Université de Jijel

Faculté des Sciences Exactes et de l'Informatique

Département Informatique

INITIATION A L'APPRENTISSAGE AUTOMATIQUE

Support de Cours pour étudiants en Master en Intelligence Artificielle

Mokhtar TAFFAR

Enseignant-Chercheur

Université de Jijel

INITIATION A L'APPRENTISSAGE AUTOMATIQUE

Avant-propos

Les algorithmes intelligents, permettant aux systèmes à la fois d'apprendre et de raisonner, particulièrement ceux destinés à l'analyse d'image et vidéo et à la vision par ordinateur, et plus généralement à l'intelligence artificielle (IA), font, depuis quelques décennies, l'objet d'intense recherches, tant en théorique qu'en pratique.

Ce cours est destiné aux étudiants préparant un master en intelligence artificielle et systèmes intelligents, notamment ceux ayant en vue la réalisation d'un projet de fin d'étude, ainsi qu'à toute personne désireuse de s'initier à la construction de modèles destinés à l'IA, à la vision par ordinateur et à l'analyse d'image et vidéo ; et même aux personnes travaillant sur les systèmes d'information et d'aide à la décision et les modèles de fouilles de données.

Nous souhaitons que cet ouvrage constitue, à sa juste valeur, une contribution effective offrant aux étudiants un recueil des notions intelligibles simplifiées sur l'écriture des algorithmes intelligents. Aussi, en présentant d'une manière sobre ces notions, il conduit aisément le lecteur à leur compréhension, en lui procurant ainsi une opportunité de maîtrise concrète de la construction de modèles automatique.

Table des matières

Chapitre 1

Introduction Générale

Chapitre 1 : Introduction Générale

1.1. Aperçu sur l'Etat Actuel

Un bref historique dans le domaine de l'apprentissage artificiel, aussi communément appelé apprentissage automatique -AA (ou Machine Learning, en anglais), nous amènent à parler des trois grandes époques de l'ordinateur, plus précisément, au tout début de l'informatique, de son évolution au fil du temps et enfin au monde d'aujourd'hui et de demain.

De nos jours, nous pouvons constater, et ce n'est qu'un point de vue, que l'évolution de l'informatique s'est faite principalement sur deux axes :

- Gain en capacité à cumuler de l'information et à sa diffusion dans des domaines tels que les fouilles de données (Data Mining), les entrepôts de données, les réseaux et services web, sans oublié leurs applications sous-jacentes sous Smartphones, d'une part, et
- Gain en intelligence des systèmes informatique, en particuliers, les domaines liés à l'intelligence artificielle lesquels sont les plus touchés par cette avancée technologique et comprennent particulièrement, les jeux, la parole, la vision par ordinateur, etc.

1.2. Le Nouveau Paradigme

Dans cette matière, nous nous intéressons aux théories, algorithmes et applications liés à un aspect particulier de l'intelligence artificielle (IA) : la faculté d'apprentissage.

De cette notion d'apprentissage, il devient facile de constater que le paradigme de programmation classique a évolué, ainsi,

- Avant, programmer consiste à prescrire une logique pour faire exécuter (une tâche).
- Maintenant, on programme pour rendre intelligent, autrement dit, programmer pour faire exécuter de manière intelligente des tâches nouvelles réputées nécessitant du raisonnement et un jugement.

Par un tel mécanisme de programmation, nous serons en mesure de faire doter un programme d'une aptitude d'apprentissage.

1.3. Origines de l'Apprentissage Automatique

La discipline de l'apprentissage automatique (AA) possède de riches fondements théoriques. On sait, désormais, répondre à des questions comme :

- Quelles méthodes d'apprentissage sont les plus efficaces pour tel ou tel type de problème ?
- Combien d'exemples d'entrainement faut-il fournir à un programme d'apprentissage pour être certain qu'il apprenne avec une efficacité donnée ?

Etant donnée la variété d'apprentissages qu'on peut rencontrer, il est aisé de deviner que les fondements de cette discipline, en occurrence l'apprentissage automatique, proviennent de diverses sciences :

- Des mathématiques pour l'informatique : algèbre linéaire, la probabilité, la logique, l'analyse élémentaire, …
- La théorie statistique de l'estimation,
- L'apprentissage Bayésien,
- L'inférence grammaticale ou l'apprentissage par renforcement, et tant d'autres.

1.4. Domaines de l'Apprentissage Automatique

Les principaux domaines d'applications de l'apprentissage automatique (AA) sont les fouilles de données et l'intelligence artificielle.

- La fouille de données (Data Mining, en anglais) est le processus d'extraction de la connaissance : il consiste à sélectionner les données à étudier à partir de bases de données (BDs) (hétérogènes ou homogènes), à épurer ces données et enfin à les utiliser en apprentissage pour construire un modèle.

Exemples,

- Trouver une prescription pour un malade (patient) à travers des fichiers médicaux antérieurs.
- Apprentissage de la reconnaissance de transactions frauduleuses par carte de crédit, par examen des transactions passées avérées frauduleuses.

- L'intelligence artificielle, la vision par ordinateur, la robotique, l'analyse et la compréhension des images, la reconnaissance de formes, reconnaître des objets dans les vidéo et extraire des contenus sémantiques des images sont autant d'applications qui requièrent la construction de modèles par apprentissage automatique. Un large ensemble d'éléments théorique et pratique, dans ces domaines, peut être retrouvé dans [Fu 74], [Mic84], [Mil 93], [Sim 84], [Wat 69].

Exemples,

- Systèmes de vidéo surveillance pour la détection des intrus.
- Logiciel biométrique de reconnaissance de visages et d'empreintes digitales.

1.5. Définitions et Particularités

Parmi les principales caractéristiques et facultés adoptées par les modèles d'apprentissage, nous citons : l'entrainement, la généralisation, l'adaptation, l'amélioration, l'intelligibilité et la prédiction.

1.5.1. Apprentissage

L'apprentissage, ou *Learning* en anglais, c'est le processus de **construire un modèle général** à partir de **données** (observations) **particulières** du monde réel.

Ainsi, le but est double :

- **Prédire** un comportement face à une nouvelle donnée.
- **Approximer** une fonction ou une densité de probabilité.

Deux (02) branches d'apprentissage existent :

- Apprentissage **symbolique**, issue de l'IA.
- Apprentissage **numérique**, issue des statistiques.

Dans la pratique, le mot **entraînement** (***Training***, en anglais) est souvent synonyme de : *apprentissage*. Ainsi, en science cognitive, l'apprentissage est définit comme étant la **capacité à améliorer les performances** au fur et à mesure de l'exercice d'une activité.

Exemple, un joueur de jeu d'échec : **assimile** (par expérience, s'entraîne) et **raisonne** (ceci lui procure une certaine intelligence ou puissance de raisonnement pour qu'il puisse progresser) –c'est le cas pour un algorithme intelligent.

1.5.2. Adaptation

Elle peut être vue comme étant la disposition du modèle (algorithme ou système) à corriger son comportement ou à remanier sa réponse (ex., prédiction) par rapport à de nouvelles situations.

Pour les tâches de perception, en vision artificielle, on accumule les **bonnes** et **mauvaises expériences**, et à partir d'elles, on peut faire **évoluer les règles** pour mieux effectuer la tâche, c'est le phénomène d'**adaptation** ou d'**amélioration**.

1.5.3. Dilemme de l'Apprentissage : Précision *Vs* Généralisation

- ***Précision*** : c'est **l'écart** entre une **valeur mesurée** ou **prédite** par le modèle et une **valeur réelle**.

- ***Généralisation*** : c'est la **capacité de reconnaître de nouveaux exemples** jamais vus auparavant.

NB : Souvent, un *seuil de généralisation* est utilisé et est propre à chaque modèle pendant l'apprentissage.

1.5.4. Intelligibilité

C'est amélioré la **compréhension des résultats d'apprentissage**, afin que le modèle puisse fournir une **connaissance claire** et **compréhensible**, au sens **interprétable** (en anglais, on parle de comprehensibility ou understandability).

Exemple, quand un expert extrait de la connaissance des bases de données (BDs), il *apprend une manière de les résumer* ou *de les formuler* (expliquer, expliciter de manière simple et précise).

D'un point de vue fouille de données, ça revient purement et simplement à contrôler l'intelligibilité (clarté) d'un modèle obtenu.

Actuellement, la **mesure d'intelligibilité** se réduit à vérifier que la connaissance produite est intelligible et que les **résultats** sont exprimés dans le **langage de l'utilisateur** et la **taille des modèles n'est pas excessive**.

1.5.5. Classification

En analyse de données, la classification consiste à **regrouper des ensembles d'exemples** (souvent, de manière non-supervisée) **en classes**. Ces classes sont généralement organisées en une structure : **clusters** (groupes ou grappes).

1.5.5.1. Classification

C'est le processus de reconnaissance en intention (par leurs propriétés) des classes décrites en extension (par les valeurs de leurs descripteurs).

Si les valeurs à prédire sont **des classes** en **petit nombre**, on parle alors de **classification**.

1.5.5.2. Régression

Elle traite des exemples où les **valeurs à prédire** sont **numériques**.

Exemple, le nombre d'exemplaires d'un ouvrage qui seront vendus = 4.000. Cette valeur peut être approchée au jour le jour, pendant que la vente continue sur une période.

1.6. Exemples d'Apprentissage Artificiel

Soit une distribution (dispersion) de deux types d'étudiants en IA, par rapport aux autres étudiants (étudiants IA et étudiants non-IA), mesurée sur deux caractéristiques majeures : la note en Algorithmique (Algo.) et la note en base de données (BD), et représentée par le graphe de la Fig. 1.1.

Problème : comment faire l'apprentissage ? Ça consiste en quoi ?

D'abord, c'est dire que la représentation visuelle (géométrique) est correcte par rapport au réel.

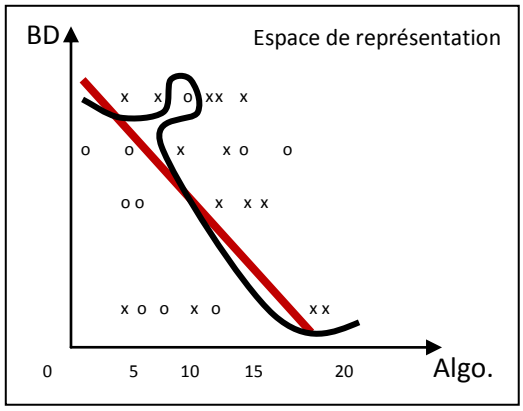


Figure 1.1. Répartition des étudiants dans l'espace visuel géométrique des notes.

Enoncé de l'apprentissage :

- **Trouver une règle** qui décide, dans l'espace de représentation choisi, **avec le moins d'erreurs possibles**, quel étudiant est de l'IA et quel étudiant ne l'est pas.
- Quelle sera cette règle ?

Règle : peut être une ligne (droite ou courbe) qui sépare les étudiants IA et non-IA.

- Si la règle de décision est une **droite** : le problème est *linéaire* (ex., équation de la forme $y=a.x+b$).

L'apprentissage sera de trouver la **meilleure droite**, en optimisant un critère étudié.

- Si la règle de décision est une **courbe** : le problème est alors *non-linéaire* (ex., polynôme, log, tangente,…).

Une règle de décision comme courbe sera plus **complexe**, elle sera avec **plus de restriction** (sur les notes des étudiants) pour l'apprentissage.

NB :

- La droite (linaire) choisie mène a une erreur mesurée par le nombre d'exemples mal classés par rapport au nombre d'exemples bien classés (ex., environ 4/20).
- La droite est souvent **moins précise** que la courbe mais **plus simple**.

Chapitre 2

Apprentissage et **C**lassification

Chapitre 2 : Apprentissage et Classification

2.1. Introduction

Le but de cet enseignement est de s'initier à l'apprentissage artificiel et à la modélisation de systèmes de classification ; mais, également, d'étudier et pouvoir, au final, concevoir des **algorithmes de prédiction**. De tels modèles seront capables de prédire la ***nature*** (forme, apparence, texture, couleur, …) et/ou le ***comportement*** (fonction, fonctionnement, variable, variations, …) de nouveaux exemples.

Cependant, la quantité de données récupérées par la technologie imagerie ou dans des fichiers BD dans divers domaines est très conséquente. L'analyse directe de ces données pourra donc s'avérer très lente ou trop complexe, voire impossible par des méthodes traditionnelles. Le but principal des techniques d'optimisation (méthode statistique, ACP, …) est de filtrer et réduire l'ensemble des données extraites ou acquises du monde réel afin de sélectionner et ne porter intérêt qu'à un sous-ensemble des variables les plus pertinentes pour le problème à traiter.

Exemple, soit un vecteur caractéristique $V = (v_1, v_2, \dots, x_n)$, $n \gg 0$, représentant les caractéristiques d'une classe objets de la nature (voitures, plantes, …), ainsi cet espace de dimension $\boldsymbol{n}$ est réduit en un espace de dimension $\boldsymbol{k}$, avec $\boldsymbol{k < n}$, qui est aussi **représentatif** et **discriminant** que celui de dimension n, mais plus facile à traiter.

2.2. Contexte Général

Une fois les données acquises du terrain ou du réel (images, vidéos, fichiers BD) sous forme de données brutes, il faut les analyser et en extraire de l'information sous forme de nouvelles connaissances, de descripteurs ou de vecteur de caractéristiques. Lorsque les données sont conséquentes leur traitement peut se faire par différentes méthodes en particulier le traitement statistique (dénombrement) lequel est un privilégié des disciplines de l'IA.

Les deux disciplines (intelligence artificielle –IA et fouille de données –FD) regroupent en commun différentes techniques de construction de modèles, elles sont récapitulées sur le schéma de la Figure 2.1.

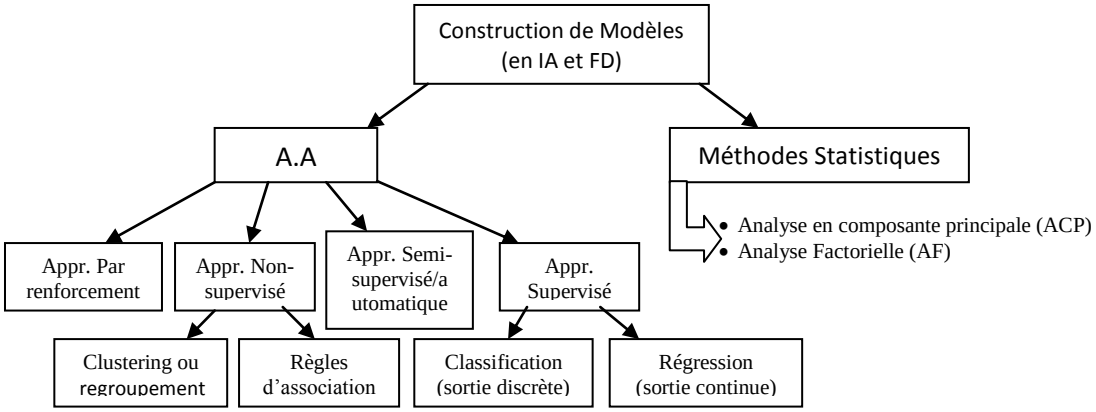


Figure 2.1. Schéma des différentes techniques issues de l'IA et FD pour la construction de modèles de données.

Les deux principales composantes de la construction de modèles d'apprentissage et de classification sont des techniques issues de la **statistique multivariée** ou bien des techniques d'**apprentissage automatique** (AA).

Nous nous intéressons plus spécialement aux techniques de l'apprentissage automatique.

2.3. Présentation de la Problématique

Les méthodes dédiées à l'AA visent à construire des modèles génériques à partir de données fournies (observées). Ainsi, ces observations ou échantillons sont vues comme des exemples illustrant **les relations entre** des variables observées.

Le but : est alors d'utiliser ces exemples pour en **déduire des caractéristiques** ou **propriétés liant ces données entre elles**.

Problème : la difficulté repose sur le fait que les données de l'ensemble d'apprentissage ne contient souvent qu'un nombre fini d'exemples (cas, des méthodes statistiques).

Ainsi, on ne dispose donc pas de l'ensemble de **tous les comportements** ou **états possibles**, en fonction de toutes les entrées possibles (cas, des **modèles non-déterministes** ou **probabilistes**).

2.4. Données d'Apprentissage

Les données d'apprentissage sont, souvent, réparties en 3 catégories :

- **L'ensemble d'apprentissage** ou **population d'entrainement** : constitue l'ensemble des candidats ou exemples (images, attributs, DB, …) utilisés pour générer le modèle d'apprentissage. Alors que,
- **l'ensemble de Test** est constitué des candidats sur lesquels sera appliqué le modèle d'apprentissage (pour tester et corriger l'algorithme).
- **L'ensemble de validation** peut être utilisé lors de l'apprentissage (comme sous population de l'ensemble d'apprentissage) afin de valider (intégrer) le modèle et d'éviter le sur-apprentissage.

NB : Selon les domaines, les connaissances ou données d'apprentissage (tel en IA) peuvent être de diverses formes : mots, phrases, variables ou attributs, des vecteurs de valeurs : définissant un ensemble de propriétés d'un objet, …

2.5. Types d'Apprentissage

En fonction du type de problème que l'on se pose, voir Fig. 2.1, on peut avoir à mettre en place différents types d'apprentissage :

- **Apprentissage Supervisé** : Cette approche a pour objectif la conception d'un modèle reliant des données d'apprentissage à un **ensemble de valeurs de sortie** (un comportement).

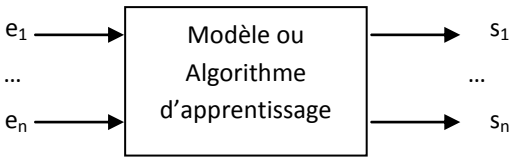


Figure 2.3. Schéma d'un modèle supervisé.

- **Apprentissage par Renforcement** [Kae 96] : Les données en entrée sont les mêmes que pour l'apprentissage supervisé, cependant **l'apprentissage est guidé par l'environnement** sous la forme de **récompenses** ou de **pénalités** données en **fonction de l'erreur commise** lors de l'apprentissage.
- **Apprentissage Non-Supervisé** [Bar 89] : Il vise à concevoir un modèle structurant l'information. La différence ici est que les **comportements** (ou catégories ou encore les classes) des données d'apprentissage ne sont pas connus, **c'est ce que l'on cherche à trouver**.

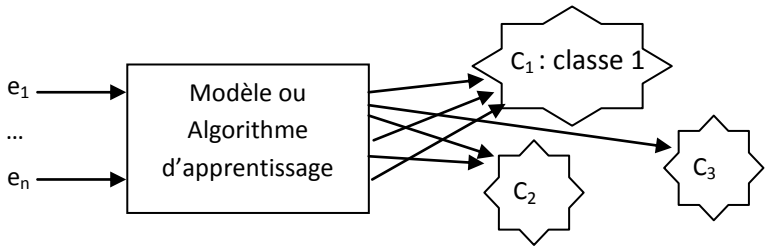


Figure 2.4. Schéma d'un modèle non supervisé.

