# Chapitre 2

# Machines à vecteur support

## 2.1 Introduction

Les machines à vecteur support se situent sur l'axe de développement de la recherche humaine des techniques d'apprentissage. Les SVMs sont une classe de techniques d'apprentissage introduite par Vladimir Vapnik au début des années 90 [26, 60], elles reposent sur une théorie mathématique solide à l'inverse des méthodes de réseaux de neurones. Elles ont été développées au sens inverse du développement des réseaux de neurones : ces derniers ont suivi un chemin heuristique de l'application et l'expérimentation vers la théorie ; alors que les SVMs sont venues de la théorie du son vers l'application. Les SVMs sont dans leur origine utilisées pour la classification binaire et la régression. Aujourd'hui, elles sont utilisées dans différents domaines de recherche et d'ingénierie tel que le diagnostic médical, le marketing, la biologie, la reconnaissance de caractères manuscrits et de visages humains. Le présent chapitre introduit les machines à vecteur support, leurs origines théoriques, leurs différentes formes et leur optimisation. Le chapitre est organisé comme suit : il commence par introduire la relation entre l'apprentissage et les SVMs. Nous verrons ensuite la forme originale des SVMs : le cas binaire, puis le cas multiclasse. Les sections qui suivent sont consacrées aux variantes les plus connues des SVMs tel que la régression et les SVMs monoclasse. Les algorithmes d'implémentation sont ensuite présentés et leurs méthodes d'évaluation sont présentées dans la dernière section du chapitre.

## 2.2 Apprentissage statistique

La théorie d'apprentissage statistique étudie les propriétés mathématiques des machines d'apprentissage [149]. Ces propriétés représentent les propriétés de la classe de fonctions ou modèles que peut implémenter la machine. L'apprentissage statistique utilise un nombre limité d'entrées (appelées exemples) d'un système avec les valeurs de leurs sorties pour apprendre une fonction qui décrit la relation fonctionnelle existante, mais non connue, entre les entrées et les sorties du système.

On suppose premièrement que les exemples d'apprentissage, appelés aussi exemples d'entrainement, sont générés selon une certaine probabilité inconnue (mais fixe) c'est-à-dire indépendants et identiquement distribués (iid). C'est une supposition standard dans la théorie d'apprentissage. Les exemples sont de dimension $m$ ($\in \Re^m$) et dans le cas d'apprentissage supervisé, accompagnés d'étiquettes caractérisant leurs types ou classes d'appartenance. Si ces étiquettes sont dénombrables, on parle de classification sinon on parle de régression. Dans le cas d'une classification binaire cette étiquette est soit +1 ou -1. L'ensemble des exemples et leurs étiquettes correspondantes est appelés ensemble d'apprentissage.

Une machine efficace d'apprentissage est une machine qui apprend de l'ensemble d'entrainement une fonction qui minimise les erreurs de classification sur l'ensemble lui même.

Soit $D = \{(x_1, y_1), .., (x_n, y_n)\}$ l'ensemble des exemples d'entrainement, avec $x_i \in \Re^m$ et $y_i = \pm 1$, et soit $f(x)$ la fonction apprise par la machine d'apprentissage. On appelle le risque empirique $R_{emp}[f]$ le taux d'erreurs effectuées par la fonction $f$ sur l'ensemble d'entrainement $D$.

$$\begin{aligned} &R_{emp} = \tfrac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) \\ &\text{avec,} \\ &L = \begin{cases} 1 & \text{si} \quad y_i = f(x_i) \\ 0 & \text{sinon} \end{cases} \end{aligned} \tag{2.1}$$

Puisque l'ensemble D ne représente qu'une simple partie de tout l'espace d'exemples, la fonction apprise $f$, qui minimise le risque empirique, peut se comporter mal avec les nouveaux exemples non vus à l'entrainement. C'est un phénomène très connu en apprentissage automatique appelé le sur-apprentissage ou apprentissage par cœur. Pour garantir que $f$, prenne en charge même les exemples non jamais vus, il faut contrôler sa *capacité de généralisation* mesurée souvent sur un autre ensemble d'exemples appelé ensemble de test réservé uniquement pour tester la machine apprise. La fonction $f$ recherchée doit donc minimiser les erreurs de classification sur les deux ensembles : d'entrainement et de test.

Trouver la fonction optimale $f$ revient toujours à un problème d'optimisation, ce qui explique la forte relation entre l'apprentissage et l'optimisation. Avant de rechercher la fonction $f$, il faut définir son type puis rechercher ses paramètres.

Dans le cas des machines à vecteur support, la fonction recherchée est de forme linéaire. Les SVMs sont, donc, des systèmes d'apprentissage qui utilisent un espace d'hypothèses de fonctions linéaires dans un espace de caractéristique à haute dimension. Cette stratégie d'apprentissage introduite par Vapnik et ses co-auteurs [150] est une méthode très puissante. Elle a pu, en quelques années depuis sa proposition, conquérir la plupart des autres systèmes d'apprentissage dans une grande variété de domaines d'application.

## 2.3 SVMs binaires

Le cas le plus simple est celui où les données d'entrainement viennent uniquement de deux classes différentes (+1 ou -1), on parle alors de classification binaire. L'idée des SVMs est de

rechercher un hyperplan (droite dans le cas de deux dimensions) qui sépare le mieux ces deux classes (figure 2.1).

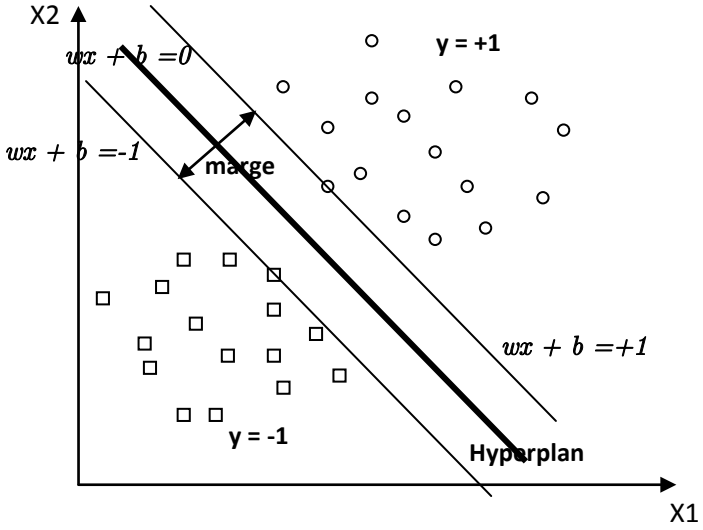


FIGURE 2.1 – SVM binaire

Si un tel hyperplan existe, c'est-à-dire si les données sont linéairement séparables, on parle d'une machine à vecteur support à marge dure (Hard margin).

### 2.3.1 SVM à marge dure

L'hyperplan séparateur est représenté par l'équation (2.2) suivante :

$$H(x) = w^T x + b \tag{2.2}$$

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

$$\begin{cases} \text{Classe} = 1 & \text{Si} \quad H(x) > 0 \\ \text{Classe} = -1 & \text{Si} \quad H(x) < 0 \end{cases} \tag{2.3}$$

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

$$\begin{cases} \text{Classe} = 1 & \text{Si} \quad H(x) > 1 \\ \text{Classe} = -1 & \text{Si} \quad H(x) < -1 \end{cases} \tag{2.4}$$

Les valeurs +1 et -1 à droite des inégalités peuvent être des constantes quelconques +a et -a, mais en divisant les deux parties des inégalités par a, on trouve les inégalités précédentes qui sont équivalentes à l'équation (2.5) :

$$y_i(w^T x_i + b) \geq 1, i = 1..n \tag{2.5}$$

L'hyperplan $w^T x + b = 0$ représente un hyperplan séparateur des deux classes, et la distance entre cet hyperplan et l'exemple le plus proche s'appelle la marge (figure 2.1). La région qui se trouve entre les deux hyperplans $w^T x + b = -1$ et $w^T x + b = +1$ est appelée la région de généralisation de la machine d'apprentissage. Plus cette région est importante, plus est la

capacité de généralisation de la machine. La maximisation de cette région est l'objectif de la phase d'entrainement qui consiste, pour la méthode SVM, à rechercher l'hyperplan qui maximise la région de généralisation c-à-d la marge. Un tel hyperplan est appelé *"hyperplan de séparation optimale"*. En supposant que les données d'apprentissage ne contiennent pas des données bruitées (mal-étiquetées) et que les données de test suivent la même probabilité que celle des données d'entraînement, l'hyperplan de marge maximale va certainement maximiser la capacité de généralisation de la machine d'apprentissage.

