Compte rendu

Support Vector Machine : Implémentation et Application

Souheib Ben Mabrouk

INDP2E

19 septembre 2021

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. [1]

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 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.

L'objectif de ce rapport est de présenter la méthode dans le cadre du cours de l'optimisation ainsi que implémenter quelques cas d'étude en se basant sur l'exemple traité en cours pour aller ensuite plus loin dans l'exploration de la méthode par des exemples plus délicats.

Pour ce faire, j'ai choisi de travailler avec **Python 3.8** disponible en ligne par le recours à **Google Colaboratory** qui me facilite le travail grâce à sa simplicité et son efficacité en termes de temps d'exécution.
D'autre part, j'utiliserai le système de composition de documents **LATEX** afin d'avoir un rapport bien conçu avec les formulations mathématiques nécessaires.

Fig. 1 – Logo de python

Fig. 2 – Logo de google colab

Fig. 3 – Logo de LATEX

Table des matières

Table des figures

Chapitre 1

Principe

1.1 Présentation

Les machines à vecteurs de support (SVM) sont des systèmes d'apprentissage qui utilisent un espace d'hypothèses de fonctions linéaires dans un espace de caractéristiques de haute dimension, entraînées avec un algorithme d'apprentissage issu de la théorie de l'optimisation qui met en œuvre un biais d'apprentissage dérivé de la théorie de l'apprentissage statistique. Le but du SVM est de trouver l'hyperplan optimal qui divise les deux classes. Il peut y avoir différents plans qui peuvent diviser les deux classes, mais l'objectif principal est de trouver un plan qui permet d'obtenir une marge maximale entre les classes. Cela signifie qu'il faut choisir l'hyperplan de telle sorte que la distance entre l'hyperplan et le point de données le plus proche soit maximisée. [2]

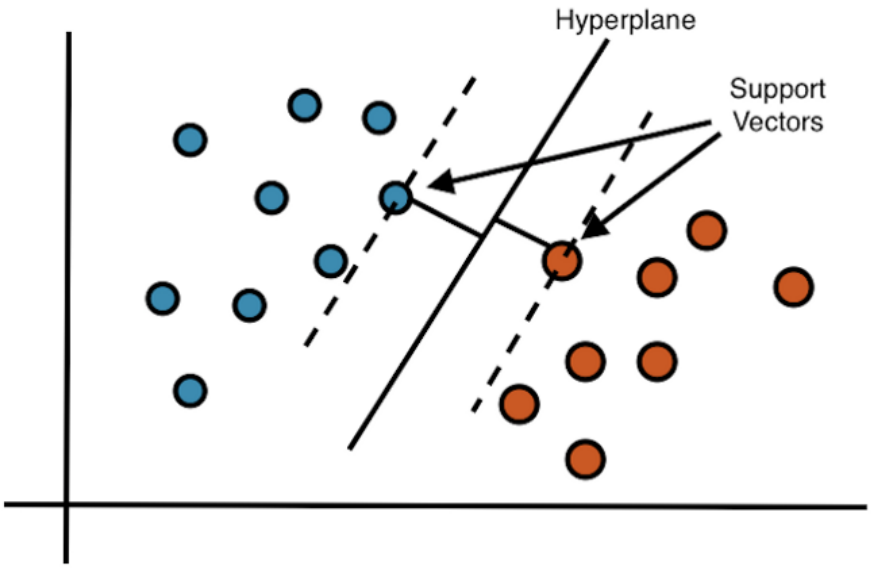

Fig. 1.1 – Exemple d'une classification SVM

1.2 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.