- **Apprentissage Semi-Supervisé** [Cha 06] : Les données d'entrée sont constituées d'exemples **étiquetés** et **non étiquetés**. Ce qui peut être très utile quand on a deux types de données, car cela permet de ne pas en laisser de côté et d'utiliser toute l'information.

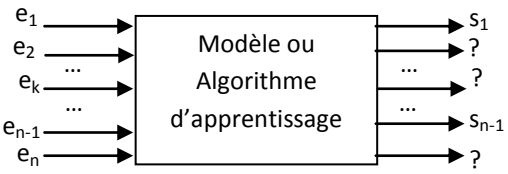


Figure 2.5. Schéma d'un modèle semi-supervisé ou incrémental.

NB : Dans notre cas, nous nous concentrerons sur les types d'apprentissage à savoir **supervisé** et **non-supervisé**.

2.6. Formulation

Une expression simplifiée du problème d'apprentissage peut être énoncée de la manière suivante :

Soit un ensemble d'apprentissage $S = \{(x_i, y_i)\}_{1\,..\,n}$ dont les éléments obéissent à la loi jointe $P(x, y) = P(x)P(y|x)$.

Ainsi, on ***cherche à approcher*** une *loi sous-jacente* $f(x)$ telle que $y_i = f(x_i)$ par une hypothèse $h_\alpha(x)$ aussi proche que possible, où les $\boldsymbol{\alpha}$ sont les **paramètres** du système d'apprentissage.

Remarque,

- Si $f(.)$ est **discrète**, on parle de **classification**.
- Si $f(.)$ est une **fonction continue**, on parle alors de **régression**.

Mais que veut-on dire par : "*aussi proche que possible*" ?

2.6.1. Calcul du risque

Pour *mesurer la qualité* d'une hypothèse $\boldsymbol{h_\alpha}$ on considère généralement une fonction de coût $\boldsymbol{Q(z=(x,y),\alpha) \in [a,b]}$ que l'on cherche à **minimiser**.

Exemples de fonction de coût,

- **Coût 0/1** : vaut **0** lorsque les étiquettes prévues et observées coïncident, **1** sinon. Ce type de coût est utilisé en *classification*.
- **Erreur quadratique :** $\boldsymbol{(f(x) - y)^2}$**.** Elle est utilisée particulièrement en *régression*.

Ainsi, on cherche à ***minimiser*** le risque : $\boldsymbol{R(\alpha) = \int Q(z,\alpha)\, dP(z)}$.

Comme on ne peut accéder directement à cette valeur, on construit donc le ***risque empirique*** qui mesure les *erreurs réalisées par le modèle* : $\boldsymbol{R_{emp}(\alpha) = \frac{1}{n}\sum_{i=1}^{n} Q(z_i,\alpha)}$.

Mais, quel est le lien entre $\boldsymbol{R_{emp}(\alpha)}$ et $\boldsymbol{R(\alpha)}$?

2.6.2. Théorie de l'Apprentissage de Vapnik

Vapnik [Vap 95] a pu montrer l'expression suivante $\forall\, \boldsymbol{m}$ ($\boldsymbol{m}$: la taille de l'échantillon d'apprentissage) avec une probabilité au moins égale à $\boldsymbol{1-\eta}$:

$$\boldsymbol{R(\alpha_m) \leq R_{emp}(\alpha_m) + (b-a)\sqrt{\frac{d_{VC}(log(2m/d_{VC})+1) - log(\eta/4)}{m}}} \qquad (2.1)$$

Cette formule n'est valide que lorsque $\boldsymbol{d_{VC} < m}$**.**

La minimisation du risque dépend :

- du **risque empirique**,
- d'un **risque structurel** lié au terme $\boldsymbol{d_{VC}}$ ($\boldsymbol{\text{dimension} - \text{VC}}$: dimension de Vapnik et Chervonenkis du modèle de classification) qui dépend de la complexité du modèle *h* choisi.

2.6.2.1. Dimension VC

Dans la théorie de l'apprentissage automatique, la $\boldsymbol{\text{dimension} - \text{VC}}$ (ou dimension de Vapnik-Chervonenkis) est une mesure de la capacité d'un algorithme de classification statistique ; elle est définie comme le *cardinal du plus grand ensemble de points* que l'algorithme peut *pulvériser*.

C'est un concept définit par Vladimir Vapnik et Alexey Chervonenkis, il est central dans la théorie de Vapnik-Chervonenkis.

La dimension VC est utilisée pour calculer la valeur de la ***marge d'erreur*** probable maximum d'un test de modèle de classification. Pour le test d'un modèle de classification sur des

données extraites de l'échantillon d'apprentissage de manière *indépendante et identiquement distribuée (i.i.d)*, cette valeur est calculée à partir de l'erreur d'apprentissage $\boldsymbol{EA(\alpha_m) = R(\alpha_m) - R_{emp}(\alpha_m)}$, comme dans l'équation (2.1) :

$$\boldsymbol{marge\ d'erreur = EA(\alpha_m) + (b - a)\sqrt{\frac{d_{VC}(log(2m/d_{VC})+1) - log(\eta/4)}{m}}} \qquad (2.2)$$

avec la probabilité de $\boldsymbol{1 - \eta}$.

2.6.2.2. Notion de capacité d'un modèle

On dit qu'un modèle de classification $\boldsymbol{f}$, prenant comme paramètre un vecteur $\boldsymbol{\theta}$, ***pulvérise*** (capacité de pulvériser) un ensemble de données $(x_1, x_2, \dots, x_n)$, si pour tout étiquetage de cet ensemble de données, il existe un $\boldsymbol{\theta}$ tel que le modèle $\boldsymbol{f}$ *ne fasse aucune erreur* dans l'évaluation de cet ensemble de données.

On appellera alors *dimension VC* d'un modèle $\boldsymbol{f}$ la *cardinalité* ou le *cardinal du plus grand ensemble pulvérisé* par $\boldsymbol{f}$.

En notant $\boldsymbol{D_f}$ la dimension VC du modèle $\boldsymbol{f}$, on aura donc :

$$\boldsymbol{D_f = max\{k | card(S) = k\ et\ f\ pulverise\ S\}}$$

2.6.3. Conditions de Construction

Ainsi, pour construire un bon modèle d'apprentissage, il est nécessaire de :

- ***Minimiser les erreurs*** sur la base d'apprentissage, c'est le principe d'induction naïf utilisé, par exemple, dans les réseaux de neurones.
- Construire un système capable de ***généraliser*** correctement.

2.7. Notions Elémentaires

2.7.1. Données

Elles représentent l'ensemble des informations relatives à un sujet ou objet d'étude. Aussi, une donnée constitue un attribut parmi d'autres si un objet d'une classe ou catégorie (ex., voiture) devrait être représenté par un ensemble de valeurs.

Une donnée devrait être spécifique et caractéristique d'une valeur ou propriété de l'objet que le système est censé discriminer ou destiné à discerner parmi les objets de l'environnement auquel il est dédié.

Exemples, la couleur RGB d'une orange, la forme ovale de la poire, le motif de la texture d'un tissu, …

2.7.2. Vecteur de Caractéristiques

Un vecteur caractéristique ou descripteur, ou encore vecteur de descripteurs, est une entité cohérente d'une ou plusieurs données caractéristiques ou propriétés regroupées, structurées et codées.

Exemple, vecteur caractéristique relatif aux avions = (prix, masse, type fuselage, motorisation, nombre réacteurs, nombre de places, type de trainée, …).

2.7.3. Classe

Une **classe**, une **catégorie** ou un **groupe**, d'objets sont des termes analogues. Une classe enseigne sur un ensemble d'objets ou échantillons de même nature et ayant le même vecteur descripteur comme modèle de description.

2.7.4. Notion de Distance

2.7.4.1. Distance d'un objet à une classe

La distance d'un point $\boldsymbol{x} \in \boldsymbol{E}$ (donnée ou objet) à une partie (partition ou classe) $\boldsymbol{A} \subset \boldsymbol{E}$, partie de ***l'espace métrique*** $\boldsymbol{E}$, est donnée par : $d(x, A) = inf\{d(x, y)/y \in A\}$.

2.7.4.2. Distance entre deux classes

La distance entre deux ensembles (classes) d'objets $\boldsymbol{E1}$ et $\boldsymbol{E2}$, avec $\boldsymbol{E1}$ et $\boldsymbol{E2}$ deux parties non vides d'un espace métrique $\boldsymbol{E}$ muni d'une distance d, est donnée par :

$$d(E1, E2) = inf\{d(x, y)/\ (x, y) \in E1 \text{ x } E2\}$$

2.7.5. Distances Usuelles

2.7.5.1. Distance de Manhattan

$$1 - ditance(A, B) = d_1(A, B) = \sum_{i=1}^{n} |b_i - a_i|$$

2.7.5.2. Distance Euclidienne

Dans le plan 2D :

$$d(A,B) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

De manière générale, pour des descripteurs de dimension n :

$$2 - distance(A,B) = d_2(A,B) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

2.7.5.3. Distance de Tchebychev

$$\infty - distance(A,B) = \lim_{p\to\infty} \sqrt[p]{\sum_{i=1}^{n}|x_i - y_i|^p} = \sup_{1\leq i\leq n}|x_i - y_i|$$

2.7.5.4. Distance de Mahalanobis

Elle se base sur la corrélation entre les variables (pour mesurer la ressemblance entre les descripteurs des échantillons) par lesquelles différents modèles peuvent être identifiés et analysés. C'est, aussi, une manière utile pour déterminer la *similarité* entre une série de données (où chaque donnée est un vecteur de valeurs) inconnue et d'autres connues.

La distance de *Mahalanobis* tient compte de la variation (variance) et de la covariation entre les données ; son calcul de se fait à travers la matrice de covariance qui permet de quantifier les écarts conjoints des donnés ou variables par rapport à leurs espérances respectives.

Chapitre 3

Apprentissage et **C**lassification **S**upervisés

Chapitre 3
Apprentissage et Classification Supervisés

3.1. Introduction

Les techniques d'apprentissage supervisé permettent de construire des modèles à partir d'exemples d'apprentissage ou d'entrainement dont on connait le comportement ou la réponse. Ces modèles peuvent, ensuite, être utilisés dans différentes applications, telles que la **prédiction** ou la **classification**.

Toutefois, selon la complexité du problème, nous pouvons aussi distinguer deux catégories de solutions : il serait mieux d'approcher les données traitées par une fonction exacte, si la complexité est faible ; cependant, en cas de complexité élevée, nous aurons plus de chance d'approcher toutes les données que si nous optons pour une solution qui représente le modèle de données par une heuristique.

Les techniques dites de modélisation prédictive (*predictive modelling* ou *predictive data mining*) [Wei 98] analysent un ensemble de données et en extraient un ensemble de règles, formant un ***modèle prédictif***, afin d'essayer de prédire, avec la plus grande précision, le comportement de nouvelles données. Les techniques dans le domaine de la prédiction sont les plus classiques, car on peut directement les utiliser dans des applications concrètes.

3.2. Formulation

Supposons qu'un jeu de données d'apprentissage est formulé par N variables (descripteur de données à N valeurs) le décrivant (par exemple, le cas des images en forme de matrices d'entiers).

Nous disposons alors, pour un ensemble donné, de deux types d'informations : un **vecteur de valeurs** $X = (x_1, \dots, x_N)$ prises par chaque variable, et une **valeur de sortie Y** appelée **valeur supervisée** ou **réponse supervisée** (qui peut être une classe si l'on prend un problème de classification).

Si nous formalisons le problème d'apprentissage décrit précédemment, nous pouvons le représenter comme un ensemble de **couples entrée-sortie** $(\boldsymbol{X_i}, \boldsymbol{Y_i})$, avec $i \in [1,\ n]$, n étant le nombre d'exemples ou d'échantillons disponibles.

On appelle alors ***fonction d'apprentissage*** la fonction notée : $\boldsymbol{l} : X \rightarrow Y$ qui associe un **résultat** (valeur) **supervisé** à chaque vecteur d'entrée.

Le ***but*** *d'un algorithme d'apprentissage supervisé* sera donc d'*approcher cette fonction* $\boldsymbol{l}$, uniquement à partir des exemples d'apprentissage.

En fonction du résultat (comportement) supervisé que l'on veut obtenir, on peut distinguer deux types problèmes :

- **Régression** : lorsque le résultat supervisé que l'on cherche à estimer est une valeur dans un ensemble **continu** de réels.
- **Classification** : lorsque l'ensemble des valeurs de sortie est **discret**. Ceci revient à attribuer une **classe** (aussi appelée *étiquette* ou *label*) pour chaque vecteur d'entrée.

Nous nous plaçons souvent dans le cas de problème classification à deux classes (**2-classe**) qui peut être facilement étendu à **N-classe**.

3.3. Problème Linéaire et Non-Linéaire

Les méthodes de classification supervisée peuvent être basées sur

- des ***hypothèses probabilistes*** (cas du classifieur naïf bayésien),
- des ***notions de proximité*** (exemple, ***k*** plus proches voisins) ou
- des recherches dans des ***espaces d'hypothèses*** (exemple, arbres de décisions).

En fonction du problème, il faut pouvoir choisir le *classifieur* approprié, c'est-à-dire celui qui sera à même de séparer au mieux les données d'apprentissage.

On dit qu'un problème est **linéairement séparable** si les exemples de classes différentes sont complètement séparables par un **hyperplan** (appelé hyperplan séparateur, ou séparatrice). Ce genre de problème se résout par des classifieurs assez simples, qui ont pour but de trouver l'équation de l'hyperplan séparateur.

Mais, le problème peut également être **non séparable** de manière linéaire comme illustré dans la figure 3.1. Dans ce cas, il faut utiliser d'autres types de classifieurs, souvent plus longs à paramétrer, mais qui obtiennent des résultats plus précis.

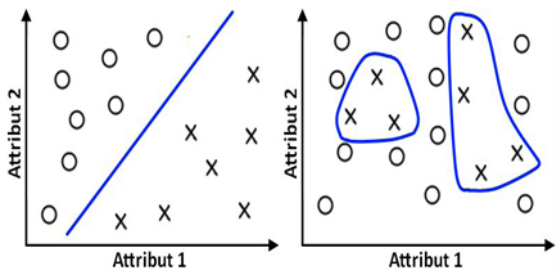


Figure 3.1. A Gauche : Problème linéairement séparable (Frontière linéaire).
A Droite : Problème non linéairement séparable.

Remarque,

Un problème, initialement, ***non linéairement séparable*** peut s'avérer séparable avec l'ajout d'un nouvel attribut (*cf*. figure 3.2). D'où l'intérêt d'un choix judicieux de ces attributs. C'est ce principe qui est utilisé par le classifieur *Support Vector Machine* (SVM) que nous verrons dans la suite (*cf*. section 3.6.3).

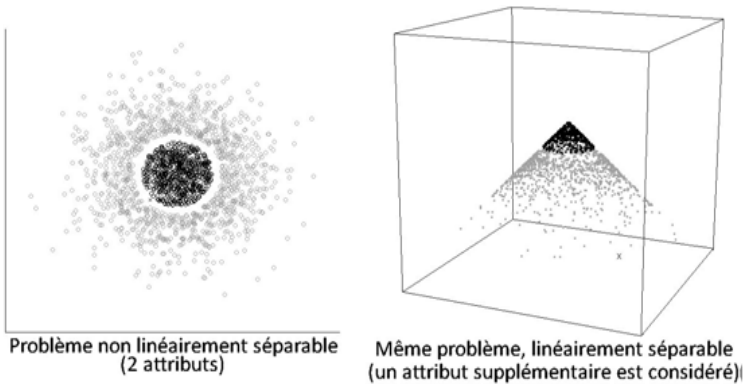


Figure 3.2. Même problème considéré avec 2 ou 3 attributs.

3.4. Classifieurs à mémoire

L'intérêt de ces classifieurs est qu'ils ne nécessitent aucune phase d'apprentissage ou d'entrainement. Ainsi, ils permettent de déduire directement la classe d'un nouvel exemple à partir de l'ensemble d'apprentissage.

3.4.1. *K*-plus proches Voisins

Le classifieur des ***k*** plus proches voisins ou ***k-ppv*** (k-Nearest Neighbor ou k-NN, en anglais) est l'un des algorithmes de classification les plus simples.

Principe : Un exemple est *classifié par vote majoritaire* de ses ***k*** "voisins" (par mesure de distance), c'est-à-dire qu'il est prédit de classe **C** si la classe la plus représentée parmi ses ***k*** voisins est la classe **C**.

Un cas particulier est le cas où ***k = 1***, l'exemple est alors affecté à la classe de son plus proche voisin.

L'opérateur de distance le plus souvent utilisé est la **distance Euclidienne**, cependant, en fonction du problème, on peut encore utiliser les distances de *Hamming*, de *Mahalanobis*, etc.

Remarques,

- Le choix du ***k*** est très important pour la classification.

- On s'abstient de choisir des ***valeurs paires*** de ***k***, pour éviter les cas d'égalité.

Exemple, sur la figure 3.3, on peut voir l'effet du choix de ***k*** sur le résultat de la classification. En effet, si **k=1; 2; 3** l'exemple à prédire (noté "**?**") serait classifié comme étant de la classe "**X**", mais si **k=5**, il serait classifié comme étant de la classe "**O**".

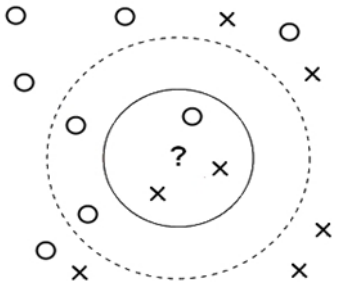


Figure 3.3 Schéma d'une classification par la méthode k-NN

Un pseudo-algorithme des *K* plus proches voisins est donné par :

Pseudo-Algorithme $K - ppv$;

Déclarations

- M : nombre de classes d'apprentissage $C = \{c_1, \dots, c_M\}$;
- N : nombre d'exemples d'entrainement $E = \{e_1, \dots, e_N\}$;
- $Ent = \{(e_i,\ c_k)\}$: ensemble d'apprentissage formé par les couples $(e_i,\ c_k)$; /*e_i est l'exemple d'apprentissage et c_k sa classe d'appartenance */
- ex : exemple test /*dont on cherche la classe d'appartenance*/

Début

< On cherche à classer e_x ? > ;

Pour Chaque exemple $(e_i,\ w) \in Ent$ **Faire**

< Calculer la distance $D(e_i,\ e_x)$ entre e_i et e_x > ;

FPour

< Trier les échantillons e_i par ordre croissant des distances > ;

Pour les k plus proches e_i de e_x (les k premières – ayant les plus petites- $D(e_i,\ e_x)$) **Faire**

< Compter le nombre d'occurrences de chaque classe > ;

FPour

< Attribuer à e_x la classe c_j la plus fréquente > ; /*Celle qui apparait le plus souvent*/

Fin.