La détermination de l'hyperplan optimal passe par la détermination de la distance euclidienne minimale entre l'hyperplan et l'exemple le plus proche des deux classes. Puisque le vecteur $w$ est orthogonal sur l'hyperplan séparateur, la droite parallèle à $w$ et reliant un exemple $x$ à l'hyperplan est donnée par la formule (2.6) :

$$\frac{aw}{\|w\|} + x = 0 \tag{2.6}$$

Où $a$ représente la distance entre $x$ et l'hyperplan. La résolution de (2.6), donne (2.7) :

$$a = -\frac{w^T x + b}{\|w\|} \tag{2.7}$$

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

$$\frac{y_i(w^T x_i + b)}{\|w\|} \geq \delta \tag{2.8}$$

Si une paire $(w, b)$ est une solution alors $(aw, ab)$ est une solution aussi où $a$ est un scalaire. On impose alors la contrainte (2.9) suivante :

$$\|w\| \, \delta \geq 1 \tag{2.9}$$

Pour trouver l'hyperplan séparateur qui maximise la marge, on doit déterminer, à partir des deux dernières inégalités, le vecteur $w$ qui possède la norme euclidienne minimale et qui vérifie la contrainte de l'équation (2.5), de bonne classification des exemples d'entrainement. L'hyperplan séparateur optimal peut être obtenu en résolvant le problème de l'équation (2.10) :

$$\begin{cases} \text{Minimiser} & \frac{1}{2}\|w\|^2 \\ \text{sous contraintes} & \\ y_i(w^T x_i + b) \geq 1 & \forall i = 1..n \end{cases} \tag{2.10}$$

Remarquons que nous pouvons obtenir le même hyperplan même en supprimant toutes les données qui vérifient l'inégalité de la contrainte. Les données qui vérifient l'égalité de la contrainte s'appellent les vecteurs supports, et ce sont ces données seules qui contribuent à la détermination de l'hyperplan. Dans la figure 2.1, les données qui se trouvent sur les deux droites $+1$ et $-1$ représentent les vecteurs supports.

Le problème de l'équation (2.10) est un problème de programmation quadratique avec contraintes linéaires. Dans ce problème, les variables sont $w$ et $b$, c-à-d que le nombre de variables est égal à $m + 1$. Généralement, le nombre de variables est important ce qui ne permet pas d'utiliser les techniques classiques de programmation quadratique. Dans ce cas le problème (2.10) est convertit en un problème dual équivalent sans contraintes de l'équation (2.11) qui introduit les multiplicateurs de Lagrange :

$$Q(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{n} \alpha_i \left\{ y_i(w^T x_i + b) - 1 \right\} \tag{2.11}$$

Où les $\alpha_i$ sont les multiplicateurs non négatifs de Lagrange. L'optimum de la fonction objective $Q$ peut être obtenu en la minimisant par rapport à $w$ et $b$ et en la maximisant par rapport

aux $\alpha_i$. À l'optimum de la fonction objective, ses dérivées par rapports aux variables $w$ et $b$ s'annulent ainsi que le produit des $\alpha_i$ aux contraintes (équation 2.12) :

$$\begin{cases} \frac{\partial Q(w,b,\alpha)}{\partial w} = 0 & (a) \\ \frac{\partial Q(w,b,\alpha)}{\partial b} = 0 & (b) \\ \alpha_i \left\{ y_i(w^T x_i + b) - 1 \right\} = 0 & (c) \\ \alpha_i \geq 0 & (d) \end{cases} \tag{2.12}$$

De (2.12.a) on déduit :

$$\begin{cases} w = \sum_{i=1}^{n} \alpha_i y_i x_i \\ \sum_{i=1}^{n} \alpha_i y_i = 0 \end{cases} \tag{2.13}$$

En remplaçant dans 2.11, on obtient le problème dual à maximiser suivant :

$$\begin{cases} \text{Maximiser} & Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{Sous contraintes} & \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \end{cases} \tag{2.14}$$

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

$$H(x) = \sum_{S} \alpha_i y_i x^T x_i + b \tag{2.15}$$

Où $S$ représente l'ensemble des vecteurs supports. $b$ peut être calculé à partir de n'importe quel vecteur support par l'équation (2.16) :

$$b = y_i - w^T x_i \tag{2.16}$$

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

$$b = \frac{1}{|S|} \sum_{i \in S} y_i - w^T x_i \tag{2.17}$$

La fonction de décision H peut être calculée, donc, pour chaque nouvel exemple $x$ par l'équation (2.15) et la décision peut être prise comme suit :

$$\begin{cases} x \in Classe + 1 & si \quad H(x) > 0 \\ x \in Classe - 1 & si \quad H(x) < 0 \\ x \quad \text{inclassifiable} & si \quad H(x) = 0 \end{cases} \tag{2.18}$$

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

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

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

### 2.3.2 SVM à marge souple

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

Dans le cas où les données ne sont pas linéairement séparables, ou contiennent du bruit (outliers : données mal étiquetées) les contraintes de l'équation (2.5) ne peuvent être vérifiées, et il y a nécessité de les relaxer un peu. Ceci peut être fait en admettant une certaine erreur de classification des données (figure 2.2) ce qui est appelé "SVM à marge souple (Soft Margin)".

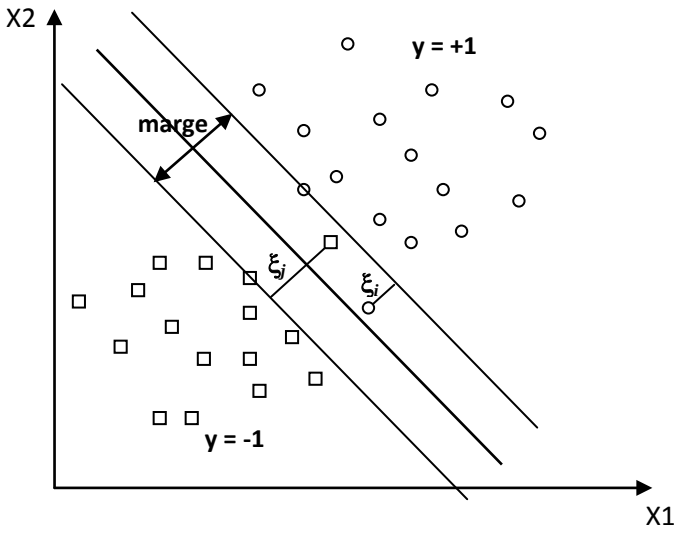


FIGURE 2.2 – SVM binaire à marge souple

On introduit alors sur les contraintes des variables $\xi_i$ dites de relaxation pour obtenir la contrainte de l'équation (2.19) :

$$y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1..n \tag{2.19}$$

Grâce aux variables de relaxation non négatives $\xi_i$, un hyperplan séparateur existera toujours.

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

En combinant avec l'équation (2.10), on obtient le problème primal de l'équation (2.20) suivant :

$$\begin{cases} \text{Minimiser} & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{sous contraintes} & \\ y_i(w^T x_i + b) \geq 1 - \xi_i & i = 1..n \\ \xi_i \geq 0 & \end{cases} \tag{2.20}$$

Où C est un paramètre positif libre (mais fixe) qui représente une balance entre les deux termes de la fonction objective (la marge et les erreurs permises) c-à-d entre la maximisation de la marge et la minimisation de l'erreur de classification. On obtient le problème dual de l'équation (2.21) où on introduit les multiplicateurs de Lagrange $\alpha_i$ et $\beta_i$ :

$$Q(w,b,\alpha,\xi,\beta) = \frac{1}{2}w^T w + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i y_i(w^T x_i + b) - 1 + \xi_i - \sum_{i=1}^{n}\beta_i\xi_i \tag{2.21}$$

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

$$\begin{cases} \frac{\partial Q(w,b,\xi,\alpha,\beta)}{\partial w} = 0 & (a) \\ \frac{\partial Q(w,b,\xi,\alpha,\beta)}{\partial b} = 0 & (b) \\ \alpha_i \left\{ y_i(w^T x_i + b) - 1 + \xi_i \right\} = 0 & (c) \\ \beta_i \xi_i = 0 & (d) \\ \alpha_i \geq 0\,;\, \beta_i \geq 0\,;\, \xi_i \geq 0 & (e) \end{cases} \tag{2.22}$$

On déduit :

$$\begin{cases} w = \sum_{i=1}^{n} \alpha_i y_i x_i \\ \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \alpha_i + \beta_i = 0 \end{cases} \tag{2.23}$$

En remplaçant l'équation (2.23) dans l'équation (2.21), on obtient le problème dual (2.24) suivant :

$$\begin{cases} \text{Maximiser} & Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{sous contraintes} & \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \end{cases} \tag{2.24}$$

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

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

Ces conditions sur les $\alpha_i$ sont appelées les conditions de Karush-Kuhn-Tucker (KKT) [77, 85], elles sont très utilisées par les algorithmes d'optimisation pour rechercher les $\alpha_i$ optimums et par conséquent l'hyperplan optimal.

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

$$H(x) = \sum_{i \in U} \alpha_i y_i x_i^T x + b \tag{2.25}$$

Pour les vecteurs supports non bornés, nous avons :

$$b = y_i - w^T x_i \tag{2.26}$$

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

$$b = \frac{1}{|U|} \sum_{i \in U} y_i - w^T x_i \tag{2.27}$$

### 2.3.3 Utilisation des noyaux

Le fait d'admettre la mal-classification de certains exemples, ne peut pas toujours donner une bonne généralisation pour un hyperplan même si ce dernier est optimisé (figure 2.3).