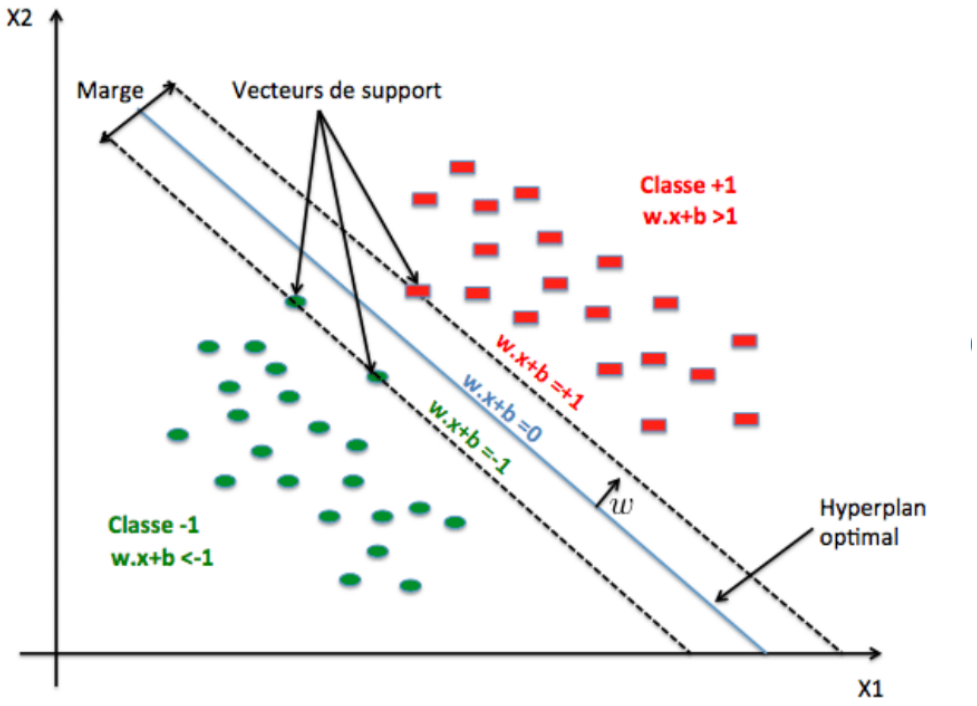

Fig. 1.2 – SVM binaire

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).

1.3 SVM à marge dure

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

$$H(x) = w^T x - b \tag{1.1}$$

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 :

$$\begin{cases} Classe = 1 & Si\ H(x) > 0 \\ Classe = -1 & Si\ H(x) < 0 \end{cases} \tag{1.2}$$

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 :

$$\begin{cases} Classe = 1 & Si\ H(x) > 1 \\ Classe = 1 & Si\ H(x) < -1 \end{cases} \tag{1.3}$$

On trouve les inégalités précédentes qui sont équivalentes à l'équation :

$$yi(w^T xi - b) \geq 1, i = 1..n \tag{1.4}$$

L'hyperplan **wT x - b =0** représente un hyperplan séparateur des deux classes, et la distance entre cet hyperplan et l'exemple le plus proche s'appelle la marge.
La région qui se trouve entre les deux hyperplans **wT x - b = - 1** et **wT 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"**.
[3] En supposant que les données d'apprentissage ne contiennent pas des données bruitées et que les données de test suivent la même probabilité que celle des données d'entraînement, l'hyperplan de marge maximale va certainement maximiser la capacité de généralisation de la machine d'apprentissage.

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

$$\frac{aw}{||w||} + x = 0 \tag{1.5}$$

Pour trouver l'hyperplan séparateur qui maximise la marge, on doit déterminer„ le vecteur w qui possède la norme euclidienne minimale et qui vérifie la contrainte de l'équation (1.4), de bonne classification des exemples d'entrainement. L'hyperplan séparateur optimal peut être obtenu en résolvant le problème de l'équation :

$$\begin{cases} Minimiser & \frac{||w||^2}{2} \\ sous\ contraintes & \\ yi(w^T xi - b) \geq 1\ , i = 1..n & \end{cases} \tag{1.6}$$

C'est la même équation obtenue dans l'exemple du cours et qu'on a expliqué la méthode basée sur l'étude du Lagrangien pour la résoudre à l'aide du théorème de de **Kuhn et Tucker**

Dans ce cas le problème est convertit en un problème dual équivalent sans contraintes qui introduit les multiplicateurs de Lagrange :

$$Q(w,b,s) = \frac{wTw}{2} - \sum_{i=1}^{n}(si * [yi(wTxi - b) - 1] \tag{1.7}$$

les si 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 si.