3.4.2. Classifieur Naïf Bayésien

La classification naïve bayésienne repose sur l'hypothèse que les **attributs** sont **fortement** (ou **naïvement**) ***indépendants***. Elle est basée sur le *théorème de Bayes* qui ne s'applique que sous cette hypothèse.

Théorème de Bayes est donné par : $P(x|y) = \frac{P(y|x)P(x)}{P(y)}$, avec $P(x|y)$ est la probabilité conditionnelle d'un événement x sachant qu'un autre événement y de probabilité non nulle s'est réalisé.

Dans le cas d'une classification, on pose H l'hypothèse selon laquelle un "vecteur d'attributs X (représentant un objet) appartient à une classe C", et l'on suppose que l'on cherche à estimer la probabilité $P(H|X)$, c'est-à-dire la probabilité que l'hypothèse H soit ***vraie***, considérant X.

Ainsi, si l'on a un nouvel exemple x $= (x_1, x_2, \dots, x_n)$ dont on veut trouver la classe, on va chercher la probabilité maximale d'appartenance à cette classe :

$$P(x)^* = argmax\, P(x_1, x_2, \dots, x_n|H) * P(H) \qquad (3.1)$$

Cette équation est directement déduite du théorème de Bayes. En effet, comme l'objectif est de faire une **maximisation** et que **le dénominateur ne dépend pas de x**, on peut le supprimer.

Cependant, cette probabilité pourrait être beaucoup trop compliquée à estimer, si l'on considère le nombre de descriptions possibles. C'est alors que l'on utilise ***l'hypothèse d'indépendance***, qui nous permet de décomposer la probabilité conditionnelle en un produit de probabilités conditionnelles. Le classifieur devient alors :

$$Classe(x) = arg\ max \prod_{i=1}^{n} P(x_i|H) * P(H) \qquad (3.2)$$

Remarque,

- **Avantage**, ce classifieur est souvent utilisé, car très simple d'emploi.
- **Inconvénient**, cependant, il est très sensible à leur corrélation.

3.4.3. Classifieur EM

EM (Expectation Maximisation ou encore Maximisation de l'Espérance, en français) est un algorithme qui permet d'étudier un modèle qui possède des **variables latentes** ou **cachées**.

Les algorithmes antérieurs étaient des algorithmes visant à estimer le paramètre θ, maximisant la vraisemblance des $p_\theta(x)$, où x est le vecteur des données observées.

3.4.3.1. Exemple

La densité représentée sur la figure 3.4 s'apparente à ***une moyenne de deux gaussiennes***.

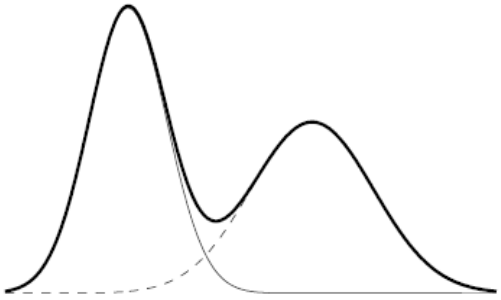


Fig. 3.4. Densité moyenne de deux densités gaussiennes, pour lequel est naturel d'introduire un modèle de mélange.

Il est donc naturel d'utiliser **un modèle de mélange**, et d'introduire **une variable cachée $\boldsymbol{z}$** de Bernoulli définissant l'appartenance à l'une ou l'autre des gaussiennes.

On a donc, $\boldsymbol{z} \in \{\mathbf{1}, \mathbf{2}\}$ et $\boldsymbol{x}|\boldsymbol{z} = \boldsymbol{i} \sim \boldsymbol{\mathcal{N}}(\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i})$. La densité $\boldsymbol{p}(\boldsymbol{x})$ est une densité ***combinaison*** convexe ***de densités normales*** :

$$\boldsymbol{p}(\boldsymbol{x}) = \boldsymbol{p}(\boldsymbol{x}, \boldsymbol{z} = \mathbf{1}) + \boldsymbol{p}(\boldsymbol{x}, \boldsymbol{z} = \mathbf{2}) = \boldsymbol{p}(\boldsymbol{x}|\ \boldsymbol{z} = \mathbf{1})\boldsymbol{p}(\boldsymbol{z} = \mathbf{1}) + \boldsymbol{p}(\boldsymbol{x}|\ \boldsymbol{z} = \mathbf{2})\boldsymbol{p}(\boldsymbol{z} = \mathbf{2})$$

C'est un ***modèle de mélange***, un moyen de modéliser simplement des phénomènes compliqués.

3.4.3.2. Objectif

C'est d'obtenir un maximum de vraisemblance [Cor 03].

Soit $\boldsymbol{z}$ la variable cachée, $\boldsymbol{x}$ sont les données observées. On suppose que les x_i, $i \in \{1, \dots, n\}$, sont des ***données i.i.d.***(indépendantes et identiquement distribuées).

La vraisemblance à maximiser:

$$p_\theta(x) = \prod_i p_\theta(x_i) = \prod_i \sum_z p_\theta(x_i, z)$$

Équivalente à, $\log p_\theta(x) = \sum_i log \sum_z p_\theta(x_i, z)$

Il est possible d'envisager deux manières différentes de résoudre ce problème :

1. de manière directe, si le problème le permet, par exemple ***par montée de gradient***.
2. en utilisant l'***algorithme EM***.

3.4.3.3. L'algorithme EM

On introduit dans l'expression de la vraisemblance la fonction $q(z)$ telle que $q(z) \geq 0$ et $\sum_z q(z) = 1$, et on a :

$\log p_\theta(x) = \log \sum_z p_\theta(x,z) = \log \sum_z \left(\frac{p_\theta(x,z)}{q(z)}\right) q(z) \geq \sum_z q(z) \log \left(\frac{p_\theta(x,z)}{q(z)}\right)$, par l'inégalité de Jensen, et la concavité de log

$$= \sum_z q(z) \log p_\theta(x,z) - \sum_z q(z) \log q(z) = \mathcal{L}(q,\theta).$$

avec égalité si et seulement si $q(z) = \frac{p_\theta(x,z)}{\sum_z p_\theta(x,z)} = p_\theta(z|x)$.

a. **Proposition**. $\forall\theta, \forall q \ \log p_\theta(x) \geq \mathcal{L}(q,\theta)$, avec égalité ssi $q(z) = p_\theta(z|x)$.

b. **Exemple**, on a créé une fonction qui est toujours en dessous la fonction $\log(p_\theta(x))$.

c. **L'Algorithme**. EM est un algorithme de maximisation alternée par rapport à q et θ.

On initialise à θ_0 puis on itère pour $t > 0$, en alternant les étapes suivantes *jusqu'à* ***convergence*** :

- **Étape-E** (E-Step) : $q_{t+1} \in \ argmax_q(\mathcal{L}(q,\theta_t))$.
- **Étape-M** (M-Step) : $\theta_{t+1} \in \ argmax_\theta(\mathcal{L}(q_{t+1},\theta))$.

d. **Les propriétés de l'algorithme**

- C'est un algorithme de montée : $\forall t \ \log(p_{\theta_t}) \geq \log(p_{\theta_{t-1}})$.
- L'algorithme converge.
- Cependant, il **ne converge pas globalement** mais vers un maximum local car on est dans un cas non convexe. Souvent, l'optimum global est infini.
- Comme pour *K-means* (*cf.* section 4.3), pour avoir un résultat plus sûr, on réitère plusieurs fois l'algorithme et on choisit l'essai avec la meilleure vraisemblance.

e. **La Méthodologie EM**

Le but qui est de ***maximiser la vraisemblance incomplète*** $\log(p_\theta(x))$. Pour cela on peut utiliser la méthodologie suivante :

1. Écrire la vraisemblance complète $l_c = \log(p_\theta(x,z))$.
2. ***E-Step*** : écrire l'espérance de la vraisemblance $E[q(z)p_\theta(x,z)]$, voulant que $q(z) = p_\theta(z|x)$.
3. ***M-Step*** : maximiser par rapport à θ.

3.4.3.4. Mélanges de Gaussiennes

Soient les couples (x_i, z_i), pour $i \in \{1, \dots, n\}$, avec $x_i \in \mathbb{R}^p$ ***i.i.d*** (indépendantes et identiquement distribuées), $z_i \sim$ Multinomiale(Π) ***i.i.d.*** et $(x_i|z_i = q) \sim \boldsymbol{\mathcal{N}(\mu_q, \Sigma_q)}$.

a. Calcul de $\boldsymbol{p(z|x)}$

On utilise la formule de Bayes pour exprimer $p(x_i)$:

$$p(x_i) = \sum_{z_i} p(x_i, z_i) = \sum_{z_i} p(x_i|\, z_i)p(z_i) = \sum_{q=1}^{k} p(x_i|\, z_i = q)p(z_i = q)$$

Puis, on exprime $p(z|x)$:

$$p(z_i = q|x_i) = \frac{p(x_i|\, z_i = q)p(z_i = q)}{p(x_i)}$$

$$\propto p(x_i|\, z_i = q)p(z_i = q)$$

$$= \frac{\prod_q \mathcal{N}(x_i|\, \mu_q, \Sigma_q)}{\sum_{l=1}^{k} \prod_l \mathcal{N}(x_i|\, \mu_l, \Sigma_l)} = \tau_i^q(\theta)$$

On rappelle que $\mathcal{N}(x_i|\, \mu, \Sigma) = -\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

b. Vraisemblance complète

$$l_c = \log p_\theta(x, z) = \sum_{\mathrm{i}=1}^{\mathrm{n}} \log p_\theta(x_i, z_i) = \sum_{\mathrm{i}=1}^{\mathrm{n}} \log\big(p(z_i)p(x_i|\, z_i)\big)$$

$$= \sum_{\mathrm{i}=1}^{\mathrm{n}} \sum_{\mathrm{q}=1}^{\mathrm{k}} \log\big(p(z_i = q)\big) + \log\big(p(x_i|\, z_i = q)\big)$$

$$= \sum_{\mathrm{i}=1}^{\mathrm{n}} \sum_{\mathrm{q}=1}^{\mathrm{k}} \log\big(\Pi_q\big) + \log\left(\frac{1}{(2\pi)^{\frac{d}{2}}} + \log\left(\frac{1}{|\Sigma_q|^{\frac{1}{2}}}\right) - \frac{1}{2}\big(x_i - \mu_q\big)^T\Sigma_q^{-1}\big(x_i - \mu_q\big)\right)$$

On écrit maintenant $E[p(z|x)]$, quantité que l'on va maximiser par la formulation méthodologique EM.

$$E[p(z|x)] = \sum_{\mathrm{i}=1}^{\mathrm{n}} \sum_{\mathrm{q}=1}^{\mathrm{k}} \tau_i^q\big(\theta_{précédent}\big)\left(\log\big(\Pi_q\big) + \log\left(\frac{1}{(2\pi)^{\frac{k}{2}}} + \log\left(\frac{1}{|\Sigma_q|^{\frac{1}{2}}}\right) - \frac{1}{2}\big(x_i - \mu_q\big)^T\Sigma_q^{-1}\big(x_i - \mu_q\big)\right)\right)$$

- ***Etape-E*** *(E-Step)*. La première étape de l'algorithme EM est de maximiser $E[p(z|x)]$ par rapport à Π. En maximisant selon les méthodes vues précédemment, on obtient :

$$\Pi_q = \frac{\sum_i \tau_i^q}{\sum_i \sum_l \tau_i^l} = \frac{1}{n}\sum_{i=1}^{n} \tau_i^q$$

- ***Etape-M*** *(M-Step).* On cherche ici à maximiser par rapport à μ et Σ. En prenant les gradients selon les μ_q puis selon les Σ_q, on obtient que :

$$\mu_q(t+1) = \frac{\sum_i \tau_i^q(t) x_i}{\sum_i \tau_i^q(t)}$$

$$\Sigma_q(t+1) = \frac{\sum_i \tau_i^q(t)(x_i - \mu_q(t+1))(x_i - \mu_q(t+1))^T}{\sum_i \tau_i^q(t)}$$

NB : L'étape-M du classifieur EM correspond à l'estimation des moyennes dans *K-means*.

3.5. Arbres de Décision

3.5.1. Contexte Général

Les arbres de décision sont un outil très populaire de classification. Leur principe repose sur la construction d'un arbre de taille limitée. La racine constitue le point de départ de l'arbre et représente l'ensemble des données d'apprentissage. Puis ces données sont segmentées en *plusieurs sous-groupes*, en fonction d'une **variable discriminante** (un des **attributs**).

Exemple, sur la Figure 3.5, la première *variable discriminante* est la température corporelle. Elle divise la population en deux sous-groupes : les personnes dont la température est supérieure à 37°C et les autres. Le processus est ensuite réitéré au deuxième niveau de l'arbre, ou les sous-populations sont segmentées à leur tour en fonction d'une autre *valeur discriminante* (dans l'exemple, c'est la toux.

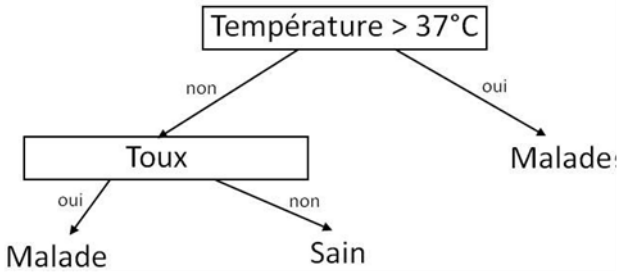


Figure 3.5 Schéma d'un arbre de décision.

Une fois l'arbre construit à partir des données d'apprentissage, on peut **prédire un nouveau cas** en le faisant descendre le long de l'arbre, jusqu'à une feuille. Comme la **feuille correspond à une classe**, l'exemple sera prédit comme faisant partie de cette ***classe***.

Dans l'exemple, on peut en déduire qu'une personne qui a une température < 37°C et qui a de la toux est prédite comme *malade*, tandis qu'une personne qui a une température < 37°C mais pas de toux est considérée comme *saine*.

Lors de la création de l'arbre, la première question qui vient à l'esprit est le choix de la variable de segmentation sur un sommet. Pourquoi par exemple avons-nous choisi la variable "température" à la racine de l'arbre ? Il nous *faut donc une* ***mesure*** *afin* ***d'évaluer la qualité d'une segmentation*** et *sélectionner la meilleure variable* sur chaque sommet. Ces algorithmes s'appuient notamment sur les techniques issues de la ***théorie de l'information***, et notamment la théorie de Shannon [Sha 48].

L'intérêt des arbres de décision est en premier lieu leur **lisibilité**. En effet, il est très simple de comprendre les décisions de l'arbre une fois celui-ci créé, ce qui n'est pas toujours le cas pour les autres classifieurs que nous verrons. D'autre part, l'algorithme de création des arbres de décision fait automatiquement **la sélection d'attributs jugés pertinents**, et ce, même sur des volumes de données importants.

3.5.2. Idée et Propriétés Générales

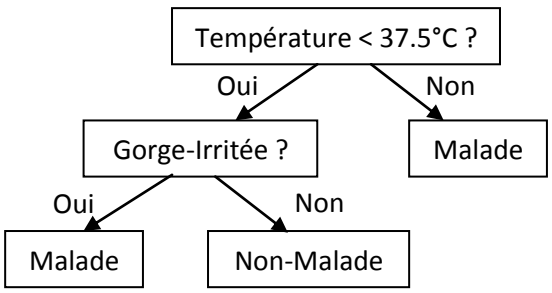
Diviser **récursivement** et le **plus efficacement** possible les individus de l'ensemble d'apprentissage par des tests définis à l'aide des variables jusqu'à ce que l'on obtienne des **sous-ensembles d'individus** ne contenant (presque) que des **exemples appartenant** à une **même classe** ! A la base, trois opérations seront nécessaires :

1. **Décider si un nœud est terminal** (tous les individus sont dans la même classe).
2. **Sélectionner un test à associer à un nœud** (utiliser critères statistiques).
3. **Affecter une classe à une feuille** (nœud terminal)- la classe majoritaire !

3.5.3. Exemple Introductif

Décider si un patient est **malade** ou **bien-portant** (**sain**) selon sa **température** et s'il a la **gorge irritée**. A partir des exemples (des patients), des attributs ou variables (Température et Gorge-irritée), des classes (malade ou sain), construire l'arbre de décision qui aura :

- 2 classes (malade et bien-portant).
- 2 variables (température et gorge-irritée).



3.5.4. Classification et Règles

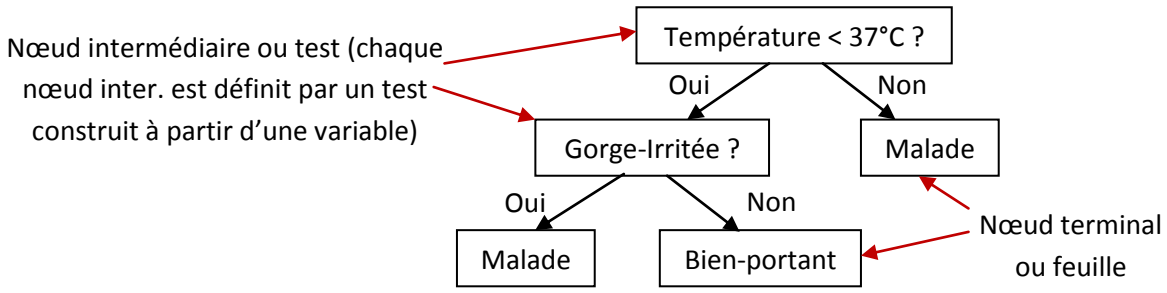
L'Arbre de Décision (AD) permet de classer un nouvel exemple : **(37.2, oui)**, c'est-à-dire, Température=37.2 et Gorge-Irritée=oui, comme appartenant à la classe **malade**.

L'AD peut être traduit en un système de règles ; lesquelles pouvant être considérées comme le pseudo-code ou l'algorithme de l'AD :

- **Si** (Temp. < 37.5) **et** (Gorge-irritée) **Alors** malade.
- **Si** (Temp. < 37.5) **et** Non (Gorge-irritée) **Alors** Sain.
- **Si** (Temp. ≥ 37.5) **Alors** malade.

3.5.5. Définition du Formalisme : Arbres de Décision

3.5.5.1. Vocabulaire



3.5.5.2. Inférence d'Arbres de Décision

Objectif : Inférer (déduire et aussi au sens de construire) un arbre de décision à partir d'exemples.

Pour ce faire, on a besoin :

a. de comprendre la répartition de la population (ex., de patients) dans l'arbre. Ainsi, il est intéressant de savoir mesurer le **degré de mélange** d'une population.
b. de la définition d'une méthode d'inférence, en saisissant :
   - Comment **sélectionner le test** à effectuer **à un nœud** ?
   - Comment décider si un **nœud est terminal** ?
   - Quelle **classe** associée à une **feuille** ?
c. Enfin, de comment tout écrire mathématiquement ?