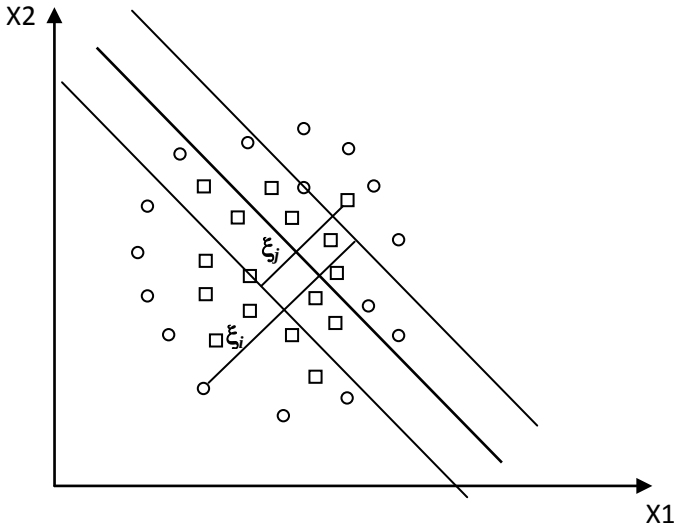


FIGURE 2.3 – Mal-adaptation de l'hyperplan aux problèmes réels de classification

Plutôt qu'une droite, la représentation idéale de la fonction de décision serait une représentation qui colle le mieux aux données d'entrainement (figure 2.4).

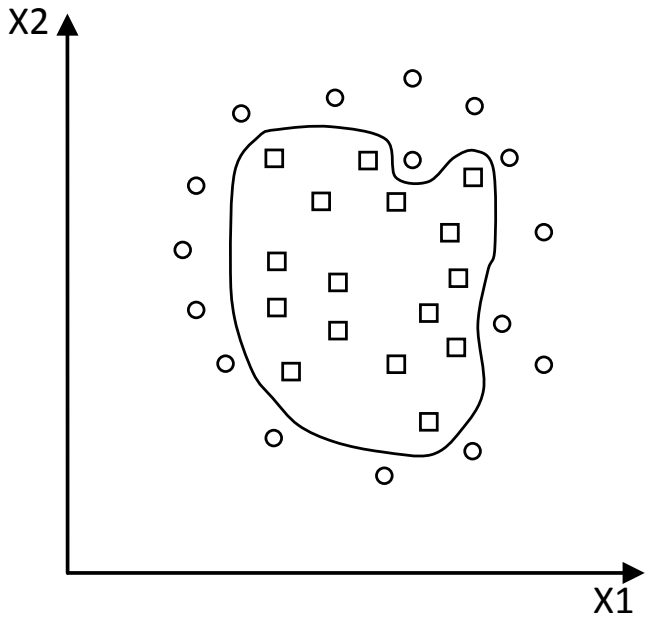


FIGURE 2.4 – Représentation idéale de la fonction de décision

La détermination d'une telle fonction non linéaire est très difficile voire impossible. Pour cela les données sont amenées dans un espace où cette fonction devient linéaire (figure 2.5), cette astuce permet de garder les mêmes modèles de problèmes d'optimisation vus dans les sections précédentes, utilisant les SVMs basées essentiellement sur le principe de séparation linéaire. Cette transformation d'espace est réalisée souvent à l'aide d'une fonction $F = \{\phi(x)|x \in X\}$ appelé *"Mapping function"* et le nouvel espace est appelé espace de caractéristiques *"Features space"*.

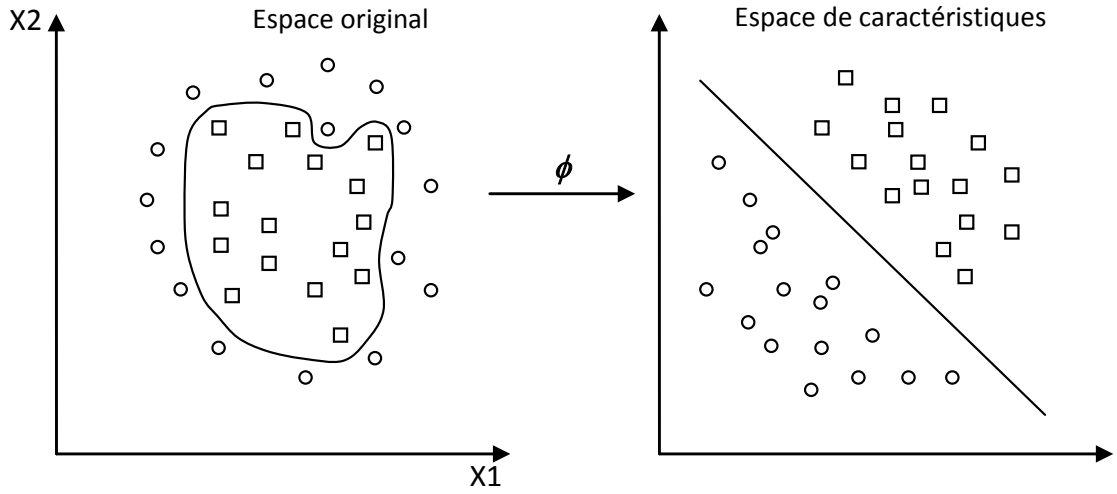


FIGURE 2.5 – Transformation d'espace

Dans ce nouvel espace de caractéristiques, la fonction objective à optimiser de l'équation (2.24) devient :

$$Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle \tag{2.28}$$

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

Dans le calcul de l'optimum de la fonction (2.28), on utilise une astuce appelée *"Noyau" ("Kernel")*, au lieu de calculer $\phi(x_i), \phi(x_j)$ et leur produit scalaire, on calcule plutôt une fonction $K(x_i, x_j)$ qui représente à la fois les deux transformations (qui peuvent être inconnues) et leur produit scalaire. Cette fonction permet de surmonter le problème de détermination de la transformation $\phi$ et permet d'apprendre des relations non linéaires par des machines linéaires. La fonction $K(x_i, x_j)$ peut être vue comme une matrice $G[n, n]$ dite de *Gram* [130] qui représente les distances entre tous les exemples :

$$\begin{bmatrix} K(x_1, x_1) & \dots & K(x_1, x_n) \\ \vdots & \ddots & \vdots \\ K(x_n, x_n) & \dots & K(x_n, x_n) \end{bmatrix}$$

Pour qu'une matrice $G$ (fonction $K$) soit un noyau, il faut qu'elle respecte les conditions de Mercer [103, 81], c-à-d qu'elle doit être semi-définie positive (symétrique et n'a pas de valeurs propres négatives). La construction de tels noyaux est largement étudiée dans [68, 98], néanmoins, il existe certains noyaux qui sont très utilisés et qui sont considérés comme standards 2.3.3. Une fois le noyau choisi, la fonction objective 2.28 peut être calculée comme suit 2.29 :

$$Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{2.29}$$

Et la fonction de décision devient :

$$H(x) = \sum_{i \in S} \alpha_i y_i K(x_i, x) + b \tag{2.30}$$

Où $S$ représente l'ensemble des vecteurs supports.

**Exemples de noyaux**

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

$$K(x_i, x_j) = x_i^T x_j \tag{2.31}$$

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

$$K(x_i, x_j) = (x_i^T x_j)^d \tag{2.32}$$

  Si $d = 1$ le noyau devient linéaire. Le noyau polynomial dit non homogène $K(x_i, x_j) = (x_i^T x_j + C)^d$ est aussi utilisé.
- Noyau RBF : Les noyaux RBF (Radial Basis functions) sont des noyaux qui peuvent être écrits sous la forme : $K(x_i, x_j) = f(d(x_i, x_j))$ où $d$ est une métrique sur $X$ et $f$ est une fonction dans $\Re$. Un exemple des noyaux RBF est le noyau Gaussien (2.33) :

$$K(x_i, x_j) = e^{\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)} \tag{2.33}$$

  Où $\sigma$ est un réel positif qui représente la largeur de bande du noyau.

### 2.3.4 Architecture générale d'une machine à vecteur support

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

Les vecteurs supports et leurs multiplicateurs sont utilisés pour calculer la fonction de décision pour un nouvel exemple. Le schéma de la figure 2.6 résume l'architecture générale d'une SVM dans le cas de la reconnaissance des chiffres manuscrits.