En simplifiant d'avantage la fonction de décision par KKT et projection dans le domaine de validation des contraintes, on peut obtenir la fonction de décision simplifiée :

$$f(x) = sgn(\sum_{i=1}^{m} si * yi(xi.x - b) \tag{1.8}$$

Ce qui montre l'utilité des vecteurs à support dans la phase de généralisation.

On va détailler la méthode de résolution dans l'exemple d'implémentation .

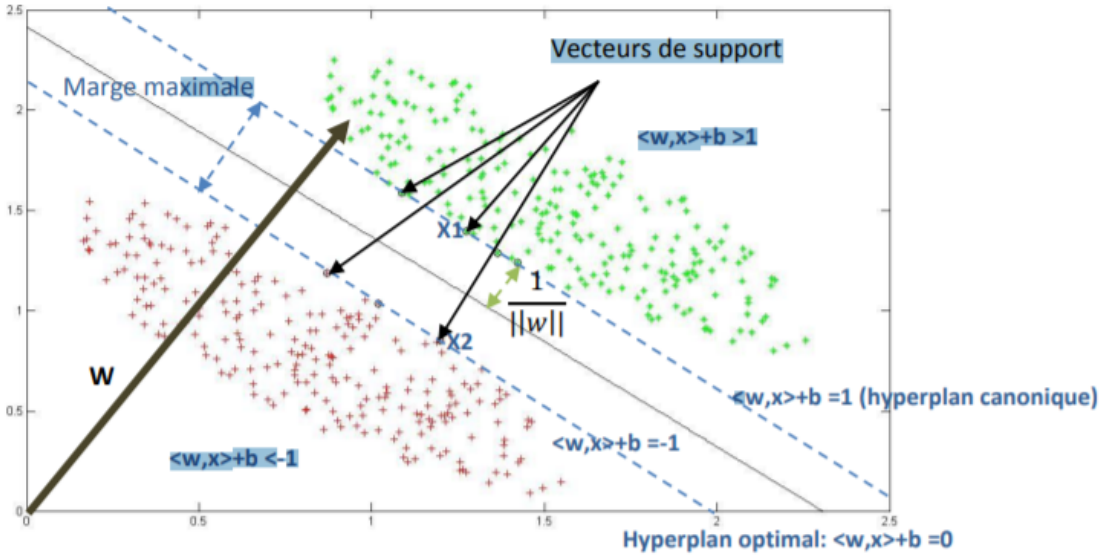

Fig. 1.3 – SVM à marge dure

1.4 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 affectera crucialement l'hyperplan.
Dans le cas où les données ne sont pas linéairement séparables, ou contiennent du bruit les contraintes ne peuvent être vérifiées, et il y a nécessité de les relaxer un peu.
Ceci peut être fait en admettant une certaine erreur de classification des données ce qui est appelé "SVM à marge souple (Soft Margin) [4]

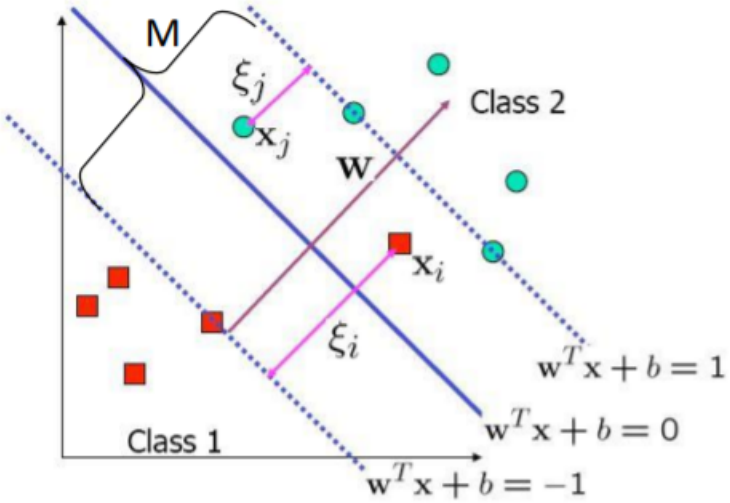

Fig. 1.4 – SVM à marge souple

On introduit alors sur les contraintes des variables li dites de relaxation.