3.5.6. Mélange et Degré de Mélange

Le calcul du degré de mélange des classes dans la population vient du besoin de comparer les différents choix possibles.

Ainsi, de ce besoin, on introduit **des fonctions** qui permettent de mesurer le *degré de mélange d'une population dans les différentes classes*.

Les propriétés de ces fonctions devraient être de la sorte :

- Le **minimum** est atteint lorsque tous les **nœuds** sont ‘‘**purs**’’ (‘‘purs’’ : si **tous les individus** associés au nœud appartiennent à la **même classe**). Ainsi, le **mélange** sera **minimal** (sinon nul).
- Le **maximum** est atteint lorsque les individus sont **équirépartis** entre les **classes** (**mélange maximal**).

3.5.6.1. Exemples de Fonctions Mélanges

- **Fonction d’Entropie**

$$Entropie(p) = -\sum_{k=1}^{c} P(k|p) \ln P(k|p)$$

Avec, classe ***k*** et nœud ***p***.

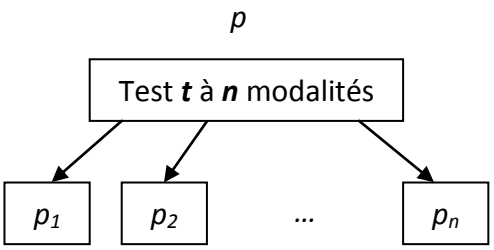
- **Fonction de Gini**

$$Gini(p) = 1 - \sum_{k=1}^{c} P^2(k|p) = 2 \sum_{k<k'} P(k|p)P(k'|p)$$

3.5.7. Notion de Gain

3.5.7.1. Représentation et Fonction Gain

- ***t*** : le test (la variable).
- ***n*** : le nombre de modalités de ***t***.
- ***i*** : la fonction pour mesurer le degré de mélange.



On introduit la fonction de gain :

$$Gain(p,t) = i(p) - \sum_{j=1}^{n} p_j\, i(p_j)$$

Avec,

- p_j : la proportion des individus de la position (nœud) p qui vont en position p_j.
- La position p est fixée !
- Le but est de **chercher le test** qui **maximise le gain** !

3.5.8. Algorithme de Construction d’un Arbre de Décision

Nous présentons le fonctionnement de l’algorithme à travers un exemple.

3.5.8.1. Données et Notations

Cet algorithme d'apprentissage et de classification correspond au classifieur CART (Classification And Regression Tree).

- **Entrée**
  - $\boldsymbol{n}$ individus,
  - $\boldsymbol{p}$ variables continues ou discrètes,
  - Une variable supplémentaire contenant la classe de chaque individu ($\boldsymbol{c}$ classes).
- **Sortie**
  - L'arbre de décision **T** construit.

Soient,

- $N(p)$: le nombre d'individus associés à la position (nœud) $\boldsymbol{p}$.
- $N(k|p)$: le nombre d'individus appartenant à la classe $\boldsymbol{k}$ en sachant qu'ils sont associés à la position $\boldsymbol{p}$.
- $\boldsymbol{P(k|p)} = \frac{N(k|p)}{N(p)}$: proportion des individus appartenant à la classe $\boldsymbol{k}$ parmi ceux de la position $\boldsymbol{p}$.

3.5.8.2. Jeu de Données

Soit le tableau suivant récapitulant l'ensemble des clients d'une compagnie d'assurance.

Id Client	Montant (M)	Age (A)	Résidence (R)	Etudes (E)	Internet (I)
1	Moyen	moyen	Village	Oui	Oui
2	élevé	moyen	Bourg	Non	Non
3	Faible	âgé	Bourg	Non	Non
4	Faible	moyen	Bourg	Oui	Oui
5	Moyen	jeune	Ville	Oui	Oui
6	élevé	âgé	Ville	Oui	Non
7	Moyen	âgé	Ville	Oui	Non
8	Faible	moyen	village	non	non

- **Variables**
  - **M** : salaire ou moyenne des montants sur le compte.
  - **A** : âge du client.
  - **R** : lieu de résidence du client.
  - **E** : le client a fait des études supérieures ou non ?
  - **I** : le client consulte ses comptes sur internet ou non ? (classe)
- **Mélange Initial**

Ainsi, avec **8** clients dont : **3** (**oui**) ont Internet (classe 1 : **oui**) et **5 non** (classe 2 : **non**), le mélange initial (selon Gini) :

$$\boldsymbol{Mélange\ initial} = \boldsymbol{Mélange(Gini)} = 1 - \left(\frac{3}{8}\right)^2 - \left(\frac{5}{8}\right)^2 = \frac{15}{32} = 0{,}46875$$

3.5.8.3. Procédé

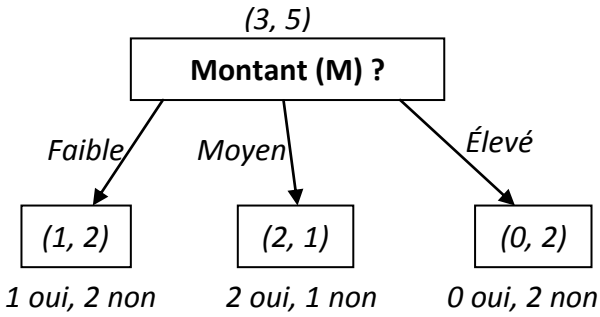
La construction est descendante : on commence par tester les candidats à la racine.

Au début, tous les individus sont regroupés (au niveau 0, la racine de l'arbre).

Ainsi, **quatre (04) constructions sont possibles**, suivant les variables : Montant (M), âge (A), résidence (R) et études (E).

3.5.8.4. Tester les Candidats à la Racine

a. Construction selon la variable M (Montant)



Mélange(Faible) = Gini = $1 - \left(\frac{1}{3}\right)^2 - \left(\frac{2}{3}\right)^2 = \frac{4}{9}$

Mélange(élevé) = Gini = $1 - \left(\frac{0}{2}\right)^2 - \left(\frac{2}{2}\right)^2 = 0$

Mélange(Moyen) = Gini = $1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = \frac{4}{9}$

On calcule le Gain selon la variable M :

$$Gain(M) = mélange(initial) - \frac{n_{Faible}}{n} \text{ x } mélange(F) - \frac{n_{Moyen}}{n} \text{ x } mélange(M) - \frac{n_{élevé}}{n} \text{ x } mélange(E)$$

$$\boldsymbol{Gain(M)} = \frac{15}{32} - \frac{3}{8}.\frac{4}{9} - \frac{3}{8}.\frac{4}{9} - \frac{2}{8}.0 = \frac{13}{96} = \mathbf{0,135}$$

b. Construction selon la variable A (âge)

Après avoir calculé les mélanges selon Gini, on calcule le Gain selon la variable A :

$$Gain(A) = mélange(initial) - \frac{n_{Jeune}}{n} \text{ x } mélange(J) - \frac{n_{Moyen}}{n} \text{ x } mélange(M) - \frac{n_{âgé}}{n} \text{ x } mélange(Ä)$$

$$\boldsymbol{Gain(A)} = \frac{15}{32} - \frac{1}{8}.0 - \frac{4}{8}.\frac{1}{2} - \frac{3}{8}.0 = \frac{7}{32} = \mathbf{0,219}$$

c. Construction selon la variable R (Résidence)

On calcule le Gain selon la variable R :

$$Gain(R) = mélange(initial) - \frac{n_{Village}}{n} \text{ x } mélange(V) - \frac{n_{Bourg}}{n} \text{ x } mélange(B) - \frac{n_{ville}}{n} \text{ x } mélange(Vi)$$

$$\boldsymbol{Gain(R)} = \frac{15}{32} - \frac{2}{8}.\frac{1}{2} - \frac{3}{8}.\frac{4}{9} - \frac{3}{8}.\frac{4}{9} = \frac{1}{96} = \mathbf{0,010}$$

d. Construction selon la variable E (étude)

Après avoir calculé les mélanges selon Gini, on calcule le Gain par rapport la variable E :

$$Gain(E) = mélange(initial) - \frac{n_{oui}}{n} \text{ x } mélange(O) - \frac{n_{non}}{n} \text{ x } mélange(N)$$

$$\boldsymbol{Gain(E)} = \frac{15}{32} - \frac{5}{8}.\frac{12}{25} - \frac{3}{8}.0 = \frac{27}{160} = \mathbf{0,169}$$

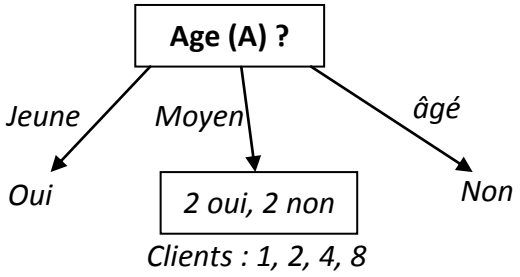
3.5.8.5. Quel Test Choisir ?

Variable Test	Composition nœuds	Gain
Montant (M)	(1, 2) ; (2, 1) ; (0, 2)	0,135
Age (A)	**(1, 0) ; (2, 2) ; (0, 3)**	**0,219**
Résidence (R)	(1, 2) ; (1, 2) ; (1, 1)	0,010
Etudes (E)	(3, 2) ; (0, 3)	0,169

Remarque,

- Sur la ***variable R***, aucune discrimination sur aucune branche, ainsi on ne gagne rien avec ce test !
- Sur la ***variable A***, deux nœuds sur trois sont ‘‘**purs**’’, ce qui semble intéressant !

3.5.8.6. Le Premier Niveau de l'Arbre Appris



L'étape suivante consiste à **ignorer les valeurs** (les supprimer du tableau de valeurs) pour laquelle ‘‘**Age = jeune**’’ et ‘‘**Age = âgé**’’ (pour les lignes : 3, 5, 6, 7) et **ne pas prendre en considération** la variable Age **A** (retirer la colonne Age).

Puis, continuer la construction des autres niveaux selon les variables restantes, à savoir : **M**, **R** et **E**.

3.5.8.7. Tester les Candidats Suivants (Construction du 2ème Niveau)

$$Mélange(initial\ de\ la\ population) = Gini = 1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2 = \frac{1}{2} = 0{,}5$$

a. Construction selon la variable M (Montant)

On calcule le Gain selon la variable M : $\boldsymbol{Gain(M)} = \frac{1}{2} - \frac{2}{4}.\frac{1}{2} - \frac{1}{4}.0 - \frac{1}{4}.0 = \frac{27}{160} = \mathbf{0,25}$

b. Construction selon la variable R (Résidence)

On calcule le Gain selon la variable R : $\boldsymbol{Gain(R)} = \frac{1}{2} - \frac{2}{4}.\frac{1}{2} - \frac{2}{4}.\frac{1}{2} - \frac{0}{4}.\infty = \frac{0}{2} = \mathbf{0}$

c. Construction selon la variable E (étude)

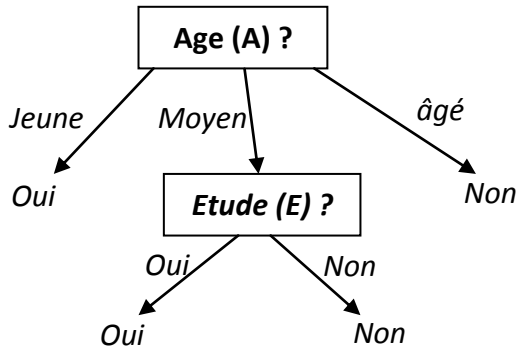
Après avoir calculé les mélanges selon Gini, on calcule le Gain par rapport la variable E :

$\boldsymbol{Gain(E)} = \frac{1}{2} - \frac{2}{4}.0 - \frac{2}{4}.0 = \mathbf{0}$

Remarque,

- Sur **E**, les deux (02) **nœuds sont purs**, et le **gain est le plus élevé**. Ainsi, le niveau 2 de l'arbre de décision est construit sur E.
- Cette construction est la dernière et termine l'arbre puisque **tous les nœuds ainsi formés sont des feuilles** (oui ou non).

3.5.8.8. Arbre Finalement Appris



NB : La structure de l'arbre de décision construit **dépend de l'ensemble d'apprentissage**.

3.5.9. Faiblesses

- C'est un algorithme Glouton, sans backtrack (sans retracer ou trace arrière).
- Transposables en règles avec des règles ayant des attributs communs, en particulier l'attribut utilisé à la racine.
- Présentent des difficultés avec les concepts disjonctifs (cf. *agaricus-lepiota*).
- Faiblesse du codage attributs-valeurs (ex., classification de molécules !).

3.5.10. Eviter le Sur-apprentissage

On pourrait calculer un arbre ‘‘**parfait**’’ ou presque. Mai, en pratique, un tel arbre n'existe pas toujours !

Ainsi, l'objectif est de construire un arbre avec la **plus petite erreur de classification** possible.

En général,

- L'**erreur** d'apprentissage **diminue** à chaque étape.
- L'**erreur réelle** (de validation) **diminue**, **se stabilise** et puis **augmente** ! c'est le phénomène de sur-apprentissage.

Ainsi, pour éviter le sur-apprentissage :

- On devrait pouvoir **arrêter la croissance de l'arbre** à tout moment, mais, à nos jours, *pas de critères théoriques* à définir !
- D'où, le **dilemme** du risque : risque **d'arrêter trop tôt** contre (vs) le risque **d'arrêter trop tard** !

Il existe des méthodes en deux phases :

1. construction de l'arbre.
2. **élagage** pour essayer de *diminuer l'erreur réelle* (de validation).

3.6. Classifieurs Construisant des Hyperplans Séparateurs

3.6.1. Analyse Discriminante Linéaire

L'analyse discriminante linéaire (Linear Discriminant Analysis ou **LDA**) [Fuk 90] a pour objectif de trouver les vecteurs discriminants optimaux (transformation) en **maximisant le ratio** entre la **distance interclasse** et la **distance intraclasse**, afin de discriminer au maximum les différentes classes. Cette technique permet à la fois de faire de la classification linéaire, mais également de réduire le nombre d'attributs.

On représente les données d'entrée par une matrice $A = (a_{ij}) \in \mathbb{R}^{n \times N}$ (pour n attributs et N exemples), où chaque colonne (notée a_k avec $1 \leq k \leq N$) correspond à un *exemple* et chaque ligne correspond à un *attribut*. Le but est alors de ***projeter ces données*** *sur un* ***espace vectoriel*** **G** de plus petite dimension ($l < n$) par une ***transformation linéaire*** $G \in \mathbb{R}^{n \times l}$. Les données projetées sont alors représentées par un vecteur $y = (y_1, y_2, \dots, y_l)$, telle que chaque composante y_k est définie par la transformation :

$$G: a_k \in \mathbb{R}^n \rightarrow y_k = G^T * a_k \in \mathbb{R}^l \qquad (3.3)$$

Cette transformation est calculée afin que la *structure des classes* de l'espace vectoriel original soit ***préservée*** *dans* ***l'espace réduit***.

Soit p le nombre de classes de notre problème. On suppose donc que les données sont partitionnées en p sous-ensembles (appelés ***clusters***). Pour chaque exemple appartenant à une classe, on peut décomposer les colonnes de la matrice A en p sous matrices $B_i \in \mathbb{R}^{nxb_i}$, avec $1 \leq i \leq p$.

On note :

- $\bar{s}$ le centre de gravité de l'ensemble des exemples,
- $\overline{s_i}$ le centre de gravité de la i^{ème} classe,
- b_i la taille de la sous matrice B_i correspondant à la $i^{ème}$ classe,
- S_i la matrice de covariance de la $i^{ème}$ classe.

L'idée : est de rendre les *clusters* les ***plus compacts possibles***, c'est-à-dire que les éléments qui les composent doivent être groupés, mais chaque *cluster* doit être éloigné des autres.

Pour quantifier la qualité d'un cluster, on définit deux matrices :

- $S_w = \sum_{i=1}^{p} b_i S_i$ la ***matrice intraclasse***,
- $S_b = \sum_{i=1}^{p} b_i (\overline{s_i} - \bar{s})(\overline{s_i} - \bar{s})^T$ la ***matrice interclasse.***

La trace (somme des coefficients diagonaux) de S_w mesure alors la ***cohésion intraclasse***, tandis que la trace de S_b mesure la ***séparation interclasse***.

Après projection, suivant la transformation G, ces matrices deviennent :

- $S_w^L = G^T S_w G$,
- $S_b^L = G^T S_b G$.

Le but : de LDA est alors de ***trouver la transformation optimale*** G qui ***maximise la trace*** de la matrice S_b^L, ***tout en minimisant la trace*** de la matrice S_w^L, ce qui peut être résumé en ce problème d'optimisation, dont on note $\boldsymbol{G}^*$ la solution :

$$\boldsymbol{G^* = maxtrace\left((S_w^L)^{-1}\, S_b^L\right) = maxtrace\left((G^T S_w G)^{-1}\, G^T S_b G\right)} \qquad (3.4)$$

Ce problème est résolu de manière exacte [Fuk 90], en faisant une décomposition spectrale sur la matrice $(S_w)^{-1}\, S_b$, afin de trouver une ***base de vecteurs propres*** qui formeront G. *Chaque colonne de G* sera alors appelée ***vecteur discriminant***.

3.6.2. Machine à Vecteurs de Support

3.6.2.1. Approche Générale

Les machines à vecteurs de support ou séparateurs à vastes marges (notées **SVM** pour *Support Vector Machines*) [Gun 98] ont été introduites lors de la conférence COLT en 1992

[Bos 92]. Elles s'appliquent aussi bien à des ***problèmes linéairement séparables*** que ***non séparables***.

Idée : En effet, l'idée principale des SVM est de reconsidérer le problème dans un espace de dimension supérieure, éventuellement de dimension infinie. Dans ce nouvel espace, il est alors probable qu'il existe un *hyperplan séparateur linéaire*. Si c'est le cas, *les* SVM *cherchent parmi l'infinité des hyperplans séparateurs celui qui maximise la marge entre les classes* (*cf*. figure 3.6).

On applique au vecteur d'entrée x une transformation non-linéaire Φ, afin de décrire les données dans un autre espace. L'espace d'arrivée $\Phi(x)$ est appelé **espace de redescription**. On cherche alors l'**hyperplan séparateur optimal**, d'équation $h(x) = \alpha\Phi(x) + \beta$, dans l'espace de redescription.

Procédé : Pour cela, on cherche ***les équations des hyperplans parallèles*** qui passent par **les vecteurs supports**, c'est-à-dire les attributs les plus proches de la frontière interclasse. On en déduit l'équation de l'hyperplan optimal, équidistant de ces hyperplans.

En pratique, on ne connaît pas la transformation Φ, on construit donc plutôt directement la fonction h, appelée ***fonction noyau***. Le *théorème de Mercer* explicite les conditions que h doit satisfaire pour être une fonction noyau : elle doit être *symétrique* et *semi-définie positive*.