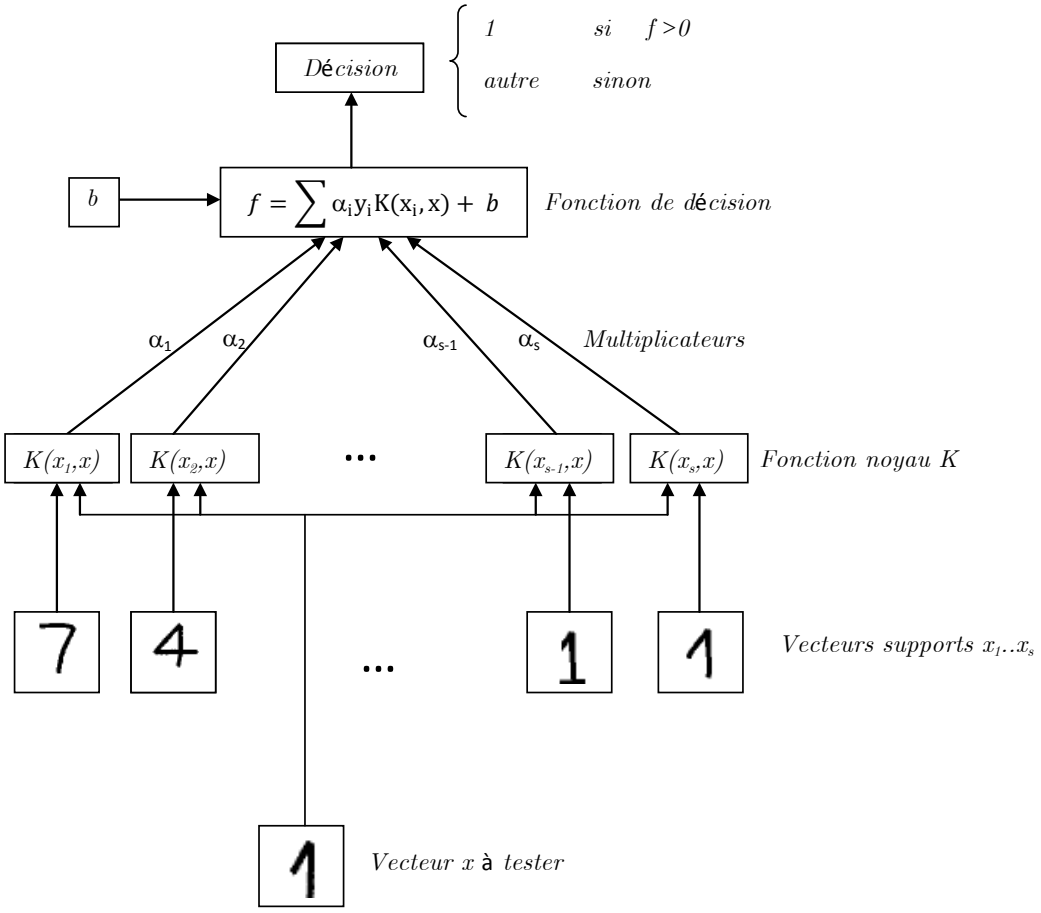


FIGURE 2.6 – Architecture d'une machine à vecteur support

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

## 2.4 SVMs multiclasse

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

Les méthodes des machines à vecteur support multiclasse, réduisent le problème multiclasse à une composition de plusieurs hyperplans biclasses permettant de tracer les frontières de décision entre les différentes classes [62, 136]. Ces méthodes décomposent l'ensemble d'exemples en plusieurs sous ensembles représentant chacun un problème de classification binaire. Pour chaque problème un hyperplan de séparation est déterminé par la méthode SVM binaire. On construit

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

### 2.4.1 Une-contre-reste (1vsR)

C'est la méthode la plus simple et la plus ancienne. Selon la formulation de Vapnik [149], elle consiste à déterminer pour chaque classe $k$ un hyperplan $H_k(w_k, b_k)$ la séparant de toutes les autres classes. Cette classe $k$ est considérée comme étant la classe positive $(+1)$ et les autres classes comme étant la classe négative $(-1)$, ce qui résulte, pour un problème à $K$ classes, en $K$ SVM binaires. Un hyperplan $H_k$ est défini pour chaque classe $k$ par la fonction de décision suivante :

$$\begin{aligned} \mathrm{H}_k(x) &= \text{signe}(\langle w_k, x\rangle + b_k) \\ &= \begin{cases} +1 & \text{si} \quad \mathrm{f}_k(x) > 0; \\ 0 & \text{sinon} \end{cases} \end{aligned} \tag{2.34}$$

La valeur retournée de l'hyperplan permet de savoir si $x$ appartient à la classe $k$ ou non. Dans le cas où il n'appartient pas à $k$ $(H_k(x) = 0)$, nous n'avons aucune information sur l'appartenance de $x$ aux autres classes. Pour le savoir, on présente $x$ à tous les hyperplans, ce qui donne la fonction de décision de l'équation (2.35) suivante :

$$k^* = \underbrace{Arg}_{(1 \leq k \leq K)} \; Max(H_k(x)) \tag{2.35}$$

Si une seule valeur $H_k(x)$ est égale à 1 et toutes les autres sont égales à 0, on conclut que $x$ appartient à la classe $k$. Le problème est que l'équation (2.35) peut être vérifiée pour plus d'une classe, ce qui produit des régions d'ambiguïté, et l'exemple $x$ est dit non classifiable. La figure 2.7 représente un cas de séparation de 3 classes.

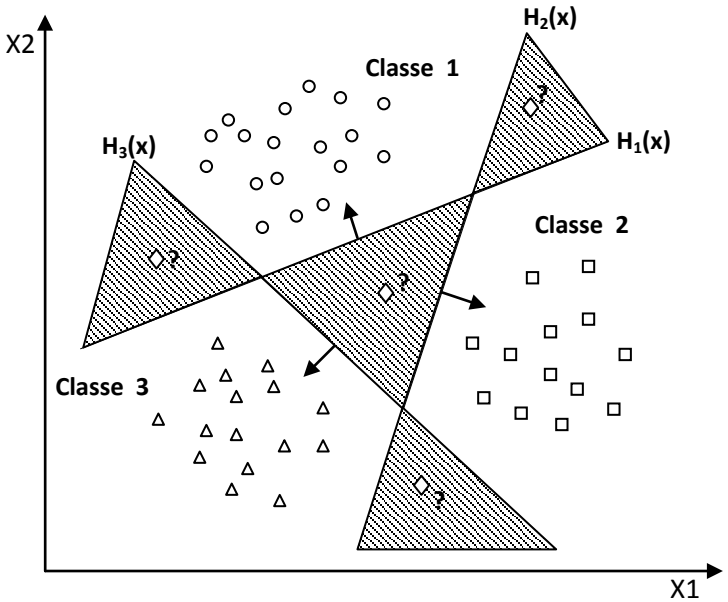


FIGURE 2.7 – Approche une-contre-reste avec des zones d'indécision

Pour surmonter cette situation, la méthode 1vsR utilise le principe de "le gagnant prend tout" ("winner-takes-all") : la classe $k$ retenue est celle qui maximise $f_k(x) = \langle w_k, x\rangle + b_k$ de l'équation (2.36).

$$k^* = \underbrace{Arg}_{(1 \leq k \leq K)} \quad Max(\langle w_k, x \rangle + b_k) \tag{2.36}$$

Géométriquement interprétée, tout nouvel exemple $x$ est affecté à la classe dont l'hyperplan est le plus loin de $x$, parmi les classes ayant $H(x) = 1$ (figure 2.8).

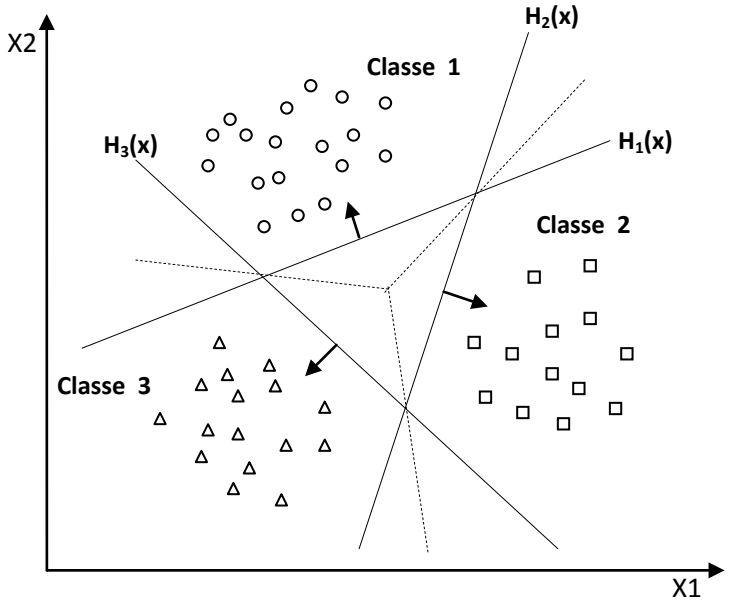


FIGURE 2.8 – Résolution des cas d'indécision dans la méthode 1vsR

La méthode 1vsR peut être utilisée pour découvrir même les cas de rejet où un exemple n'appartient à aucune des $K$ classes. Pour cela, on prend les deux fonctions de décision les plus élevées, puis on calcule leur différence, si elle est au dessous d'un certain seuil, l'exemple est rejeté.

Souvent, la méthode 1vsR est critiquée à cause de son asymétrie [130], puisque chaque hyperplan est entrainé sur un nombre d'exemples négatifs beaucoup plus important que le nombre d'exemples positifs. Par exemple dans le cas de l'OCR, le classifieur du caractère 'A' est entrainé sur des exemples positifs représentant 'A' et des exemples négatifs représentant tous les autres caractères. La méthode une contre une suivante est une méthode symétrique qui corrige ce problème.

### 2.4.2 Une-contre-une (1vs1)

Cette méthode, appelée aussi *"pairwise"*, revient à Kner et ses co-auteurs [80] qui l'ont proposée pour les réseaux de neurones. Elle consiste à utiliser un classifieur pour chaque paire de classes. Au lieu d'apprendre $K$ fonctions de décisions, la méthode 1vs1 discrimine chaque classe de chaque autre classe, ainsi $K(K-1)/2$ fonctions de décisions sont apprises.

