algorithme suivant en illustre le principe.

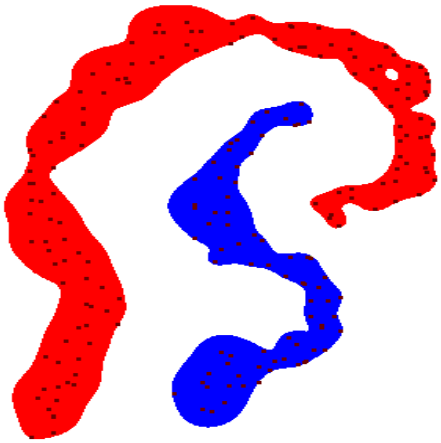
Algorithme 11 Algorithme de Bresenham

1: **procedure** SEGMENT(x, y)
2: Déterminer le sens dominant de la ligne ;
3: **si** La pente est inférieure à 45° **alors** Choisir l'axe X
4: **sinon** Choisir l'axe Y
5: **fin si**
6: Prendre un pas dans la direction dominante ;
7: **si** Une étape est nécessaire dans le sens secondaire **alors** Prendre cette étape
8: **fin si**
9: **si** Point destinataire non atteint **alors** Aller vers l'étape 6
10: **fin si**
11: **fin procedure**

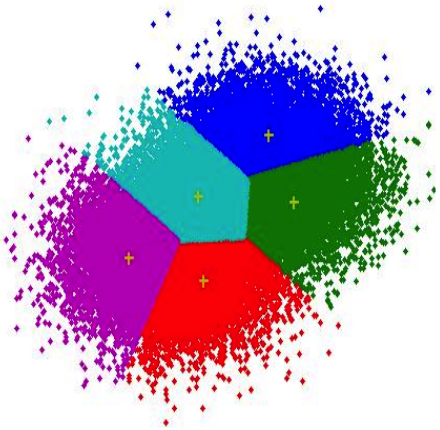
La version 3D de l'algorithme de Bresenham est utilisée en synthèse d'image pour calculer les segments dans des scènes 3D. Cette version consiste à choisir le sens dominant parmi trois axes X,Y et Z au lieu de deux. Au niveau de l'axe dominant on fait appel à l'algorithme 2D sur les deux axes restants. La figure suivante en illustre le principe de segmentation :



Dans la pratique, les données à clusterer possèdent plusieurs caractéristiques voire des milliers, et l'utilisation de la méthode SVC necéssite l'utilisation des tracés de segments à n dimensions ($n > 3$). Dans ce cas, le tracé de segments devient plus compliqué et nécessite des temps de calcul énormes. La méthode SVC est très précise comparant aux méthodes classiques de clustering telle que k-means. SVC permet de traiter des cas très compliqués d'interférence des clusters grâce à l'utilisation du noyau RBF à l'inverse des méthodes basée sur les distances et les centres. La figure suivante illustre les capacités de séparation des clusters des méthodes SVC et k-means.



SVC



Kmeans