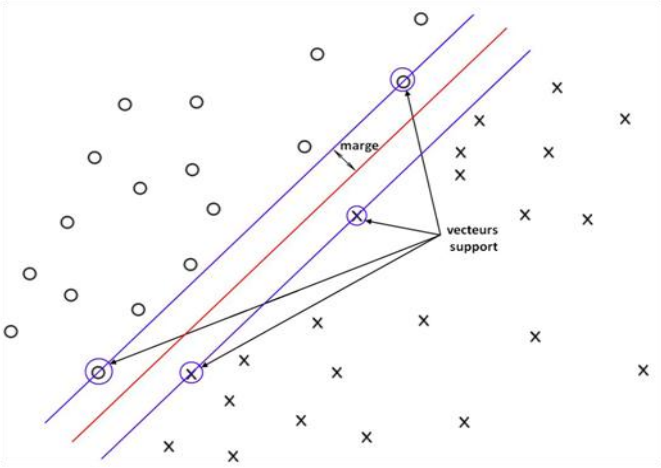


Figure 3.6. Principe du Séparateur à Vaste Marge (SVM)

Remarque. Lorsque l'on utilise les SVM, *le premier paramétrage* consiste donc *à choisir une fonction noyau*, parmi de nombreuses candidates : *Linéaire*, *polynomial*, *gaussien*, etc.

3.6.2.2. Problème du Perceptron

Prenons le cas d'un classifieur linéaire donné par :

$$y(x) = sign(\boldsymbol{w}.\boldsymbol{x} + \boldsymbol{b}) \qquad (3.5)$$

Graphiquement cela peut être représenté par :

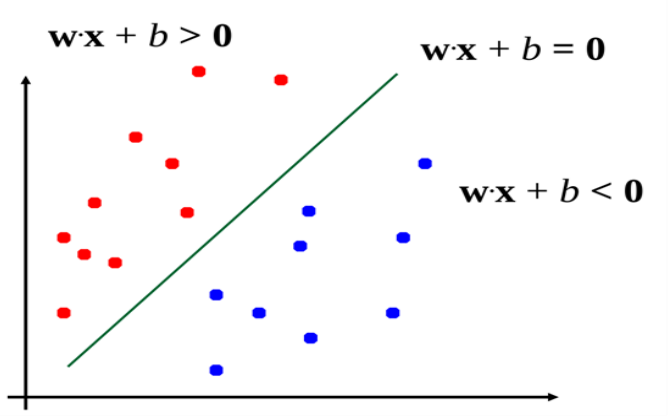


Figure 3.7. Classifieur Linéaire Simple.

Mais, puisqu'il existe de nombreux choix possibles pour $\boldsymbol{w}$ et $\boldsymbol{b}$:

$$y(x) = sign(\boldsymbol{w}.x + \boldsymbol{b}) = sign(\boldsymbol{k}w.x + \boldsymbol{k}.b) \qquad (3.6)$$

Problème de choix : Quel est le meilleur classifieur ?

3.6.2.3. Cas Séparable

- **Notion de Marge**

Dans le cas linéairement séparable, on va considérer les ***points*** les ***plus près*** *de* ***l'hyperplan*** séparateur appelés : **vecteurs supports** (support vectors, en anglais).

Pour tout point de l'espace des exemples, la distance à l'hyperplan séparateur, représentée par la figure 3.9, est donnée par :

$$\boldsymbol{r} = \frac{|\boldsymbol{w}.\boldsymbol{x}+\boldsymbol{b}|}{\|\boldsymbol{w}\|} \qquad (3.7)$$

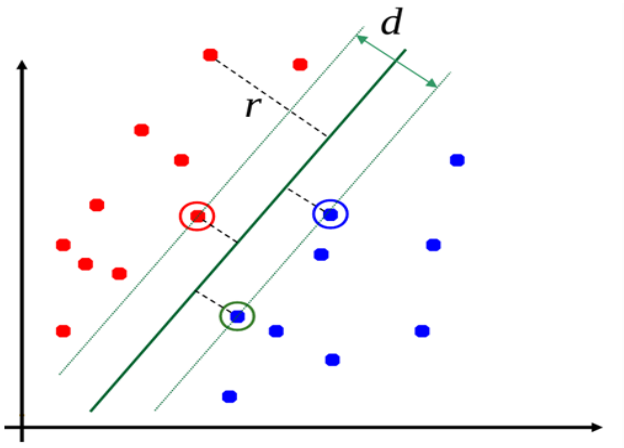


Figure 3.9. Vecteurs Supports et Distance à l'Hyperplan Séparateur.

Ainsi, on appelle **marge** $\boldsymbol{d}$ la distance entre les 2 classes. C'est cette distance $\boldsymbol{d}$ qu'on souhaiterait ***maximiser***.

- **Quantification de la Marge**

Pour limiter l'espace des ''*possibles*'', on ne considère que les points les plus proches, ceux qui sont situés sur les *hyperplans canoniques* donnés par :

$$\boldsymbol{w}.x + \boldsymbol{b} = \pm 1 \tag{3.8}$$

Dans ce cas, la marge est :

$$\boldsymbol{d} = \frac{\mathbf{2}}{\|\boldsymbol{w}\|} \tag{3.9}$$

Ainsi, les conditions d'une bonne classification seront :

$$\begin{cases} \boldsymbol{w}.x + \boldsymbol{b} \geq 1, \ si\ y_i = 1 \\ \boldsymbol{w}.x + \boldsymbol{b} < 1, \ si\ y_i = -1 \end{cases} \tag{3.10}$$

- **Maximisation de la Marge**

Le problème revient alors à trouver $\boldsymbol{w}$ et $\boldsymbol{b}$ tels que $\boldsymbol{d} = \frac{\mathbf{2}}{\|\boldsymbol{w}\|}$ est *maximale* $\forall\ (x_i, y_i)$, sous les contraintes données par l'équation (3.10) :

$$\begin{cases} \boldsymbol{w}.x + \boldsymbol{b} \geq 1, \ si\ y_i = 1 \\ \boldsymbol{w}.x + \boldsymbol{b} < 1, \ si\ y_i = -1 \end{cases}$$

De façon équivalente, le problème peut s'écrire plus simplement comme la ***minimisation*** de :

$$\frac{\mathbf{1}}{\mathbf{2}}\|\mathbf{w}\|^{\mathbf{2}} \tag{3.11}$$

Sous les contraintes : $y_i(\boldsymbol{w}.x + \boldsymbol{b}) \geq 1, \ \forall\ i \in [1, N]$.

Cette minimisation est possible sous les conditions dites de "Karush-Kuhn-Tucker (KKT)" :

Soit le Lagrangien $\boldsymbol{\mathcal{L}}$:

$$\boldsymbol{\mathcal{L}}(\boldsymbol{w}, \boldsymbol{b}, \boldsymbol{\lambda}) = \frac{\mathbf{1}}{\mathbf{2}}\|\mathbf{w}\|^{\mathbf{2}} - \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}} \boldsymbol{\lambda}_i[y_i(\boldsymbol{w}.x_i + \boldsymbol{b}) - \mathbf{1}]$$

Les conditions de KKT sont alors :

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{w}} = 0\ ; \frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{b}} = 0\ ; \frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\lambda}_i} \geq 0, \boldsymbol{\lambda}_i \geq 0$$

$$\boldsymbol{\lambda}_i[y_i(\boldsymbol{w}.x_i + \boldsymbol{b}) - \mathbf{1}] = \mathbf{0}$$

Par ailleurs cette dernière condition implique que pour tout point ne vérifiant pas $y_i(\boldsymbol{w}.x_i + \boldsymbol{b}) = \mathbf{1}$, le $\boldsymbol{\lambda_i}$ est nul.

Les **points** qui vérifient $y_i(\boldsymbol{w}.x_i + \boldsymbol{b}) = \mathbf{1}$, sont appelés **vecteurs supports**. Ce sont les *points les plus près de la marge*. Ils sont sensés être peu nombreux par rapport à l'ensemble des exemples.

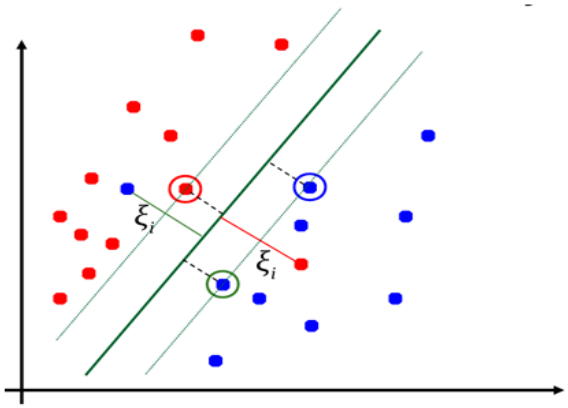


Figure 3.10. Classification à Marges Souples pour un Problème non Linéairement Séparable.

3.6.2.4. Applications des SVM

Les avantages théoriques (minimisation de l'erreur empirique et structurelle) et pratiques (algorithmes optimisés) des SVM en ont fait un outil très prisé dans de nombreux problèmes pratiques de classification [Can 07].

Dans bien des cas, il s'agit de construire un noyau (donc une mesure de similarité) adapté aux données à traiter.

Exemples d'applications,

- Classification de données biologiques/physiques, Classification de documents numériques.
- Reconnaissance d'expressions faciales, Classification de textures, E-learning.
- Détection d'intrusion, Reconnaissance de la parole.
- Reconnaissance d'Image Basée Contenu (CBIR : Content Based Image Retrieval).

3.6.3. Réseaux de Neurones

Les réseaux de neurones artificiels sont construits à partir du *modèle biologique des neurones*. Leurs fondements proviennent des premiers travaux sur les réseaux de neurones [Let 59] qui conçurent un modèle simplifié du neurone biologique, appelé le *neurone formel*. Ce dernier peut être représenté comme un automate à n entrées et une sortie.

Le premier modèle de réseau de neurones complet a été nommé ***Perceptron*** [Ros 58] (*cf.* figure 3.7). Dans ce modèle, une *fonction de combinaison associe les entrées du neurone* (par

exemple en les sommant), puis compare ce résultat à une ***valeur seuil***, via une ***fonction d'activation***.

Typiquement, la fonction d'activation renvoie une ***valeur binaire***, relativement au fait qu'elle ait été ***activée*** *ou* ***pas***.

Les paramètres importants de ce modèle sont les **coefficients synaptiques** (c'est-à-dire les **poids**), le **seuil**, et **la façon de les ajuster** lors de l'apprentissage.

En effet, il faut choisir un **mécanisme** permettant de calculer ces paramètres et de les **faire converger** vers une valeur assurant une classification aussi proche que possible de l'optimale, lors de la phase d'apprentissage.

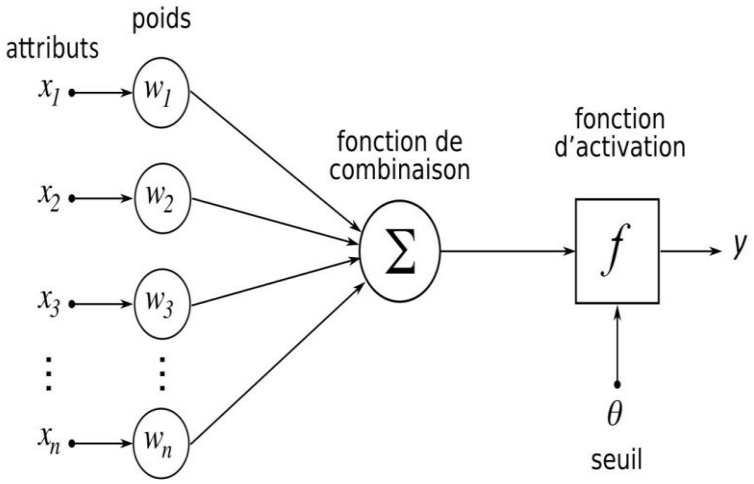


Figure 3.7. Schéma du perceptron

On peut représenter la sortie du réseau par l'équation suivante (avec ***f*** la fonction d'activation):

$$y = f\left(\sum_{i=1}^{n}(w_i x_i) - \theta\right) \qquad (3.8)$$

Remarque. Dans sa première version, le neurone formel était implémenté avec une ***fonction à seuil*** (pour la fonction d'activation), mais de nombreuses versions existent (fonction linéaire par morceaux, sigmoïde, Gaussienne, etc.), pouvant ainsi fournir plusieurs valeurs de sortie.

Amélioration : Le modèle du **perceptron multicouche** (MLP, Multi-Layer Perceptron, en anglais) [Rum 86] reprend le principe du perceptron, en ajoutant plusieurs **vecteurs de poids** (appelées **couches cachées**), afin d'augmenter les combinaisons possibles (*cf.* figure 3.8).

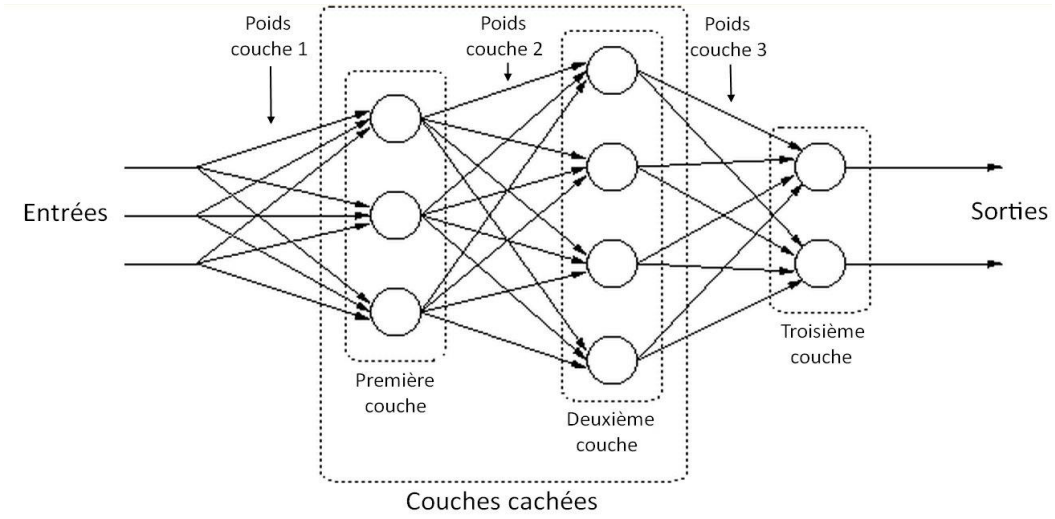


Figure 3.8. Schéma du perceptron multi couches. Source : [Ter 03]

Les neurones d'une couche sont reliés aux neurones des couches adjacentes par des **liaisons pondérées**. Ainsi, **le poids** de chacune de ces liaisons est l'*élément clef du fonctionnement du réseau*.

Remarque : On peut considérer que la fonction d'activation est la même à chaque couche, mais qu'elle varie d'une couche à l'autre.

Chapitre 4

Apprentissage et Classification non Supervisés

Chapitre 4
Apprentissage et Classification Non-Supervisés

4.1. Introduction

Il s'agit d'une tâche principale en intelligence artificielle et dans la fouille exploratoire de données. C'est une technique d'analyse statistique des données très utilisée dans de nombreux domaines y compris l'apprentissage automatique, la reconnaissance de formes, le traitement d'images, la recherche d'information, etc.

L'idée est donc de ***découvrir des groupes au sein des données***, de façon automatique.

4.1.1. Types de Données

Les données traitées en classification automatique peuvent être de différents formats et peuvent provenir de différentes sources : images, signaux, textes (caractères, fichiers, attributs, …), sons, symboles ; ou encore toutes autres valeurs de paramètres obtenues par d'autres types de mesures.

Par exemple, en reconnaissance des objets dans les images, les informations collectées sur les objets à traiter seront des données multidimensionnelles, telles les couleurs d'une zone de l'image ou les variations d'intensité lumineuse pour les images infrarouges.

Exemple. Une image couleur contenant $\boldsymbol{n}$ lignes et $\boldsymbol{m}$ colonnes et donc $\boldsymbol{n}$ **x** $\boldsymbol{m}$ *pixels couleurs*. Chaque pixel, composé de trois composantes couleur : R (rouge), V (verte) et B (bleue), peut être modélisé par une structure "*pixel*" composée au moins des champs : ‘‘couleur(R, V, B)’’ et "label" ; l'image quant à elle sera un tableau de $\boldsymbol{n}$ **x** $\boldsymbol{m}$ *structures pixel*.

4.1.2. Qu'est ce qu'une Classe ?

Une classe (ou groupe) est un ensemble formé par des données **homogènes** : qui "se ressemblent" au sens d'un critère de similarité (distance, densité de probabilité, etc.).

4.2. Algorithme des Centres Mobiles

Proposé par Lloyd [Llo 57], l'algorithme s'avère très utile pour la classification de données uni- ou multidimensionnelles.

$\boldsymbol{K-means}$ est un algorithme de minimisation alternée qui, étant donné un entier $\boldsymbol{K}$, va chercher à séparer un ensemble de points en $\boldsymbol{K}$ *clusters* ou groupes (voir Figure 4.1).

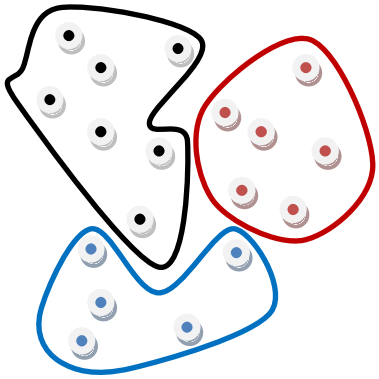


Figure 4.1 Clustering sur un Ensemble de Points (Objets) 2D, en 3 Clusters.

4.2.1. Combien de Classes ?

Le nombre ***K*** représente le **nombre de classes** que l'algorithme doit former à partir des propriétés des échantillons ou exemples.

Le nombre de groupes *K* peut être supposé fixe (donné par l'utilisateur) ou fixé par la nature du problème à traiter. C'est le cas, par exemple, si l'on s'intéresse à classer des images de chiffres manuscrits (nombre de classes = 10 : 0, …, 9) ou de lettres manuscrites (nombre de classes = nombres de caractères de l'alphabet), etc.

4.2.2. Données, Classes et Métrique

Exemple. Considérons une image couleur. Une image contient n pixels $(x_1, \dots, x_n)$, chaque pixel x_i contient 3 valeurs (R : rouge, G : vert, B : bleu). On peut donc représenter le $i^{ème}$ pixel ($i = 1, \dots, n$) par un vecteur x_i de dimension $d = 3$: $x_i = (x_{i1}, x_{i2}, x_{i3})^T \in \{0, 1, \dots, 255\}^3$.

Ainsi, si le problème de classification consiste à répartir les données pixel en 3 classes : rouge, verte, bleue. L'idée revient à mesurer la proportion (métrique en taux) de chaque composante de couleur (R%, V% et B%) contenue dans chaque pixel afin de pouvoir l'affecter à la classe d'appartenance.