Pour chaque paire de classes $(k, s)$, la méthode 1vs1 définit une fonction de décision binaire $h_{ks} : \Re \rightarrow \{-1, +1\}$. L'affectation d'un nouvel exemple se fait par liste de vote. On teste un exemple par le calcul de sa fonction de décision pour chaque hyperplan. Pour chaque test, on vote pour la classe à laquelle appartient l'exemple (classe gagnante). On définit pour le faire la fonction de décision binaire $H_{ks}(x)$ de l'équation 2.37.

$$\begin{aligned} \mathrm{H}_{ks}(x) &= \mathrm{signe}(\mathrm{f}_{ks}(x)) \\ &= \begin{cases} +1 & \text{si } \mathrm{f}_{ks}(x) > 0; \\ 0 & \text{sinon} \end{cases} \end{aligned} \tag{2.37}$$

Sur la base des $K(K-1)/2$ fonctions de décision binaires, on définit $K$ autres fonctions de décision (équation 2.38) :

$$H_k(x) = \sum_{s=1}^{m} H_{ks}(x) \tag{2.38}$$

Un nouvel exemple est affecté à la classe la plus votée. La règle de classification d'un nouvel exemple $x$ est donnée par l'équation 2.39 :

$$k^* = \underbrace{Arg}_{(1 \leq k \leq K)} \ (MaxH_k(x)) \tag{2.39}$$

Malheureusement, la fonction 2.39 peut être vérifiée pour plusieurs classes, ce qui produit des zones d'indécisions. La méthode de vote affecte dans ce cas, un exemple aléatoirement à l'une des classes les plus votées.

La Figure 2.9 représente un exemple de classification de trois classes avec la zone d'indécision.

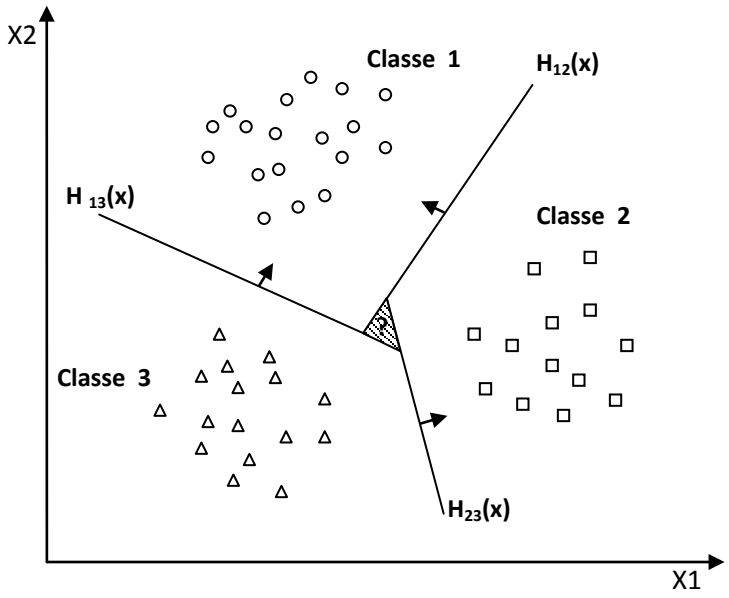


FIGURE 2.9 – Approche une-contre-une

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

### 2.4.3 Graphe de décision

C'est une méthode développée par Platt et all [118] pour résoudre le problème des zones d'indécision dans la méthode 1vs1. Premièrement, l'entrainement est effectué par la même méthode

1vs1 de la section précédente pour obtenir $K(K-1)/2$ hyperplans. Puis, au lieu d'utiliser le vote pour l'affectation des nouveaux exemples, on construit un graphe de décision. Pour cela, on définit une mesure $E_{ks}$ de la capacité de généralisation sur les différents hyperplans obtenus c-à-d pour chaque paire de classes. Cette mesure représente le rapport entre le nombre de vecteurs supports de l'hyperplan et le nombre d'exemples des deux classes correspondantes (équation 2.40).

$$E_{ks} = \frac{N_{vs}}{N_{exemples}} \tag{2.40}$$

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

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

On obtient ainsi un graphe de décision similaire à l'exemple de la figure 2.10 dont les feuilles sont les classes et les nœuds internes sont les hyperplans :

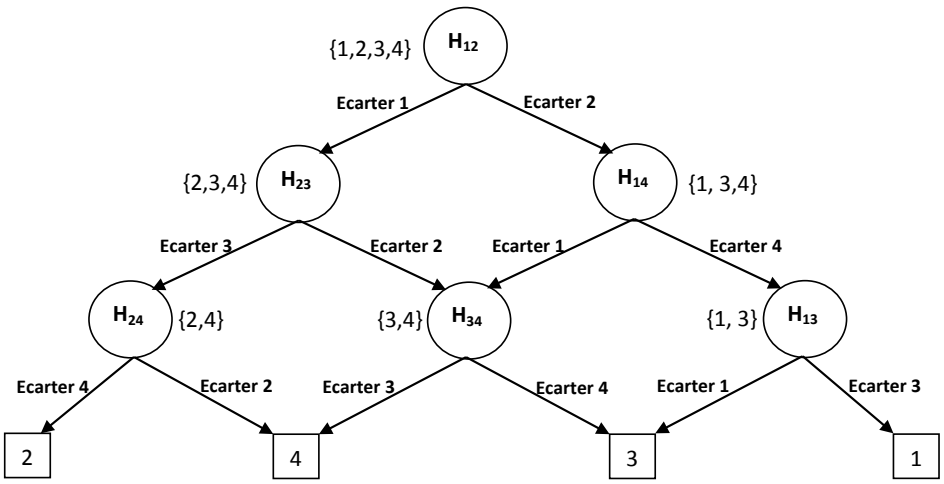


FIGURE 2.10 – Graphe de décision acyclique orienté à quatre classes

Un nouvel exemple $x$ est exposé premièrement à l'hyperplan de la racine. Si la décision est positive, on continue avec le fils gauche sinon avec le fils droit jusqu'à atteindre une feuille. La feuille atteinte représente la classe de l'exemple $x$. Contrairement à la méthode de vote, qui teste pour classifier un exemple $K(K-1)/2$ hyperplans, la méthode DAG en teste uniquement $K-1$, ce qui la rend très rapide en classification par rapport aux méthodes 1vs1 et 1vsR [143].

### 2.4.4 SVMs basées arbres de décision

Dans cette méthode, on apprend pour $K$ classes, $(K-1)$ hyperplans. Chaque hyperplan sépare une ou plusieurs classes du reste, selon un découpage choisi. On peut choisir, par exemple,

un découpage semblable à la méthode 1vsR où l'hyperplan $H_i$ sépare la classe $i$ des classes $i+1$, $i+2$,...,$K$ (cf. figure 2.11).

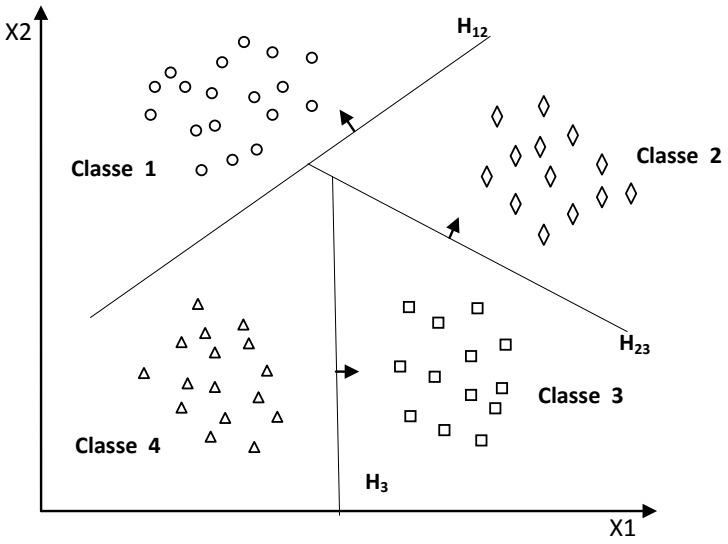


FIGURE 2.11 – SVM multiclasse par arbre de décision

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

On remarque sur la figure 2.11, que les classes d'indices inférieurs ont des zones de classification plus importantes, ce qui pose des problèmes pour la capacité de généralisation du modèle de décision obtenu. Cela montre l'importance de l'ordre de découpage choisi.

Il existe plusieurs types d'arbres de décision. Certains trient les classes selon l'ordre décroissant du nombre de leurs exemples, pour placer les classes volumineuses dans des zones importantes. D'autres méthodes [142] subdivisent, à chaque fois, les classes en deux ensembles équilibrés, ce qui génère un arbre de décision binaire équilibré. Dans plusieurs autres propositions [131, 132], on effectue des clustering selon une métrique choisie pour séparer les classes.

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

## 2.5 SVM monoclasse (Novelty detection)

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

(étranger ou outlier).

Pour la classification SVM monoclasse, il est supposé que seules les données de la classe cible sont disponibles. L'objectif est de trouver une frontière qui sépare les exemples de la classe cible du reste de l'espace, autrement dit, une frontière autour de la classe cible qui accepte autant d'exemples cibles que possible[62]. Cette frontière est représentée par une fonction de décision positive à l'intérieur de la classe et négative en dehors. La figure 2.12 représente, en deux dimensions, un cas de séparation d'une classe de toute autre classe.