$$yi(w^T xi - b) \geq 1 - li, i = 1..n \tag{1.9}$$

Grâce aux variables de relaxation non négatives li , un hyperplan séparateur existera toujours. Si li < 1, xi ne respecte pas la marge mais reste bien classé, sinon xi 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.
le problème devient donc :

$$\begin{cases} Minimiser & \frac{||w||^2}{2} + C\sum_{i=1}^{n} li \\ sous\ contraintes & \\ yi(w^T xi - b) \geq 1 - li\ , i = 1..n & \\ li \geq 0 & \end{cases} \tag{1.10}$$

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.

Chapitre 2

Implémentation

On implémente dans cette partie l'exemple d'application étudier en classe.
L'objectif de la classification est de regrouper les individus selon des critères définis sur ces paramètres. Ceci revient à définir des frontières qui séparent entre les classes d'individus. Un exemple important est celui de la classification de l'état nutritionnel chez l'adulte selon l'OMS.
Dans le cadre de notre étude, **on considère un individu comme hypertendu si sont IMC $>$ 20 [kg/m¨2]**.

2.1 Préparation de la dataset

Pour avoir des résultats bien raffinés, on construit la dataset de façon à éliminer le bruit et bien équitter les individus.
Pour cela, on construit une dataset à 245 individus dont 150 sont hypertendus.
Le code suivant traduit la création et la visualisation de la dataset.

```
import random
import matplotlib.pyplot as plt
import numpy as np
dataset=[(1,random.uniform(1.5,1.8),random.uniform(85,125)) for i in range(150) ]
dataset+=[(-1,random.uniform(1.7,1.9),random.uniform(40,70)) for i in range(90) ]
dataset+= [(-1,1.5,44.0),(-1,1.6,51.0),(-1,1.6,45.0),(-1,1.65,52.0),(-1,1.7,50.0)]
y=np.array( [ i[0]  for i in dataset ])
#y est vecteur colonne contenant les etiquettes {1,-1}

X=np.array( [ i[1:]  for i in dataset ])
#X est une matrice de Nindividu lignes et 2 colonnes (L,T)

colors=np.array(["red" if i ==1 else "blue" for i in y ])
plt.scatter(X[:, 0], X[:, 1],  marker='o', c=colors)
```

Le résultat de ce code est donnée par la figure :

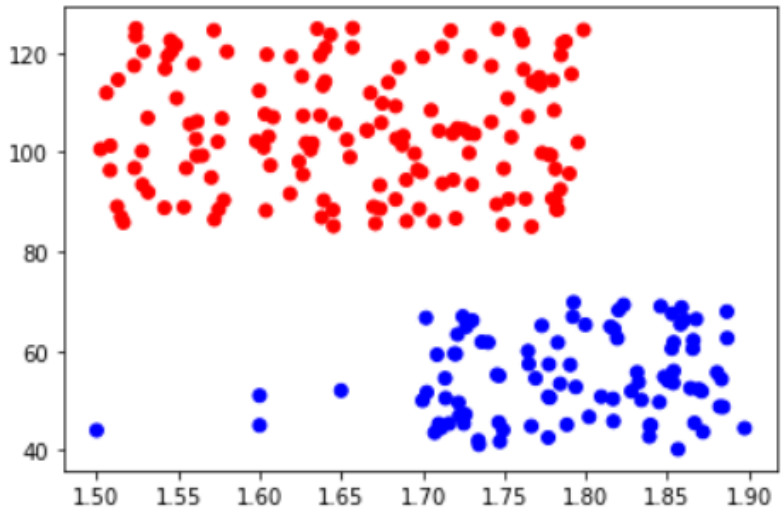

Fig. 2.1 – Visualisation de la dataset

On remarque que, comme prévu, les deux classes sont visuellement séparables par des SVM.

2.2 Traduction des équations des SVM

Partant de l'équation (1.4) yi(wT xi - b) ≥ 1 ,$i = 1..n$ On écrit les contraintes sous la forme démontrée dans le support du cours :

$$l = max(0,1 - yi(w^T xi - b) \tag{2.1}$$

Cette équation traduit le fait qu'un élement positionné au dessous de la limite des non-hypertendus avec label=-1 ou au dessus de la limite des hypertendus avec un label=1 alors il est bien classé.