Mais, on peut aussi procéder de la manière suivante : Si l'on connaît les classes de certains pixels, on pourra prédire les classes des autres pixels en choisissant une mesure de (dis)**similarité** ou de (dis)**ressemblance**, par exemple une *simple distance*, ou encore une *mesure de probabilité*, etc. Chaque pixel à classer aura donc *la classe de celui qui lui est* ***le plus proche*** au sens de la mesure de (dis)similarité choisie.

Remarque. Ceci est très utilisé dans des domaines telle la segmentation d'image.

De manière générale, on peut représenter les données comme un ensemble de vecteurs $(x_1, \dots, x_n)$, chaque x_i est composée de d composantes réelles : $x_i = (x_{i1}, \dots, x_{ij}, \dots, x_{id})^T \in \mathbb{R}^d$.

4.2.3. Exemple Introductif

Afin d'illustrer le fonctionnement de l'algorithme $\boldsymbol{K-means}$, considérant l'ensemble des données suivant, consistant en 7 objets représentés chacun par un descripteur à 2 paramètres :

Sujets	X	Y
1	2,0	2,0
2	3,0	4,0
3	6,0	8,0
4	10,0	14,0
5	7,0	10,0
6	9,0	10,0
7	7,0	9,0

On veut grouper ces données (selon leurs similarités) en deux (k=2) groupes : G_1 et G_2.

On procèdera de la manière suivante :

1. A partir des $\boldsymbol{n}$ données $(x_1, x_2, \dots, x_n)$, choisir (au hasard) les $\boldsymbol{k=2}$ centres $\mu = (\mu_1, \mu_2, \dots, \mu_k)$ des $\boldsymbol{k}$ **groupes** initiaux $G = (G_1, G_2, \dots, G_k)$ à générer.
2. A partir de l'itération $\boldsymbol{t=0}$, et pour chaque objet x_i ($\boldsymbol{i}$ = 1 .. *n*) :
   a. Calculer sa distance $dist(x_i, \mu_j)$ de x_i à chaque centre μ_j ($\boldsymbol{j}$ = 1 ... *k*),
   b. Affecter ou réaffecter x_i au groupe G_l de centre μ_l (qu'est le plus proche à x_i), si $dist(x_i, \mu_l) = minimale$.
3. Recalculer le centre μ_j (la moyenne) de chaque groupe G_j ($\boldsymbol{j}$ = 1 … *k*),

1ère étape : Initialisation (**itération 0**) : Recherche d'une partition initiale.

Ça consiste à choisir les centres initiaux des 2 groupes. Prenant, par exemple, les objets 1 et 4 (les plus éloignés –*min* et *max*, selon une distance ; optant pour la distance Euclidienne) comme centres de G_1 et G_2.

Itération 1ère:

2ème étape : Les objets restants sont examinés un par un et localisés par rapport au plus proche cluster.

Ceci nous mènent à calculer d'abord les distances de chaque objet aux 2 groupes G_1 et G_2, représentés respectivement par leurs centres ou centroïdes $\boldsymbol{C_1}$ et $\boldsymbol{C_2}$; ces distances sont données par le tableau suivant (les valeurs des distances sont arrondîtes, et on ne retient qu'un seul chiffre après la virgule) :

3ème étape : Une fois que les distances sont connues, on réaffecte chaque objet au cluster $\boldsymbol{G_i}$, si sa distance au centre $\boldsymbol{C_i}$ est minimale.

Puisque, on n'est pas sûr que chaque objet soit assigné au **bon cluster**, ceci nous oblige à **réitérer** une fois de plus sur **l'étape 2** : ce qui revient à recalculer les distances de chaque objet aux 2 nouveaux centres $\boldsymbol{C_1}$ et $\boldsymbol{C_2}$. Puis, réaffecter chaque objet au groupe dont la distance à son centre est minimale.

Itération 2ème:

Recalculons les distances de chaque objet aux 2 groupes G_1 et G_2.

Une fois les distances recalculées, on réaffecte chaque objet au cluster $\boldsymbol{G_i}$ qui lui est le plus proche. Ici, seul l'objet 3 est reclassé dans G_2. Puis, on recalcule les barycentres des nouveaux groupes.

Itération 3ème:

Recalculons les distances de chaque objet aux 2 groupes G_1 et G_2 ; on obtient :

On recalcule les barycentres des nouveaux groupes, mais ce n'est pas nécessaire puisque les groupes n'ont pas changé, ainsi, il y a ***convergence***.

Condition de Convergence :

L'itération s'arrête lorsque chaque objet est plus proche de son propre moyen cluster que celui des autres groupes, et la solution finale des clusters sera ce dernier partitionnement.

4.2.4. Approche de Classification

4.2.4.1. Notations

On utilise les notations suivantes :

- Les $x_i \in \mathbb{R}^d$, $i \in \{1, .., n\}$ sont les points (objets) à séparer.
- Les z_i^k sont des variables indicatrices associées aux x_i telles que $z_i^k = 1$ si x_i appartient au cluster k, $z_i^k = 0$ sinon.
- z est la matrice des z_i^k.
- μ est le vecteur des $\mu_k \in \mathbb{R}^d$, où μ_k est le centre du cluster k.

4.2.4.2. Mesure de Distorsion

L'objectif de l'algorithme des centres mobiles ou *K-means* pour la classification automatique d'un ensemble de données (exemples) $(x_1, x_2, \dots, x_n)$, $x_i \in \mathbb{R}^d$, est de **minimiser le critère d'erreur** (dite aussi **erreur de distorsion** $\mathcal{J}(\mu, z)$) suivant, par rapport **aux centres des classes** $\psi = \mu = (\mu_1, \dots, \mu_K)$ et **les classes** (partitions) $z = (z_1, \dots, z_K)$:

$$\mathcal{J}(\psi, z) = \mathcal{J}(\mu, z) = \mathcal{J}(\mu_1, \dots, \mu_K, z) = \sum_{k=1}^{K} \sum_{i=1}^{n} z_{ik} \|x_i - \mu_k\|^2 \qquad (4.1)$$

avec, $\boldsymbol{z_{ik}}$ est une variable binaire qui vaut **1** si la classe du $i^{ème}$ exemple x_i est $\boldsymbol{k}$ et **0** sinon. $\boldsymbol{z}$ est une matrice binaire.

La **distorsion** ou le **critère** $\mathcal{J}(\psi, z)$ correspond à la ***distance euclidienne totale*** entre chaque donnée x_i et le centre μ_{z_i} dont elle est la plus proche (qui doit être minimale) au sens de la distance Euclidienne :

$$\|x_i - \mu_k\|^2 = d(x_i, \mu_k) = \sqrt{\sum_{j=1}^{d}(x_{ij} - \mu_{kj})^2} \tag{4.2}$$

4.2.5. Etapes de l'Algorithme *K*-means

Le but de l'algorithme est de minimiser $\mathcal{J}(\mu, z)$, il se présente sous la forme d'un algorithme de minimisation alternée. L'algorithme *K-means* est composé des trois étapes suivantes :

1. **Etape Initialisation** : choisir le vecteur μ. On initialise les ***K*** centres des classes $(\mu_1^{(0)}, \dots, \mu_K^{(0)})$ (au choix, plusieurs options d'initialisation) pour donner le pas de départ de l'algorithme (par exemple, choisir aléatoirement les ***K*** centres ou données parmi les données à traiter comme centres des clusters). Il s'agit donc de démarrer à l'itération **t = 0** avec des valeurs initiales $(\mu_1^{(0)}, \dots, \mu_K^{(0)})$ pour les paramètres du modèle.

2. **Etape d'affectation (classification)** : Chaque donnée (exemple) x_i est assignée à la classe c_k du centre dont elle est la plus proche : $\forall$ *i = 1, …, n* :

$$z_{ik}^{(t)} = \begin{cases} 1 & si\ k = arg\ min_{z\in\{1,..,K\}}\|x_i - \mu_z\|^2 \\ 0 & sinon. \end{cases} \tag{4.3}$$

   Ainsi, la classe : $c_k = \{x_i / k = arg\ min_k \|x_i - \mu_k\|^2\}$ est l'ensemble des exemples les plus proches de μ_k.

   Ceci revient à minimiser $\mathcal{J}$ par rapport à z : $z_i^k = 1$ pour $k \in arg\ min\|x_i - \mu_k\|$, c'est-à-dire, on associe à x_i le centre μ_k le plus proche.

3. **Etape de recalage des centres** : le centre $\boldsymbol{\mu}$ de chaque classe ***k*** est recalculé comme étant la moyenne arithmétique de toutes les données appartenant à cette classe (suite à l'étape d'affectation précédente) : $\forall$ ***k*** *= 1, …, K* :

$$\boldsymbol{\mu}_k^{(t+1)} = \frac{\sum_{i=1}^{n} z_{ik}^{(t)} x_i}{\sum_{i=1}^{n} z_{ik}^{(t)}} \tag{4.4}$$

   t étant l'itération courante.

   Ce qui revient à minimiser $\mathcal{J}$ par rapport à μ : $\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{n} z_i^k x_i}{\sum_{i=1}^{n} z_i^k}$

   Les nouveaux centres μ_k recalculés des clusters c_k sont la moyenne de l'ensemble c_k :

$$\mu_k = \frac{\sum_{i\in c_k} x_i}{|c_k|} \tag{4.5}$$

Réitérer les étapes (2) et (3) jusqu'à convergence ou équilibre.

Remarques

- L'étape de minimisation par rapport à z revient à répartir les x_i selon les cellules de *Voronoï* dont les centres sont les μ_k.
- Dans l'étape de minimisation selon μ, μ_k est obtenu en annulant la *k-ième* coordonnée du gradient de $\mathcal{J}$ selon μ.

4.2.6. Propriétés du *K*-means

- C'est un algorithme de regroupement simple et rapide, mais aussi très utilisé.
- La méthode *k-means* **minimise** une *mesure de dissemblance intra-classe* pour les *k-groupes*.
- Chaque objet est affecté au cluster dont le centre (centroîde/barycentre) est le plus proche.
- Le centre d'un groupe est la moyenne de tous les points (éléments) de ce groupe.
- Son inconvénient est qu'il produit *un résultat* **différent** à chaque exécution (initialisation).

4.2.6.1. Choix de *K*

Le choix de K n'est pas universel, on remarque que si on augmente K, *la distorsion* ***diminue***, et ***s'annule*** lorsque *chaque point est centre de son cluster*.

Pour pallier à ce phénomène il possible de rajouter un terme en fonction de K dans l'expression de $\mathcal{J}$, mais là encore son choix est arbitraire.

4.3. Classification Hiérarchique Ascendante

La classification hiérarchique ou (re)groupement hiérarchique (ou *clustering*) est une méthode de classification automatique qui consiste à effectuer une **suite de regroupements** en **agrégeant** à chaque étape les **objets** (données ou descripteurs d'objets) ou les **groupes d'objets** les plus proches.

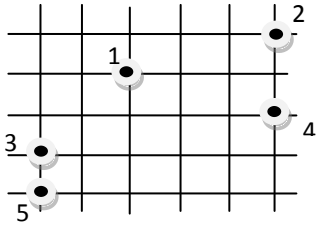
On trouve des applications dans des domaines très divers tels que la biologie (classements par espèce, genre), l'archéologie, le traitement d'images, le traitement de requêtes, etc.

4.3.1. Exemple introductif

Cet exemple est inspiré de [Bel 92].

Soit un ensemble d'objets représentés par des points numérotés de 1 à 5, dans un repère euclidien.

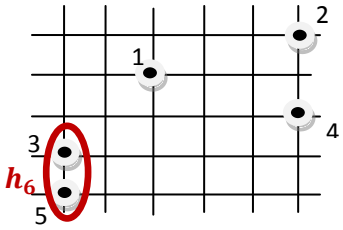
Notons $\boldsymbol{d}$ la distance euclidienne mesurée entre les objets.



d	1	2	3	4	**5**
1	0	$\sqrt{10}$	$\sqrt{8}$	$\sqrt{10}$	$\sqrt{13}$
2		0	$\sqrt{34}$	2	$\sqrt{41}$
3			0	$\sqrt{26}$	**1**
4				0	$\sqrt{29}$
5					0

1ère étape : Cette première étape de la méthode nous conduit à regrouper les **objets 3 et 5**, qui sont les points les **plus proches** (distance minimale), $\boldsymbol{d = 1}$, et à former un groupe $h_6 = \{3, 5\}$.

A ce groupe est associé son ***niveau***, ou ***indice d'agrégation f***, qui est la distance entre ses deux sous-groupes $\{3\}$ et $\{5\}$, $f(h_6) = 1$.

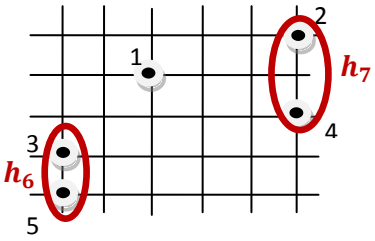


Plusieurs solutions sont possibles. Nous en proposons deux à titre d'exemples :

1. **Le saut minimal** : qui consiste à affecter à la distance entre deux groupes la distance entre leurs objets **les plus proches**, et
2. **Le diamètre maximal** : qui consiste à retenir la distance entre leurs objets **les plus éloignés**.

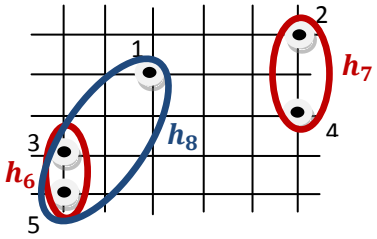
2ème étape : A l'étape suivante, on a la même agrégation pour les deux distances $\boldsymbol{d = 2}$ (même distance minimale), $h_7 = \{2, 4\}$, $f(h_7) = 2$:

Saut minimal			
d	1	h_6	h_7
1	0	$\sqrt{8}$	$\sqrt{10}$
h_6		0	$\sqrt{26}$
h_7			0



3ème étape : A la troisième étape, les deux hiérarchies deviennent différentes :

- ***Saut minimal :*** $h_8 = \{1\} \cup h_6 = \{1,3,5\}, f(h_8) = \sqrt{8}$



Saut minimal		
d	h_7	h_8
h_7	0	$\sqrt{10}$
h_8		0

Si on continue à la dernière étape ($4^{ème}$ itération de regroupement), tous les objets sont regroupés :

- ***Saut minimal :*** $h_9 = h_7 \cup h_8 = \{1,2,3,4,5\}, f(h_9) = \sqrt{10}$

Remarque important : Le ***choix de la distance*** entre deux groupes influe sur les regroupements.

4.3.2. Dendrogramme

Cette hiérarchie de regroupement (ou de *clustering*, en anglais) des objets peut être représentée par un diagramme dit : ***dendrogramme***. C'est une représentation arborescente d'une hiérarchie.

La hiérarchie que nous avons obtenue par le critère de ***saut minimal*** dans l'exemple introductif, est représentée par l'arbre ou le dendrogramme de la figure 4.2 :

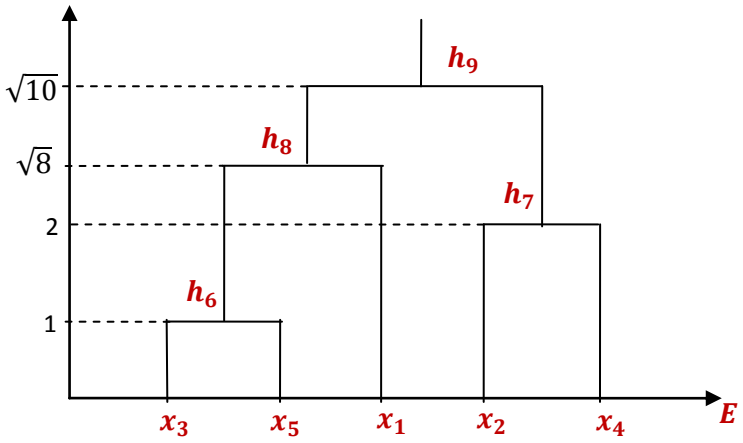


Figure 4.2. Dendrogramme : Hiérarchie obtenue par le saut minimal.

Par définition, une hiérarchie indicée ***H*** est isomorphe à un arbre dont les nœuds sont associés aux éléments de ***H*** et la relation ''***fils de***'', à la relation de borne supérieure pour l'inclusion.

Les feuilles représentent les objets et la racine l'ensemble ***E***. Deux nœuds h_1 et h_2 sont ou bien sur deux branches différentes $(h_1 \cap h_2)$, ou bien sur une même branche, l'un étant alors inclut dans l'autre $(h_1 \subset h_2 \ \ ou\ h_2 \subset h_1)$.

De plus l'ordonnée d'un nœud correspond à son indice ***f***. Elle est croissante des feuilles vers la racine et les feuilles sont d'ordonnée nulle.

4.3.3. Etapes du Regroupement Hiérarchique

D'une manière générale, l'algorithme de clustering ou (re)groupement hiérarchique ascendant peut comprendre plusieurs **itérations**, dont le nombre maximal d'itérations est donné par le ***nombre*** et/ou la ***nature*** des objets à regrouper.

Cependant, on peut décider de l'arrêt du regroupement (convergence) par un ***seuil de regroupement*** selon le ***niveau d'agrégation désiré***.

Chaque itération de l'algorithme de classification par clustering hiérarchique comprend :

1. Calcul de la matrice des distances entre objets, pour la 1ère étape (ou mise à jour de la matrice des distances entre objets, pour les étapes suivantes).
2. Déterminer la distance minimale (niveau d'agrégation) et repérer les groupes ou objets concernés.
3. Tester si le niveau d'agrégation n'a pas dépassé le seuil ou le nombre d'itérations est suffisant, sinon arrêt.
4. Regrouper les objets ou groupes concernés. Réitérer sur 1.

4.3.4. Recherche d'une Ultramétrique à partir d'une Hiérarchie Indicée

Soit H une hiérarchie indicée d'indice f.

Soit la fonction $\boldsymbol{\alpha} : E \text{ x } E \rightarrow H$ qui, à tout couple (a, b) de E x E, associe le plus petit élément de H contenant à la fois a et b.

On pose $u(a,b) = f\big(\alpha(a,b)\big)$, ainsi ***u retourne le niveau du premier ancêtre commun aux deux individus*** dans l'arbre de la hiérarchie. Alors u est une ultramétrique [Bel 92] :

- $u(x,x) = 0$ car $\alpha(x,x) = \{x\}$ et $f(\{x\}) = 0$;
- $u(x,y) = u(y,x)$ car $\alpha(x,y) = \alpha(y,x)$;
- Il reste à démontrer que $u(x,y) \leq sup\big(u(x,z), u(y,z)\big)$.

Posons $h_1 = \alpha(x,y)$; $h_2 = \alpha(x,z)$; $h_3 = \alpha(y,z)$ et raisonnons par l'absurde en supposant :

$$\begin{cases} u(x,y) > u(x,z) \\ u(x,y) > u(y,z) \end{cases} \quad \text{c'est-à-dire} \quad \begin{cases} f(h_1) > f(h_2) \\ f(h_1) > f(h_3) \end{cases}$$

or h_1 et h_2 contiennent x, c'est-à-dire $h_1 \cap h_2 \neq \emptyset$. D'où on a, soit $h_1 \subset h_2$, soit $h_2 \subset h_1$.