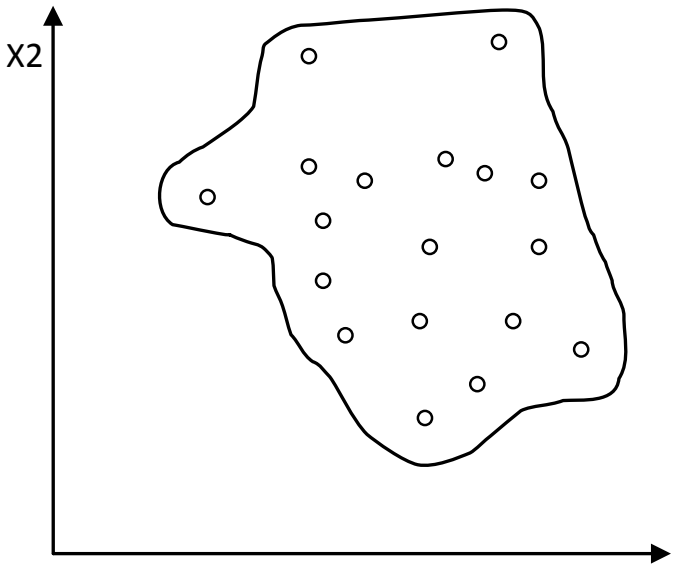


FIGURE 2.12 – Séparation des exemples d'une classe du reste de l'espace

Pour résoudre ce cas de problèmes, la technique SVM monoclasse utilise le même modèle binaire décrit dans la section 2.3 avec une astuce en plus ; l'origine de l'espace est considérée comme étant la seule instance de la classe négative. Le problème revient, donc, à trouver un hyperplan qui sépare les exemples de la classe cible de l'origine, et qui maximise la marge entre les deux (figure 2.13).

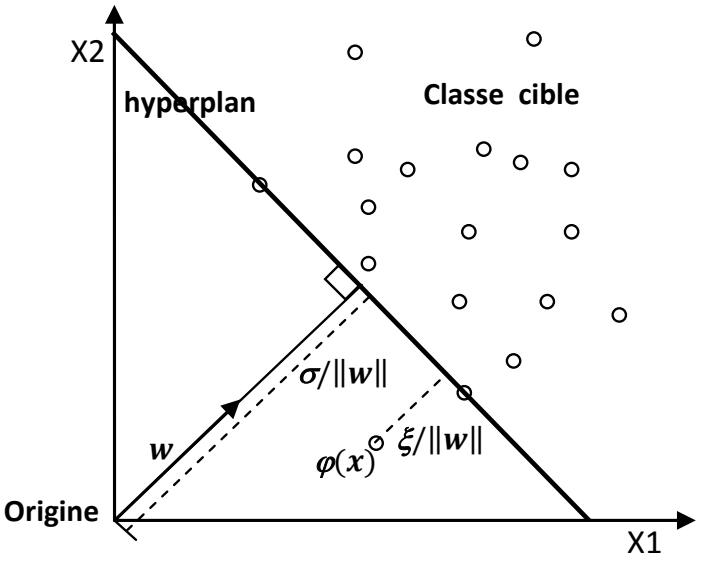


FIGURE 2.13 – SVM monoclasse à marge maximale

Le problème est modélisé par le problème primal de programmation quadratique de l'équation (2.41) dont l'objectif est de maximiser la marge et minimiser les erreurs de classification. La contrainte est la bonne classification des exemples d'entrainement.

$$\begin{cases} min_{w,\xi,\rho} \frac{1}{2} \|w\|^2 + \frac{1}{vN} \sum_{i=1}^{l} \xi_i - \rho \\ \langle w, \phi(x_i) \rangle \geq \rho - \xi_i \\ \xi_i \geq 0 \qquad i = 1, 2..N \end{cases} \tag{2.41}$$

Où $N$ est le nombre d'exemples de la classe cible, $(w, \rho)$ les paramètres permettant de localiser l'hyperplan, $\xi_i$ représentent les erreurs permises sur les exemples, pénalisées par le paramètre $v$ et $\phi$ est une transformation d'espace semblable à celle du cas binaire.
Une fois $(w, \rho)$ déterminés, tout nouvel exemple pourra être classé par la fonction de décision de l'équation (2.42) :

$$f(x) =< w, \phi(x_i) > -\rho \tag{2.42}$$

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

En fait, la résolution du problème de l'équation 2.41 est réalisée par l'introduction des multiplicateurs de Lagrange pour obtenir le problème dual de l'équation (2.43) :

$$\begin{cases} Minimiser_{\alpha} & \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \\ sous\ contraintes & \\ & \sum_{i=1}^{n} \alpha_i = 1 \\ & 0 \leq \alpha_i \leq \frac{1}{vN} \end{cases} \tag{2.43}$$

Où $K$ est un noyau qui représente la transformation d'espace $\phi$.
Une fois les $\alpha_i$ déterminés ils peuvent être dans l'un des trois cas suivants :

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

La fonction de décision pour tout exemple $x$ est donnée par l'équation 2.44 :

$$f(x) = \sum_{i=1}^{l} \alpha_i K(x_i, x) - \rho \tag{2.44}$$

Où $\rho$ peut être déterminé à partir d'un exemple $x_i$ d'apprentissage dont $\alpha_i \neq 0$ par l'équation 2.45 :

$$\rho = \sum_{j} \alpha_j K(x_j, x_i) \tag{2.45}$$

Cette méthode de résolution n'est pas la seule qui existe. Dans [145], Duin et ses co-auteurs ont proposé de résoudre le problème par la recherche de l'hypersphère de rayon minimal qui englobe tous les exemples d'apprentissage (figure 2.14).

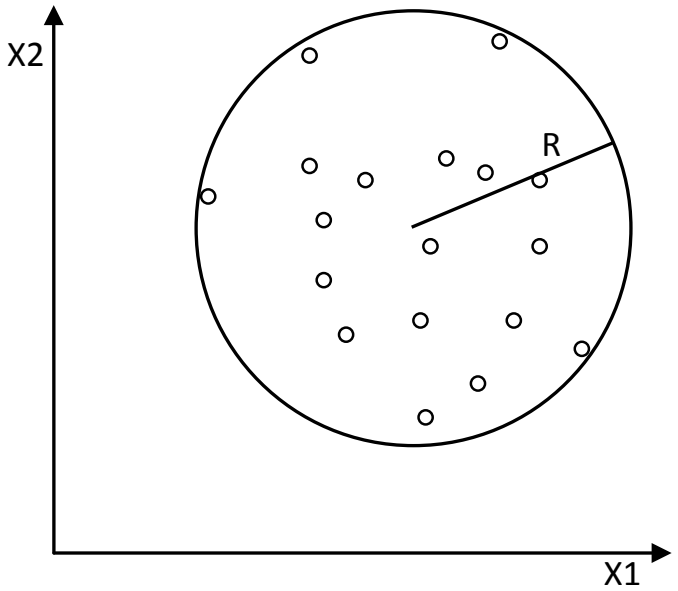


FIGURE 2.14 – SVM monoclasse à base d'hypersphère

Le problème revient à minimiser $R$ tout en gardant les exemples dans l'hypersphère (équation 2.46).

$$\begin{cases} Minimiser_{R,\xi} & \|R\|^2 + \frac{1}{vN} \sum_{i=1}^{l} \xi_i; \quad 0 < v \leq 1 \\ \text{Sous} \quad \text{contraintes} & \\ & \|\phi(x_i) - c\|^2 \leq R^2 + \xi_i \\ & \xi_i \leq 0 \qquad i = 1, 2..N \end{cases} \tag{2.46}$$

Où $R$ est le rayon de l'hypersphère, $c$ est le centre des exemples, $\xi_i$ les erreurs permises pénalisées par le paramètre $v$ et $\phi$ est une transformation d'espace. Le problème est résolut en introduisant les multiplicateurs de Lagrange $\alpha_i$ et en le transformant en le problème dual de l'équation 2.47.

$$\begin{cases} Minimiser_{\alpha} & \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) - \sum_{i} \alpha_i K(x_i, x_i) \\ sous\ contraintes & \\ & \sum_{i=1}^{n} \alpha_i = 1 \\ & 0 \leq \alpha_i \leq \frac{1}{vN} \end{cases} \tag{2.47}$$

Après la détermination des $\alpha$ optimaux, la fonction de décision pourra être calculée pour un nouvel exemple $x$ par la fonction de l'équation (2.48).

$$f(x) = R^2 - \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) + 2 \sum_i \alpha_i K(x_i, x) - K(x, x) \tag{2.48}$$

Cette méthode est moins précise que la précédente à cause de la forme sphérique de la frontière de la décision qui limite les capacités de généralisation de la machine apprise.