Ainsi, on écrit :

$$J = \lambda.||w^2|| + \frac{\sum_{i=1}^{n} max(0,1 - yi(w^T xi - b))}{n} \tag{2.2}$$

Donc si yi . f(x) >= 1 alors

$$Ji = \lambda||w^2|| \tag{2.3}$$

Alors,

$$\begin{cases} \frac{\partial Ji}{\partial wk} = 2\lambda.wk \\ \frac{\partial Ji}{\partial b} = 0 \end{cases} \tag{2.4}$$

Sinon :

$$J = \lambda.||w^2|| + 1 - yi(w^T xi - b) \tag{2.5}$$

Alors,

$$\begin{cases} \frac{\partial Ji}{\partial wk} = 2\lambda.wk - yi.xi \\ \frac{\partial Ji}{\partial b} = yi \end{cases} \tag{2.6}$$

On applique **l'algorithme du gradient descendant avec projection :**

$$\begin{cases} p = 0.001 \quad , w = O_{75} \\ Pour\ k = 0...N_{iterations} \\ w = w - p.dw \\ b = b - p.db \\ fin\ pour \end{cases} \tag{2.7}$$

Remarque :

- La projection dans l'espace des valeurs admissibles est réalisée par l'équation (2.1)
- La fonction J est **convexe** donc le gradient descendant converge essentiellement vers **le minimum absolu** de la fonction.

2.3 Implémentation de l'algorithme en Python

Après avoir mettre en oeuvre la modélisation mathématique du problème et l'optimiser dans le cadre des contraintes données, on code le raisonnement antérieur sous forme d'un code python avec des commentaires à chaque étape pour expliquer le choix des variables et des opérations effectuées.

```
import random
import matplotlib.pyplot as plt
import numpy as np

class mySVM:
    def __init__(self, pas=0.001, lamda=0.0001, N=10000):
        self.lr = pas
        self.lamda = lamda
        self.N = N          #nbre d'iterations
        self.w = None
        self.b = None
```

```
    def fit(self, X, y):
        Nindividus, Nparametres = X.shape
        self.w = np.zeros(Nparametres)
        self.b = 0
        #initialisation

        for _ in range(self.N):
            for idx, x_i in enumerate(X):
                condition = y[idx] * (np.dot(x_i, self.w) - self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lamda * self.w)
                else:
                    self.w -= self.lr * (
                        2 * self.lamda * self.w - np.dot(x_i, y[idx])
                    )
                    self.b -= self.lr * y[idx]
    #Cette partie de code permet de calculer le gradient selon les conditions (2.3)et(2.5)
    # et l'evaluer pour chaque w(k) afin de  calculer w(k+1)

    def predict(self, X):
        g = np.dot(X, self.w) - self.b
        return np.sign(g)
    #c'est l'implementation de l'equation (1.8)

donnee = mySVM()
donnee.fit(X, y)
predictions = donnee.predict(X)
print(donnee.w, donnee.b)

def visualize_svm():
    def hyperplan(x, w, b, decalage):
        return (-w[0] * x + b + decalage) / w[1]
    # calcul de l'equation de l'hyperplan

    fig = plt.figure()
    a = fig.add_subplot(1, 1, 1)
    plt.scatter(X[:, 0], X[:, 1],  marker='o', c=colors)
    x0_1 = np.amin(X[:, 0])
    x0_2 = np.amax(X[:, 0])

    x1_1 = hyperplan(x0_1, donnee.w, donnee.b, 0)
    x1_2 = hyperplan(x0_2, donnee.w, donnee.b, 0)

    x1_1_m = hyperplan(x0_1, donnee.w, donnee.b, -1)
    x1_2_m = hyperplan(x0_2, donnee.w, donnee.b, -1)

    x1_1_p = hyperplan(x0_1, donnee.w, donnee.b, 1)
    x1_2_p = hyperplan(x0_2, donnee.w, donnee.b, 1)
```

```
    #Les points x0_1, x0_2, x1_1 , x1_2, x1_1_m , x1_2_m , x1_1_p et x1_2_p  permetteront
    #de dessiner les droites des SVM et la droite de l'hyperplan median.



    a.plot([x0_1, x0_2], [x1_1, x1_2], "y")
    #dessiner l'hyperplan median en jaune

    a.plot([x0_1, x0_2], [x1_1_m, x1_2_m], "g--")
    #dessiner le premier SVM en vert

    a.plot([x0_1, x0_2], [x1_1_p, x1_2_p], "g--")
    #dessiner le deuxi me SVM en vert

    plt.show()

visualize_svm()
```