- Cas où $h_1 \subset h_2$: d'après la définition de la hiérarchie : $f(h_1) \leq f(h_2)$, d'où une contradiction avec les hypothèses ;
- Cas où $h_2 \subset h_1$: ceci conduit de nouveau à deux sous-cas possibles qui sont $h_2 \subset h_3$ ou $h_3 \subset h_2$;
- Cas où $h_2 \subset h_3$: ceci entraîne que h_3 contient $\{x, y, z\}$. Or h_1 est le plus petit élément de H contenant $\{x, y\}$, ce qui entraîne que $h_1 \subset h_3$, c'est-à-dire $f(h_1) \leq f(h_3)$, d'où contradiction ;
- Cas où $h_3 \subset h_2$: le même raisonnement que précédemment conduit à la conclusion contradictoire $h_1 \subset h_2$.

4.3.5. Recherche d'une Hiérarchie Indicée à partir d'une Ultramétrique

Le processus de construction de la hiérarchie à partir d'une ultramétrique u est récursif. Il fonctionne comme suit [Bel 92] :

1. On forme l'ensemble des parties à un élément : $\{x_1\}, \{x_2\}, \ldots \{x_n\}$.
2. On agrège les deux éléments h_i et h_j les plus proches au sens de u, la distance ultramétrique reste stable par cette agrégation (en effet, pour tout nœud h_k, $\{h_i, h_j, h_k\}$ forme un triangle isocèle dont la base est $\{h_i, h_j\}$).
3. On réitère l'étape 2 sur $\{h_1, h_2, \ldots, \{h_i, h_j\}, \ldots, h_p\}$ et ceci jusqu'à ce que tous les éléments soient regroupés en une seule classe.

Exemple, soit l'ultramétrique u :

u	1	2	3	4	5
1	0	8	8	8	5
2		0	2	4	8
3			0	4	8
4				0	8
5					0

On agrège $\{2\}$ et $\{3\}$ et on recommence avec $\{\{1\},\{2,3\},\{4\},\{5\}\}$.

Pour obtenir la distance entre les classes $\{1\}$ et $\{2,3\}$, on remarque que, u étant une ultramétrique et $\{2,3\}$ étant les points les plus proches, $\{1,2,3\}$ est un triangle isocèle de base $\{2,3\}$, donc $u(1,2) = u(1,3)$.

Il suffit ainsi de poser $u(1,\{2,3\}) = u(1,2)$. Cette distance étendue aux nœuds reste une ultramétrique.

u	1	$\{2,3\}$	4	5
1	0	8	8	5
$\{2,3\}$		0	4	8
4			0	8
5				0

On agrège $\{2,3\}$ et $\{4\}$:

u	1	$\{2,3,4\}$	5
1	0	8	5
$\{2,3,4\}$		0	8
5			0

On agrège $\{1\}$ et $\{5\}$:

u	$\{1,5\}$	$\{2,3,4\}$
$\{1,5\}$	0	8
$\{2,3,4\}$		0

On agrège enfin $\{1,5\}$ et $\{2,3,4\}$. On obtient ainsi la hiérarchie de la figure 4.3.

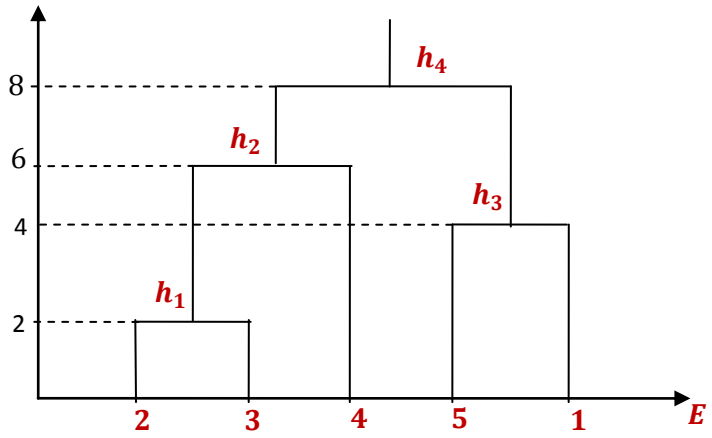


Figure 4.3. Dendrogramme : Hiérarchie obtenue à partir de l'ultramétrique u [Bel 92].

4.3.5.1. Exemple : Classification de consonnes en reconnaissance de la parole

Cet exemple illustre une classification par regroupement hiérarchique de consonnes en reconnaissance de la parole, extrait de [Bel 92].

Il s'agit de déterminer les ressemblances entre seize (16) consonnes, telles qu'elles sont perçues par l'oreille, ceci sans faire intervenir des connaissances sur la similarité de leur forme, sur leur spectre de fréquence, etc. [Bou 85].

Les données sont rassemblées au cours de séances d'expérimentation, où un individu prononce une consonne, au hasard, dans un environnement vocalique et le son émis est dégradé de différentes manières : en lui superposant un bruit blanc, en le faisant passer à travers un filtre de fréquence, etc. Des auditeurs notent le son qu'ils perçoivent et confondent certaines consonnes.

Le tableau suivant donne la *matrice de confusion des consonnes* où *chaque nombre indique la fréquence avec laquelle une consonne a été prise pour une autre*. Ce nombre est appelé *indice de confusion*. Lorsque *deux consonnes sont proches*, l'*indice de confusion est élevé*.

p											
t	22										
k	43	24									
f	10	5	7								
s	5	5	6	6							
b	2	1	1	4	2						
d	2	2	2	2	3	5					
g	1	1	3	1	3	6	34				
v	1	2	2	3	2	21	5	5			
z	2	2	2	1	3	5	10	13	8		
m	2	2	2	1	1	3	2	3	3	1	
n	1	1	2	1	1	2	3	3	2	1	15
	p	**t**	**k**	**f**	**s**	**b**	**d**	**g**	**v**	**z**	**m**

On voit, par exemple, que les consonnes ***b*** et ***d*** sont confondues dans **5%** des cas, et que ***b*** et ***v*** sont confondues dans **21%** des cas.

A partir de ce tableau, on construit le dendrogramme de la figure 4.4.

La méthode utilisée regroupe en classes les consonnes qui sont les plus proches (***k*** et ***p*** par exemple) ; on considère ensuite cette classe comme une consonne fictive dont l'indice de proximité avec toutes les autres consonnes est recalculé : l'indice de la classe {k, p} avec une autre consonne, ***s*** par exemple, est le plus petit des deux indices de ***k*** avec ***s*** et de ***p*** avec ***s***, c'est-à-dire le minimum du couple (6, 5), soit 5. Sur cet arbre ou dendrogramme, le niveau d'un nœud est le plus petit indice entre les consonnes d'une même classe.

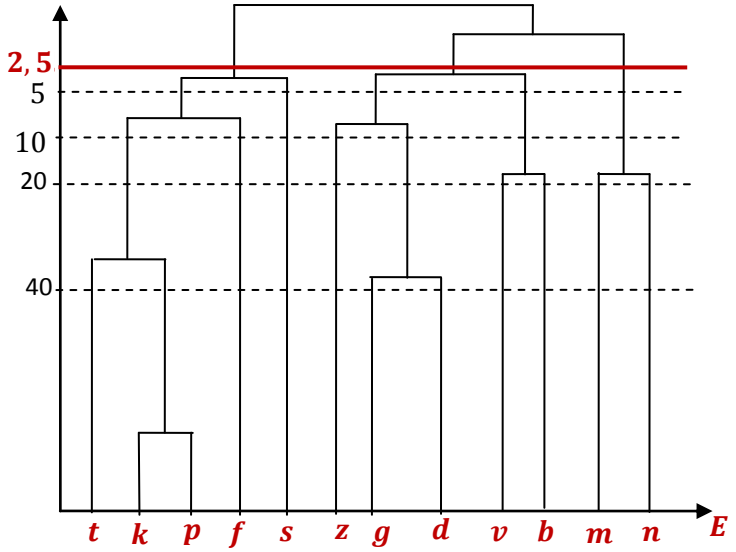


Figure 4.4. Dendrogramme : Regroupement Hiérarchique des Consonnes [Bou 85].

En coupant en 2,5, on distingue trois grandes classes de consonnes : *nasales*, *sourdes non nasales* et *sonores*, tel décrit sur la figure 4.5.

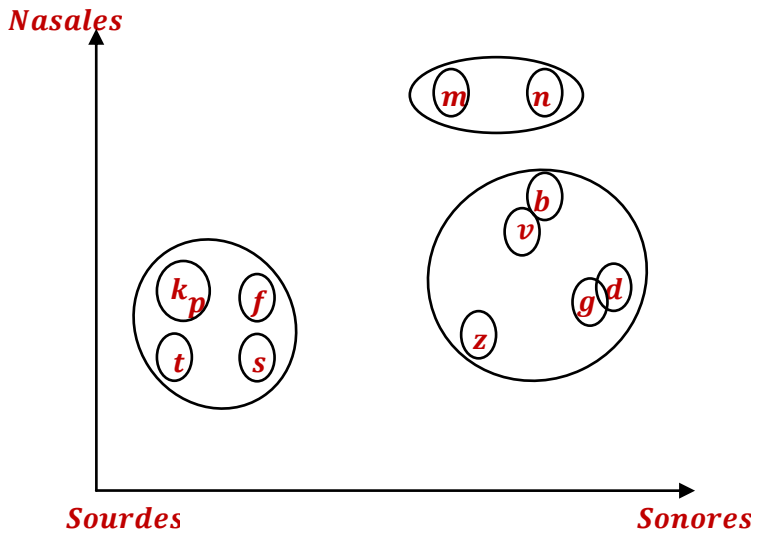


Figure 4.5. Classification Hiérarchique des Consonnes, d'après [Bou 85].

Chapitre 5

Notions Connexes à l'Apprentissage Artificiel

Chapitre 5
Notions Connexes à l'Apprentissage Artificiel

5.1. Introduction

L'analyse des données fournies par les bases de données (tels les fichiers et entrepôts de données) ou les moyens multimédias (telles les images, vidéo, etc.) est faite par des méthodes issues de la statistique et reprises par la fouille de données [Has 01] et/ou la vision par ordinateur.

L'analyse de ces données est un problème d'apprentissage supervisé, c'est-à-dire que l'on essaie de prédire le comportement de nouveaux cas, en apprenant à partir de cas d'entrainement déjà vécus ; ou même dans certaines circonstances d'apprentissage automatique non supervisé.

L'originalité : de certains problèmes vienne du fait que dans les données brutes il y a très peu de cas d'étude (d'exemples d'apprentissage) en comparaison du nombre d'individus dans le réel.

Dans la suite de ce chapitre, nous verrons que des méthodes d'optimisation pourraient être utilisées dans le cadre de la *sélection d'attributs*, et notamment pour les *techniques wrapper*. Ces méthodes ne sont pas toujours adaptées à la dimension du problème, car le nombre de variables peut varier de plusieurs milliers à des dizaines de milliers.

5.2. Phénomène de Sur-apprentissage

Ce phénomène est l'un des plus importants lorsque l'on travaille dans le domaine de l'apprentissage.

En effet, il arrive souvent que les exemples de la base d'apprentissage comportent des valeurs approximatives ou bruitées ou même redondantes. Si l'on oblige le modèle de prédiction à répondre de façon quasi parfaite relativement à ces exemples, *il devient biaisé par des valeurs erronées*.

Exemple, imaginons qu'on présente au modèle (ex., réseau de neurones) des couples $(x_i, f(x_i))$ situés sur une droite d'équation $y = ax + b$, *mais bruités* de sorte que les points ne soient pas exactement sur la droite.

- S'il y a un ***bon apprentissage***, le modèle de prédiction sera de la forme $ax + b$ pour toute valeur de x présentée.
- En revanche, *s'il y a* ***sur-apprentissage*** (***overfitting*** en anglais), le modèle répond un peu plus que $ax + b$ ou un peu moins, car chaque couple $(x_i, f(x_i))$ positionné en dehors de la droite va influencer la décision.

La théorie de la ***régularisation statistique*** introduite par Vladimir Vapnik (appelée ***théorie de Vapnik-Chervonenkis***) [Vap 95] permet d'anticiper, d'étudier et de réguler les phénomènes liés au sur-apprentissage.

La méthode la plus simple ***pour éviter ce phénomène*** est de partager la population d'apprentissage en deux sous-ensembles. Le premier sert à l'apprentissage et le deuxième sert à l'évaluation de l'apprentissage. *Tant que l'erreur obtenue sur le deuxième ensemble diminue, on peut continuer l'apprentissage, sinon on l'arrête* (cf. figure 5.1).

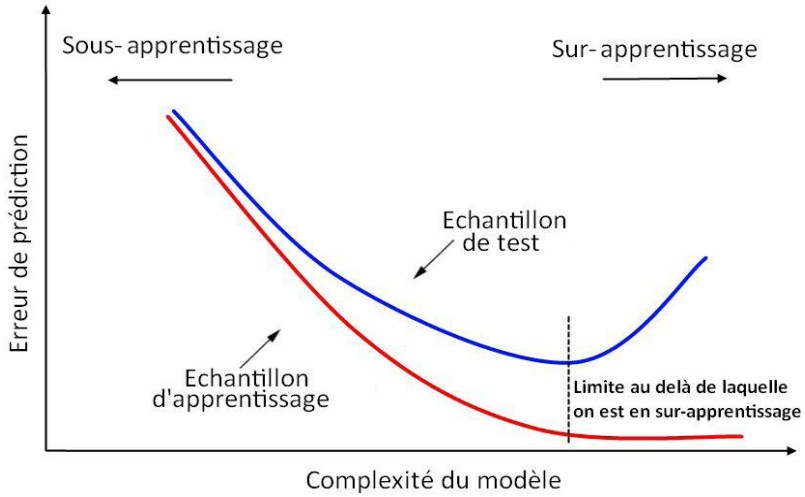


Figure 5.1. Erreur de prédiction sur les ensembles d'apprentissage et de test, en fonction de la complexité du modèle [Gar 11].

5.3. Sélection d'Attributs

Dans certains domaines (telle en bioinformatique, et plus particulièrement dans l'étude de puces à ADN), les problèmes posés ont une originalité qui vient du fait que nous avons un très grand nombre de variables (ex., de gènes) pour un très petit nombre de cas d'études.

5.3.1. Principe

Le but de la sélection d'attributs est ***de choisir***, parmi toutes les variables du problème, ***un sous-ensemble*** de taille réduite *contenant* ***les variables les plus pertinentes*** concernant le problème.

Les raisons de son utilisation sont multiples :

- La construction d'un modèle de prédiction peut s'avérer très lourde, voire impossible, si le **nombre de variable est trop élevé** (*malédiction* de la dimensionnalité ou *curse of dimensionality*) ;
- Améliorer la qualité de la solution en supprimant les variables qui sont sources de bruits [Xu 09] ;
- Éviter le sur-apprentissage, en améliorant les capacités de généralisation ;

5.3.1.1. Définition

On peut d'ailleurs distinguer trois types de variables selon leur congruence ou adéquation au problème de classification :

- ✓ Les variables **fortement pertinentes** : indispensables pour représenter le problème,
- ✓ Les variables **faiblement pertinentes** : utiles pour la compréhension de certains exemples, et

5.3.1.2. Calcul de la Corrélation

Pour calculer la corrélation de deux vecteurs $x = \{x_1, \dots, x_n\}$ et $y = \{y_1, \dots, y_n\}$, on peut utiliser le **coefficient de corrélation** de **Bravais-Pearson** (*cf.* équation (5.1)).

$$r_p = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sigma_x \sigma_y} \qquad (5.1)$$

En désignant par $\bar{x}$, $\bar{y}$ les **moyennes** et σ_x, σ_y les **écart-types** des vecteurs x et y.

Remarques

- ✓ À part ces cas d'exception, les méthodes de sélection d'attributs peuvent être séparées en deux grandes catégories : les ***méthodes par filtre*** et les ***méthodes wrapper*** [Inz 04].

5.3.2. Méthodes par Filtre

Ces méthodes attribuent une ***valeur*** (ou *score*) à chaque variable en fonction d'un critère statistique.

Les variables peuvent ainsi être classées afin de sélectionner celles qui ont le meilleur score.

Formalisation du problème : On pose $x = x_1, \dots, x_n$ un attribut dont les valeurs varient parmi n exemples. On note $\bar{x}$ la moyenne et σ l'écart-type des valeurs prises par x sur l'ensemble des n exemples.

Nous présentons quelques critères classiques de la littérature. Afin de simplifier les équations, nous nous plaçons dans le cadre des ***problèmes à deux classes*** ($\boldsymbol{p = 2}$).

5.3.2.1. t-test de *Welsh*

Le ***t-test*** est probablement le test le plus connu et le plus utilisé. Il en existe différentes variantes qui dépendent de la ***répartition des classes*** et de ***la variance de chaque classe***.

Le *t-test de Welsh* est le plus général, mais il pose tout de même *l'hypothèse d'une* ***distribution normale des exemples***.

$$t(x) = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{{\sigma_1}^2}{n_1} + \frac{{\sigma_2}^2}{n_2}}} \qquad (5.2)$$

5.3.2.2. F-Test (ou test de Fisher)

Ce test repose sur la ***distribution de Fisher***, qui consiste à ***comparer la différence entre deux variances*** [Dun 55].

On peut remarquer que, dans le cas de deux classes, $F(x) = t(x)^2$

$$F(x) = \frac{(\bar{x}_1 - \bar{x}_2)^2}{\frac{{\sigma_1}^2}{n_1} + \frac{{\sigma_2}^2}{n_2}} \qquad (5.3)$$

5.3.2.3. Rapport Signal sur Bruit

Ce critère (aussi appelé ***P-measure***) provient des équations utilisées dans le traitement du signal.

Ici, les attributs ayant de grandes valeurs de $|P(x)|$ seront **fortement corrélés** à la séparation des classes.

5.3.2.4. Test du χ^2

Le test du χ^2 est à l'origine utilisé pour ***évaluer l'indépendance de variables discrètes***.

Cependant, il est possible d'appliquer ce test sur des variables à valeurs continues, en discrétisant les données [Liu 95].

5.3.3. Méthodes *Wrapper*

D'autres méthodes de sélection d'attributs se basent sur ***l'optimisation d'une fonction objective*** afin de trouver un sous-ensemble optimal [Blu 97].

L'espace de recherche est assimilé à l'ensemble des sous-ensembles d'attributs possibles.