## 2.6 SVM pour la régression

Dans leur origine, les SVMs ont été développées pour des problèmes de classification. Cependant, leur nature leur permet de résoudre également des problèmes de régression. La régression est un cas particulier de classification où les classes des exemples ne sont pas dénombrables c-à-d continues. Le problème consiste à trouver, en utilisant $D = \{(x_1, y_1), .., (x_n, y_n)\}$, une fonction $\hat{f} : \Re^m \rightarrow \Re$ qui rapproche le plus possible des $y_i$, en d'autre terme qui minimise la différence entre les $\hat{f}(x_i)$ et les $y_i$.

Souvent, $\hat{f}$ est considérée comme fonction linéaire : $\hat{f} = \langle w, x \rangle + b$, où $w$ est un vecteur et $b$ est un scalaire. Le problème revient donc à trouver un hyperplan caractérisé par $w^*$ et $b^*$ qui minimise l'écart global entre $\hat{f}$ et les $y_i$ (équation 2.49).

$$(w^*, b^*) = argmin_{w,b} \sum_{i=1}^{n} (y_i - \langle w, x_i \rangle - b)^2 \tag{2.49}$$

Pour résoudre ce problème, les SVMs utilisent une astuce semblable a celle utilisée en classification [15, 51, 62]. On propose de modéliser la fonction de régression par un hyperplan qui se situe au centre d'un hyper-tube de largeur $2\epsilon$ contenant tous les exemples d'entrainement (figure 2.15.a).

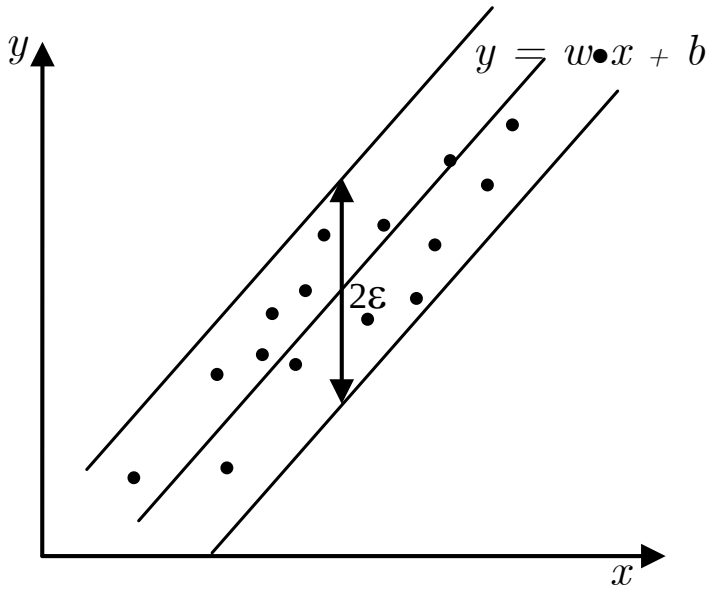


FIGURE 2.15 – Hyper-tube modélisant la fonction de régression

Plusieurs hyper-tubes, de largeur $2\epsilon$ contenant tous les exemples d'entrainement, peuvent exister. L'hyper-tube optimal est celui qui minimise la distance entre les exemples d'entrainement

et ses frontières, autrement dit, qui maximise la distance des exemples de l'hyperplan du centre (figure 2.15).

La détermination de l'hyper-tube optimal est semblable à la détermination de l'hyperplan optimal de marge maximale dans le cas de classification. On doit donc rechercher un hyper-tube de marge maximale avec tous les exemples d'entrainement à l'intérieur. Par une analyse similaire à celle du problème de classification, la solution du problème de régression est réduite à la résolution du problème dual d'optimisation quadratique de l'équation (2.50).

$$\begin{cases} \textit{Maximiser}_{\alpha,\alpha'} & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i-\alpha'_i)(\alpha_j-\alpha'_j)\langle x_i,x_j\rangle \\ & -\epsilon\sum_{i=1}^{n}(\alpha_i+\alpha'_i)+\sum_{i=1}^{n}y_i(\alpha_i-\alpha'_i) \\ \textit{sous contraintes} & \\ & \sum_{i=1}^{n}(\alpha_i-\alpha'_i)=0 \\ & 0\leq\alpha_i,\alpha'_i\leq C \end{cases} \tag{2.50}$$

Où les $\alpha_i$ et les $\alpha'_i$ sont les coefficients des exemples respectivement au dessus et au dessous de l'hyperplan et C est un paramètre pour leur pénalisation. La fonction de sortie $\hat{f}(x)$ peut être donnée par l'équation (2.51).

$$\hat{f}(x)=\sum_{i=1}^{n}(\alpha_i-\alpha'_i)\langle x_i,x\rangle+b \tag{2.51}$$

Où $b$ est calculé à partir d'un exemple dont $0<\alpha_i<C$ (vecteur support) par l'équation (2.52).

$$b=y_i-\langle w,x_i\rangle-\epsilon \tag{2.52}$$

### Utilisation des noyaux

Parmi les motivations les plus solides du développement des machines à vecteur support pour la régression, est leur extension simple aux cas non linéaires, grâce à l'utilisation des noyaux. En effet, d'une manière similaire au cas de classification, on fait une transformation d'espace $\phi$ pour se trouver toujours face à une régression linéaire. La transformation d'espace inverse $\phi^{-1}$, permet de retourner à l'espace d'origine après la résolution dans le nouvel espace (figure 2.16).

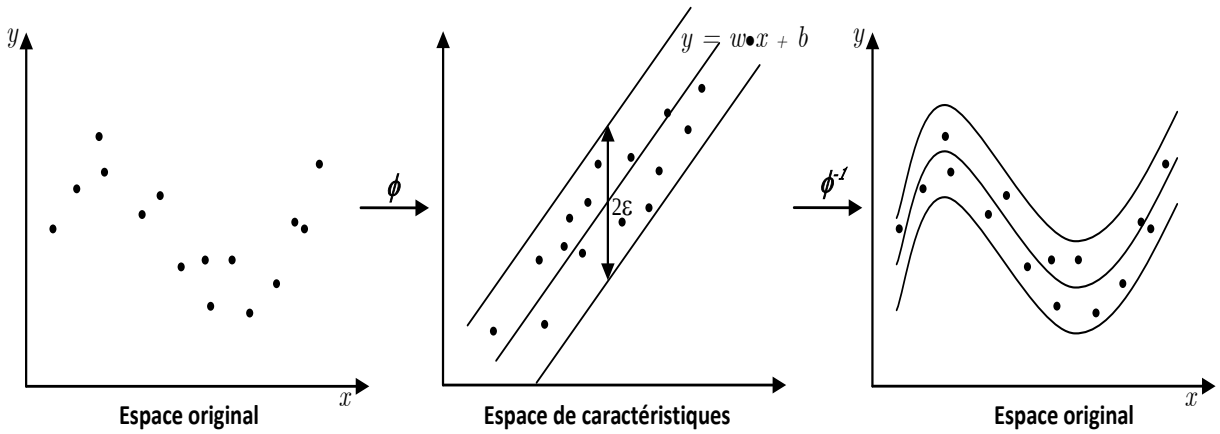


FIGURE 2.16 – Utilisation des noyaux pour la résolution de la régression non linéaire

Comme dans le cas de classification, la transformation $\phi$ et son inverse sont réalisées grâce à une fonction réelle $K(x_i, x_j)$ appelée *Noyau (Kernel)*. Le produit scalaire dans les équation (2.50) et (2.51) est remplacé par la fonction du noyau.

## 2.7 Implémentation des SVMs

L'implémentation des SVMs pour la classification binaire consiste à la résolution du problème dual de programmation quadratique de l'équation (2.53) pour déterminer l'hyperplan de marge maximale.

$$\begin{cases} \text{Maximiser} & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{sous contraintes} & \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \end{cases} \tag{2.53}$$

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

La résolution de ce problème consiste à déterminer les $\alpha_i$ optimaux. Si le nombre d'exemples est modeste (de l'ordre de 1000), les méthodes classiques de programmation quadratique tel que les méthodes quasi-Newton [108] ou les méthodes du point intérieur [148], peuvent être utilisées. Si par contre le nombre d'exemples est important (le cas habituel), des méthodes d'optimisation sont indispensables pour résoudre le problème en un temps acceptable.

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

Pour résoudre ces problèmes, plusieurs méthodes ont été développées. La méthode de *shnuking* [112], effectue l'entrainement sur un nombre limité d'exemples, choisis par une heuristique [130, 36], puis ajoute itérativement les autres jusqu'à atteindre l'hyperplan optimal. Parmi les implémentations classiques de cette méthode on trouve le SVMlight [72].

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

### Optimisation séquentielle minimale

L'algorithme d'optimisation séquentielle optimale (SMO) a été proposé premièrement par Platt et all en 1999 [117], c'est l'algorithme le plus utilisé actuellement pour les problèmes de grande taille. Cet algorithme pousse la décomposition des $\alpha_i$ à son maximum : à chaque itération, uniquement deux multiplicateurs de Lagrange $\alpha_i$ du problème dual sont optimisés. En effet, la première contrainte du problème de l'équation (2.53) implique que le plus petit nombre de $\alpha_i$ qui peuvent être optimisés conjointement est de 2. À Chaque fois qu'un multiplicateur est mis à jour, un autre multiplicateur au moins doit être ajusté afin de maintenir la contrainte satisfaite.

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

Le point fort de l'algorithme est que l'optimisation des deux multiplicateurs choisis se fait analytiquement, ce qui réduit considérablement le temps d'entrainement. En plus, l'algorithme ne fait aucune opération matricielle et n'a pas besoin de maintenir la matrice du noyau en mémoire (voir [117, 36, 130] pour plus de détails).

Plusieurs optimisations peuvent être ajoutées à l'algorithme pour l'accélérer davantage. Premièrement, la matrice du noyau K peut être gérée par une méthode de cache tel que LRU (least

recently used), pour garder une simple partie de la matrice en mémoire et mettre à jour uniquement les entrées les plus anciennes. Selon [53] cette technique peut garantir de trouver jusqu'à 80 % des éléments dans une cache d'une taille de 10 % de la matrice K.

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

Les auteurs de [128, 129, 53] proposent de larguer les exemples dont les multiplicateurs correspondants atteignent leurs limites supérieures ou inférieures (0 ou $C$), au cours de progression de l'algorithme. L'idée consiste à écarter les exemples $x_i$ dont les $\alpha_i = 0$, au cours de progression de l'algorithme puisque ce sont uniquement les exemples avec $\alpha_i \neq 0$ qui influencent la solution finale.

De la même manière l'algorithme SMO est utilisé pour implémenter la régression et la SVM monoclasse, les détails de ces variantes peuvent être trouvés dans [36, 130].

Divers packages d'implémentation du SMO peuvent être trouvés dans la littérature, particulièrement LIBSVM [28] et SVMTORCH [32].

La parallélisation de l'algorithme SMO peut l'accélérer considérablement dés lors qu'il utilise lui même une subdivision du problème dual en de plus petits problèmes. Les auteurs de [27] proposent un algorithme qui subdivise l'ensemble d'entrainement en des petites partitions traitée chacune par un processeur différent.

## 2.8 Tuning et évaluation du modèle

L'apprentissage supervisé effectué par la méthode SVM utilise une partie de l'ensemble d'exemples d'un espace, pour calculer un modèle de décision qui sera généralisé sur l'ensemble de tous les exemples de l'espace. Il est très important d'avoir des mesures permettant de qualifier le comportement du modèle appris sur les exemple qui ne sont pas utilisées lors de l'entrainement. Ces métriques sont calculées soit sur les exemples d'entrainement eux mêmes ou sur des exemples réservés à l'avance pour les tests.

### 2.8.1 Métriques de performances

Deux métriques sont généralement utilisées :

#### 2.8.1.1 Taux de reconnaissance

La métrique intuitive utilisée est la précision du modèle appelée aussi le taux de reconnaissance. Elle représente le rapport entre le nombre d'exemples correctement classés et le nombre total d'exemples testées. L'équation (2.54) donne la formule utilisée.

$$\begin{aligned} &P = \tfrac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i)) \\ &\text{avec,} \\ &L = \begin{cases} 1 & si \quad y_i = \hat{f}(x_i) \\ 0 & \text{sinon} \end{cases} \end{aligned} \tag{2.54}$$

Généralement, la précision est donnée sous forme de pourcentage ce qui nécessite de multiplier la précision de l'équation 2.54 par 100.

#### 2.8.1.2 Matrice de confusion

La mesure précédente donne le taux d'erreurs commises par le modèle de décision obtenu ($100 - \textit{précision}$), mais ne donne aucune information sur la nature de ces erreurs.

Dans la plupart des cas d'application, il est très important de connaître la nature des erreurs commises. Par exemple, lors de l'utilisation d'un modèle de décision pour des objectifs médicaux,

considérer un échantillon non cancéreux alors qu'il l'est, est beaucoup plus grave de considérer un échantillon cancéreux alors qu'il ne l'est pas.

Dans le cas de classification binaire, le résultat de test d'un modèle peut être une possibilité parmi quatre :

$$\begin{cases} \hat{f}(x_i) = +1 \quad et \quad y_i = +1 \quad \text{correcte} \quad \text{positive} \\ \hat{f}(x_i) = +1 \quad et \quad y_i = -1 \quad \text{fausse} \quad \text{positive} \\ \hat{f}(x_i) = -1 \quad et \quad y_i = -1 \quad \text{correcte} \quad \text{négative} \\ \hat{f}(x_i) = -1 \quad et \quad y_i = +1 \quad \text{fausse} \quad \text{négative} \end{cases} \tag{2.55}$$

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

La matrice de confusion (table 2.1) est une matrice qui contient en lignes les observations $y$, et en colonnes les prédictions $\hat{f}(x)$. Les éléments de la matrice représentent le nombre d'exemples correspondants à chaque cas.

TABLE 2.1 – Matrice de confusion pour la classification binaire

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

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

$$P = \frac{CP + CN}{CP + FP + CN + FN} \tag{2.56}$$

Deux autres mesures sont utilisées dans la littérature : la sensitivité $Sv$ et la spécificité $Sp$. La sensitivité représente le rapport entre les observations positives correctement prédites et le nombre des observations positives, et la spécificité représente le rapport entre les observations négatives correctement prédites et le nombre total des observations négatives (équation 2.57).

$$\begin{cases} Sv = & \frac{CP}{CP+FN} \\ Sp = & \frac{CN}{CN+FP} \end{cases} \tag{2.57}$$

Une autre métrique calculée à base de la sensitivité et la spécificité est utilisé par les auteurs de [69]. C'est la moyenne harmonique (équation 2.58).

$$\text{Moyenne} \quad \text{harmonique} = \frac{2 \times Sv \times Sp}{Sv + Sp} \tag{2.58}$$

### 2.8.2 Évaluation

La qualité d'un modèle de décision obtenu par la méthode SVM dépend de plusieurs paramètres, à savoir le paramètre de pénalisation sur les multiplicateurs de Lagrange C, le noyau utilisé et ses paramètres ($\sigma$ dans le cas du noyau Gassien) et les exemples utilisés pour l'entrainement (les vecteurs supports).

Le choix des valeurs de ces paramètres se fait actuellement à travers plusieurs essais et évaluation pour atteindre des performances satisfaisantes du modèle. Les paramètres $(C^*, K^*, \sigma^*)$

optimaux pour un modèle donné sont les paramètres qui lui permettent de donner une précision de 100%.

Cette situation serait idéale si l'ensemble des exemples représentait parfaitement l'ensemble de tous les exemples possibles. Le modèle appris peut donner une très grande précision face aux exemples d'entrainement, mais se comporte très mal avec les nouveaux exemples.

Cela représente un phénomène très connu en apprentissage qui est le sur-apprentissage ou l'apprentissage par cœur. Le sur-apprentissage donne, généralement, des modèles à faible capacité de généralisation, et par conséquent la mesure de précision n'est pas suffisante pour qualifier les performances d'un modèle. Les méthodes d'évaluation permettent de tirer des conclusions sur le comportement d'un modèle face à tout l'espace d'exemples en limitant l'influence des exemples d'entrainement et du bruit qui peut y exister (erreurs d'étiquetage, erreurs d'acquisition, ...) et leur ordre sur le modèle appris.

#### 2.8.2.1 Méthode HoldOut

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

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

#### 2.8.2.2 Validation croisée

Pour minimiser l'influence du choix du partitionnement de l'ensemble des exemples, la validation croisée subdivise l'ensemble d'entrainement initial en $k$ sous ensemble disjoints $D_1, D_2, .., D_k$ de même taille. L'entrainement et le test sont effectués $k$ fois. A l'itération $i$ le sous-ensemble $D_i$ est réservé pour le test et le reste des exemples sont utilisés pour entrainer le modèle. La précision finale du modèle est égale à la moyenne des k précisions de test.

La méthode Leave-One-Out [62] est un cas particulier de la validation croisée où $k = N$. Á chaque itération, le modèle est entrainé sur $N - 1$ exemples et testé sur l'exemple exclu de l'entrainement. On obtient à la fin $N$ précisions, la précision du modèle est égale à leur moyenne.

#### 2.8.2.3 Le Bootstrap

La méthode de Bootstrap [49], appelée aussi échantillonnage par remplacement, entraine le modèle sur un ensemble de $N$ exemples choisis aléatoirement de l'ensemble des exemples, des exemples peuvent être choisis plus d'une fois et d'autre ne se seront pas choisis du tout. Les exemples non choisis pour l'entrainement sont utilisés pour le test. Cette opération peut être répétée plusieurs fois pour obtenir une précision moyenne du modèle.

Parmi les méthodes de Bootstrap les plus utilisées, la méthode ".632" qui tire son nom du fait que 63.2 % des exemples contribuent à l'entrainement et les restants (36.8%) contribuent aux tests.

À chaque prélèvement, un exemple a une probabilité $1/N$ d'être choisi et $(1 - 1/N)$ de ne pas l'être, et puisqu'on répète le prélèvement $N$ fois, chaque exemple aura une probabilité de $(1 - 1/N)^N$ de ne pas être choisi du tout dans un ensemble d'entrainement. Si $N$ est grand cette probabilité approche de $e^{-1} = 0.368$. La méthode répète le processus $k$ fois et la précision finale $P$ est donnée par :

$$P = \sum_{i=1}^{k} (0.632 \times Pi_{test} + 0.368 \times Pi_{entr}) \tag{2.59}$$

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