Les deux méthodes d'optimisation les plus utilisées dans ce domaine sont des ***méthodes de recherche locale*** *de type* ***glouton***, qui permettent de pallier au problème lié à la grande dimension :

1. La première est appelée ***Forward Selection*** (*Sélection ascendante, cf.* figure 5.2). Elle *démarre d'une solution ne contenant aucun attribut* et parcourt l'ensemble des attributs en les ajoutant à la solution initiale. Lorsqu'elle les a tous parcourus, elle garde le meilleur, l'ajoute à la solution optimale, et recommence jusqu'à ce qu'un critère d'arrêt soit atteint.

2. La deuxième méthode, appelée ***Backward Elimination*** (*Élimination descendante, cf.* figure 5.3) fonctionne de la même façon, mais *en partant d'une solution contenant tous les attributs* et en les retirant tour à tour, pour finalement retirer celui qui obtient les moins bons résultats.

Fonction ForwardSelection ;

Début

Soit $P = \emptyset$: l'ensemble solution des attributs sélectionnés ;

Soit Q : l'ensemble des attributs possibles ;

Tant que < la condition d'arrêt n'est pas vérifiée > **Faire**

Pour chaque attribut $a \in Q$ **Faire**

$P' = P \cup \{a\}$;

Créer le modèle de prédiction avec les variables de P' et estimer ses performances ;

Fin pour

$P = P \cup \{a^*\}$ où a^* est l'attribut ayant obtenu les meilleures performances ;

$Q = Q - \{a\}$;

Fin tant que

Retourner $\boldsymbol{P}$ **;**

Fin.

Figure 5.2. Pseudo code de la méthode d'optimisation "***Forward Selection***"

Fonction BackwardElimination ;

Début

Soit P : l'ensemble des attributs possibles ;

Tant que < la condition d'arrêt n'est pas vérifiée > **Faire**

Pour chaque attribut $a \in P$ **Faire**

$P' = P - \{a\}$;

Créer le modèle de prédiction avec les variables de P' et estimer ses performances ;

Fin pour

$P = P - \{a^*\}$ où a^* est l'attribut ayant obtenu les moins bonnes performances ;

Fin tant que

Retourner $\boldsymbol{P}$ **;**

Fin.

Figure 5.3. Pseudo code de la méthode d'optimisation **"Backward Elimination"**

5.4. Critères de Performance

L'évaluation de la performance d'un classifieur se fait par *le* ***taux de bonnes classifications***.

Exemple, si pour 100 exemples de tests, 89 ont été prédits correctement par notre modèle de prédiction, on pourra dire que ce modèle a une précision de 89% (souvent écrit 0,89).

Mais la ***précision*** n'est pas le seul critère à prendre en compte, notamment pour les problèmes à deux classes.

En effet, lorsque l'on travaille avec des **modèles de prédiction binaires**, de nouveaux critères peuvent entrer en jeu (par exemple, dans le domaine médical). On *ne s'intéresse pas seulement au nombre de prédiction correctes*, mais on veut également savoir si une ***prédiction a été faite positive*** alors que *l'exemple montrait réellement un résultat négatif* (on parle de *faux positif*) ou l'inverse (on parle alors de *faux négatif*), comme le montre le tableau 5.1, dit **tableau de confusion**.

	C	$\bar{C}$
Prédit C	Vrai Positif (VP) ou True Positive (TP)	Faux Positif (FP) ou False Positive (FP)
Prédit $\bar{C}$	Faux Négatif (FN) ou False Negative (FN)	Vrai Négatif (VN) ou True Negative (TN)

Tableau 5.1. Résultats détaillés d'une prédiction binaire (table de confusion).

On peut alors définir les termes suivants :

- **Le taux de vrais positifs** ("True Positive rate") ou **Sensibilité** (sensitivity), ou encore **Rappel** (recall) : C'est la probabilité que la prédiction soit C lorsque l'exemple est de classe C. On la calcule par l'équation :

$$S_e = tpr = \frac{TP}{TP + FN}$$

- **Le taux de vrais négatifs** ("True Negative rate") ou **Valeur Prédictive Négative :** C'est la probabilité que l'exemple soit de classe $\bar{C}$ lorsque la prédiction est $\bar{C}$. On la calcule par l'équation :

$$tnr = vpn = \frac{TN}{TN + FN}$$

- **Le taux de faux positifs** ("False Positive rate") :

$$fpr = \frac{FP}{FP + TN}$$

- **Le taux de faux négatifs** ("False Negative rate") :

$$fnr = \frac{FN}{FN + TN}$$

- **Le taux de bonne classification** ou **l'exactitude** (accuracy) :

$$acc = tbc = \frac{TP + TN}{TP + FN + TN + FP} \approx (TP + FN) * tpr + (FP + TN) * (1 - fpr)$$

Remarque :

- Ces valeurs sont primordiales lors d'une *classification binaire* et sont malheureusement souvent délaissées au profit de la *précision*.
- Elles permettent de déterminer si le modèle est performant à prédire une caractéristique spécifique dans une population donnée.

5.4.1. Compromis Sensibilité/Spécificité

- La **sensibilité** et la **spécificité** d'une classification *doivent toujours être données ensemble*.
- On peut d'ailleurs définir l'***indice de Youden*** : $I_{Youden} = S_e + S_p - 1$. Cet indice révèle ***l'efficacité de la prédiction*** : s'il est *négatif* ou *nul*, la *classification est* ***inefficace***. Elle est d'autant plus efficace qu'il se rapproche de 1.
- Dans notre exemple, le $\boldsymbol{I_{Youden}}$ vaudrait **0**, révélant que *la prédiction est* ***inefficace***.

5.4.2. Représentation Graphique des Performances

Principalement, deux courbes sont les plus utilisées dans la littérature de recherches pour comparer les performances des classifieurs : la **Courbe Rappel-Précision** (Precision-Recall Curve –PRC, en anglais) et la **courbe ROC** (Receiver Operator Characteristic –ROC curve, en anglais).

- La courbe Rappel-Précision (PRC –Precision Recall Curve) utilise comme abscisse les valeurs de *Rappel* (Recall) obtenues par l'algorithme de classification, alors que l'axe des ordonnées est réservé aux valeurs de performance obtenues en *Précision*. Les deux axes sont gradués par une échelle de 1 : 0 – 0,1 – 0,2 – 0,3 – 0,4 – 0,5 – 0,6 – 0,7 – 0,8 – 0,9 – 1 (les valeurs sont des taux compris entre 0 à 1).

- La courbe ROC, quant à elle, utilise le *Taux de Faux Positifs* (False Positive Rate) comme valeurs de l'*axe des abscisses* et le *Taux de Vrais Positifs* (True Positive Rate) en *ordonnée*. Les deux axes sont gradués sur une échelle de 1 : 0 – 0,1 – 0,2 – 0,3 – 0,4 – 0,5 – 0,6 – 0,7 – 0,8 – 0,9 – 1 (les valeurs sont des taux compris entre 0 à 1).

5.5. Méthodes de Validation

Pour avoir une estimation correcte de l'erreur de classification, il faut recourir à un ensemble d'exemples qui n'ont pas servi pour l'apprentissage : il s'agit de *l'ensemble de test* [Gar 11].

Les *techniques de validation croisée* (ou *cross validation*) [Sto 77] [Koh 95] permettent d'obtenir une *estimation des performances du classifieur*, *en exploitant la* ***totalité du jeu de données***. Ceci est obtenu en faisant plusieurs tests sur différents ensembles d'apprentissage et de test, et en faisant la moyenne des résultats. En effet, si l'on obtient une *bonne précision en moyenne*, ainsi qu'*un écart-type faible*, notre méthode de prédiction pourra être considérée comme **robuste**.

5.5.1. Validation Croisée : $k - Fold$

L'algorithme de validation croisée $\boldsymbol{k - Fold}$ consiste à segmenter aléatoirement les exemples du jeu de données initial en $\boldsymbol{k}$ *sous-ensembles disjoints numérotés de* $\boldsymbol{1}$ *à* $\boldsymbol{k}$ [Gar11].

Ensuite, on va écarter le $\mathbf{1^{er}}$ **bloc** qui nous *servira d'ensemble de test*, et utiliser les $\boldsymbol{k - 1}$ autres blocs afin de *constituer l'ensemble d'apprentissage*.

De manière générale, on choisit $\boldsymbol{k = 10}$ si le jeu de données est suffisamment grand.

5.5.2. Validation Croisée : $Leave\ One\ Out$

Cette méthode est dérivée de la méthode de validation $\boldsymbol{k - Fold}$, en prenant $\boldsymbol{k = n}$, $\boldsymbol{n}$ étant le nombre d'exemples.

A chaque itération, on va donc *faire l'****apprentissage sur tous*** *les exemples* ***moins un***, et *tester sur un seul exemple*, afin de vérifier s'il est prédit correctement.

5.5.3. Validation Croisée : $Repeated\ Random\ SubSampling$

Cette méthode est également dérivée de la méthode de validation $\boldsymbol{k - Fold}$ et y est même assimilée par erreur dans de nombreuses publications.

Une fois la prédiction faite et les performances du classifieur estimées, l'ensemble initial est reconstitué. On *recommence ensuite cette procédure* ***autant de fois que l'on veut*** et on calcule la moyenne et l'écart-type des performances sur l'ensemble des tests.

5.5.4. $Bootstrapping$

Le *bootstrapping* [Efr 83] est une technique très souvent utilisée dans le cadre de jeux de données contenant peu d'exemples.

Lorsque l'on utilise le principe de validation croisée, on sélectionne aléatoirement les exemples dans le jeu de données initial afin de constituer les ensembles d'apprentissage et de test, en prenant garde à ne pas prendre deux fois le même exemple.

Bibliographie

Livres de Base

[Bel 92] Abdel Belaïd et Yolande Belaïd. *Reconnaissance des Formes : Méthodes et Applications*. ISBN-13: 978-2729603991, InterÉditions, 1992.

[Cor 03] Antoine Cornuéjols et Laurent Miclet. *Apprentissage Artificiel : Concepts et Algorithmes*, éditions Eyrolles, 2003.

[Dev 82] Pierre Devijver, Josef Kittler, "Pattern Recognition: A Statistical Approach". Prentice-Hall, 1982.

[Dub 90] Bernard Dubuisson, "Diagnostic et reconnaissance des formes". Hermès Science Pub., 1990.

[Dud 73] Richard DUDA and Peter HART, "Pattern Classification and Scene Analysis". John Wiley & Sons, 1973.

[Fu 74] King-Sun FU, "Syntactic Methods in Pattern Recognition". Academic Press, 1974.

[Gai 83] Gérard GAILLAT, "Méthodes statistiques de reconnaissance des formes". Publication ENSTA, 1983.

[Mic 84] Laurent MICLET, "Méthodes structurelles pour la reconnaissance des formes". Eyrolles et CNET - ENST, 1984.

[Mil 93] Maurice MILGRAM, "Reconnaissance des formes : Méthodes numériques et connexionnistes". Armand Colin, 1993.

[Pav 82] Theo PAVLIDIS, "Structural Pattern Recognition". Springer Verlag, 1982.

[Sim 84] Jean-Claude SIMON, "La reconnaissance des formes par algorithmes". Masson, 1984.

[Wat 69] Satosi WATANABE, "Knowing and Guessing". John Wiley, 1969.

Autres Références

[Bar 89] H. B. Barlow. *Unsupervised Learning.* Neural Computation, vol. 1, no. 3, pages 295-311, 1989.

[Bis 95] C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[Blu 97] A. L. Blum and P. Langley. *Selection of Relevant Features and Examples in Machine Learning*. Artificial Intelligence, vol. 97, no. 1-2, pages 245-271, 1997.

[Bor 05] Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. *Fast kernel classifiers with online and active learning*. Journal of Machine Learning Research, 6:1579–1619, September 2005.

[Bos 92] B. E. Boser, I. Guyon, and V. Vapnik. *A Training Algorithm for Optimal Margin Classifiers*. Proc. of the 5th ACM Conf. on Computational Learning Theory (COLT'92), pp. 144-152, Pittsburgh, PA, USA, July, 1992.

[Bou 85] J. M. Bouroche et G. Saporta. *L'analyse des données*. Rapport interne, 1985.

[Bro 00] M. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. W. Sugnet, T. S. Furey, M. Ares, and D. Haussler. *Knowledge-based Analysis of Microarray Gene Expression Data By Using Support Vector Machines*. In Proc. of National Academy of Sciences (PNAS), vol. 97, no. 1, pp. 262-267, 2000.

[Can 07] S., CANU. *Machines à noyaux pour l'apprentissage statistique*. Techniques de l'ingénieur - Dossier: TE5255. 2007.

[Cha 06] O. Chapelle, B. Schölkopf and A. Zien. *Semi-supervised learning*. Adaptive computation and machine learning. MIT Press, 2006.

[Che 00] L. Chen. A new LDA-based face recognition system which can solve the small sample size problem. Pattern Recognition, vol. 33, no. 10, pages 1713-1726, 2000.

[Coc 93] A. Cochocki and R. Unbehauen. *Neural networks for optimization and signal processing*. John Wiley & Sons, 1993.

[Cor 95] C. Cortes and V. Vapnik. *Support-vector networks*. Machine Learning, 20(3) : 273–297. 1995.

[Dun 55] D. B. Duncan. *Multiple range and multiple F tests*. Biometrics, vol. 11, pages 1-42, 1955.

[Efr 79] B. Efron. *Bootstrap Methods: Another Look at the Jackknife*. The Annals of Statistics, vol. 7, no. 1, pages 1-26, 1979.

[Efr 83] B. Efron and G. Gong. *A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation*. The American Statistician, vol. 37, no. 1, pages 36-48, 1983.

[Fuk 90] K. Fukunaga. *Introduction to statistical pattern recognition*. Academic Press, 1990.

[Gar 11] V. Gardeux, *Conception d'heuristiques d'optimisation pour les problèmes de grande dimension : Application à l'analyse de données de puces à ADN*. Thèse de doctorat, 2011, MSTIC, Université de Paris-Est Créteil.

[Gol 99] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, and C. D. Bloomfield. *Molecular classification of cancer: class discovery and class prediction by gene expression monitoring*. Science, vol. 286, no. 5439, pages 531-537, 1999.

[Inz 04] I. Inza, P. Larranaga, R. Blanco and A. Cerrolaza. *Filter versus wrapper gene selection approaches in DNA microarray domains*. Artificial Intelligence in Medicine, vol. 31, no. 2, pp. 91-103, 2004.

[Jai 97] A. Jain and D. Zongker. *Feature selection: Evaluation, application, and small sample performance*. IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 19, no. 2, pages 153-158, 1997.

[Kae 96] L. P. Kaelbling, M. Littman and A. Moore. *Reinforcement Learning : A Survey*. Journal of Artificial Intelligence Research, vol. 4, issue 1, pages 237-285, 1996.

[Koh 95] R. Kohavi. *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*. In Proc. of the 14th Inter. Joint Conf. on Artificial Intelligence (IJCAI'95), pages 1137-1143, Montreal, Quebec, Canada, August 20-25, 1995.

[Koh 97] R. Kohavi and G. H. John. *Wrappers for feature subset selection*. Artificial Intelligence, vol. 97, no. 1, pages 273-324, 1997.

[Let 59] J. Lettvin, H. Maturana, W. McCulloch, and W. Pitts. What the Frog's Eye Tells the Frog's Brain. In Proc. of the Institute of Radio Engineers, vol. 47, issue 11, pages 1940-1951, Nov. 1959.

[Liu 95] H. Liu and R. Setiono. *Chi2 : Feature Selection and Discretization of Numeric Attributes*. In Proceedings of the Seventh International Conference on Tools with Artificial Intelligence, pages 388-391, Herndon, VA, USA, November 5-8, 1995.

[Mur 94] P. Murphy and M. J. Pazzani. *Exploring the decision forest*. In Computational Learning and Natural Language Workshop, Provincetown, pages 257-275. 1994.

[Pla 99] John Platt. *Fast training of support vector machines using sequential minimal optimization*. In B. Scholkopf, C. Burges, and A. Smola, editors, Advanced in Kernel Methods – Support Vector Learning, pages 185–208. MIT Press, 1999.

[Qui 86] J.R. Quinlan. Induction of decision trees. Machine learning, vol.1, no. 1, pp. 81-106, 1986.

[Qui 93] J. R. Quinlan. *C4.5 : programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 1993.

[Ros 58] F. Rosenblatt. *The perceptron* : a probabilistic model for information storage and organization in the brain. Psychological Review, vol. 65, no. 6, pages 386-408, 1958.

[Rou 69] M. Roux. *Un algorithme pour construire une hiérarchie particulière*. Thèse 3ème cycle, ISUP, Université Paris VI, 1969.

[Rum 86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1: Foundations, chapter 8 : Learning internal representations by error propagation, pages 318-362. MIT Press, 1986.

[Tho 05] C. E. Thomaz and D. F. Gillies. A Maximum Uncertainty LDA-Based Approach for Limited Sample Size Problems - With Application to Face Recognition. In Proceedings IEEE of the 18th Symposium on Computer Graphics and Image Processing (SIBGRAPI 2005), pp 89-96, 2005.

[Vap 95] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag, New York, USA, 1995.

[Vis 03] S. V. N Vishwanathan, Alexander J. Smola, and M. Narasimha Murty. *SimpleSVM*. Proceedings of the Twentieth International Conference on Machine Learning, 2003.

[Wan 07] X. Wang, J. Yang, X. Teng, W. Xia and R. Jensen. *Feature selection based on rough sets and particle swarm optimization*. Pattern Recognition Letter, vol. 28, no. 4, pages 459-471, 2007.

[Web 96] Webb, G. I. (1996). *Further experimental evidence against the utility of occam's razor*. Journal of Artificial Intelligence Research, 4 : pages 397-417.

[Wol 95] Wolpert, D. H. and Macready, W. G. (1995). *No free lunch theorems for search*. Technical Report SFI-TR-95-02-010, The Santa Fe Institute.

[Xin 01] E. P. Xing, M. I. Jordan and R. M. Karp. *Feature Selection for High-Dimensional Genomic Microarray Data*. In Proceedings of 18th International Conference on Machine Learning (ICML'01), pages 601-608, Williams College, Williamstown, USA, June 28–July 1, 2001.

[Xu 09] P. Xu, G. N. Brock, and R. S. Parrish. *Modified linear discriminant analysis approaches for classification of high-dimensional microarray data*. Computational Statistics and Data Analysis, vol. 53, no. 5, pages 1674-1687, 2009.

[Yan 97] Y. Yang and J. O. Pedersen. *A Comparative Study on Feature Selection in Text Categorization*. In Proc. of 14th Int'l Conf. on Machine Learning (ICML), pp. 412-420, Nashville, USA, July 8-12, 1997.

[Yan 98] J. Yang and V. G. Honavar. *Feature Subset Selection Using a Genetic Algorithm*. IEEE Intelligent Systems, vol. 13, no. 2, pages 44-49, 1998.

[Ye 05] J. Ye and B. Yu. Characterization of a family of algorithms for generalized discriminant analysis on undersampled problems. Journal of Machine Learning Research, vol. 6, pages 483-502, 2005.