Le résultat de ce code est donnée par la figure :

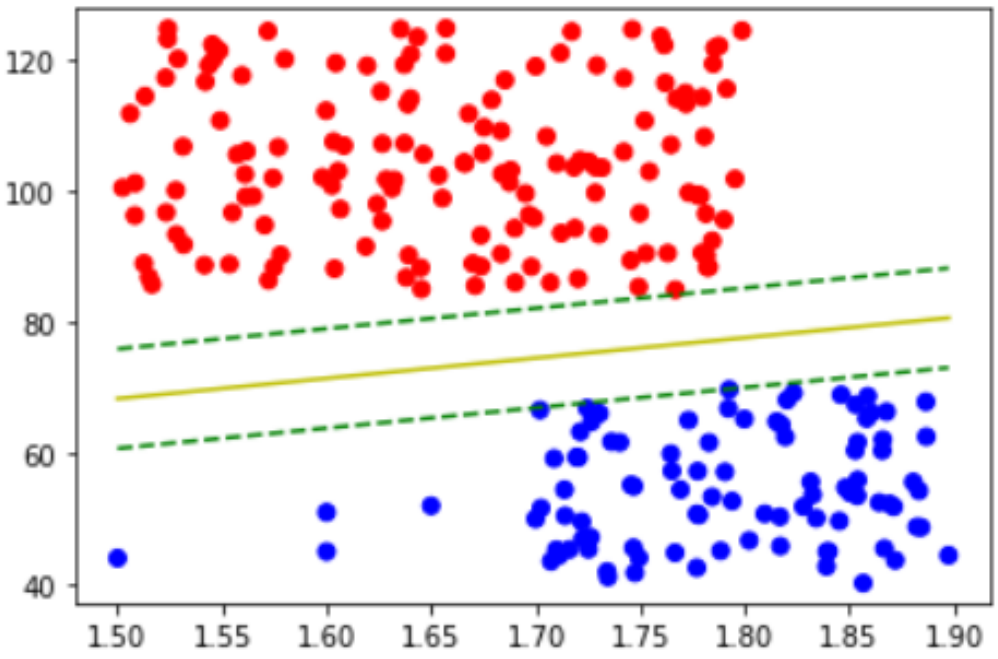

Fig. 2.2 – Résultat final

Remarque :

La présentation de la dataset peut changer un peu dans sa forme d'une exécution à une autre vu l'utilisation de la commande random lors de la génération de la dataset dans la partie (2.1)

2.4 Interprétations des résultats

On a réussi à mettre en oeuvre la méthode SVM dans notre exemple de dataset et le code obtenu est généralisable pour toutes dataset à éléments séparables.
On remarque qu'en jouant sur les paramètres lamda et Nitérations, on peut avoir des résultats différents et plus que Nitérations est grand et lamda est petite, on aura un classement plus performant mais qui coute plus en terme de temps et de la mémoire.
Donc, on peut contrôler ses paramètres selon les capacités Hardware ainsi que la nature du problème et la précision qu'il exige.

Après avoir créer notre classe SVM, on peut faire des prédictions sur des individus non existants dans la dataset afin de tester s'ils sont hypertendus ou pas.
Faisons pour cela ce petit code pour tester ces individus et On peut de même les placer en violet dans notre repère.

```
L=[(1.6,80),(1.7,80),(1.6,55),(1.9,80),(1.77,100),(1.76,82),(1.7,110)]
color=np.array(["purple" for i in L ])
for x in L:
  print(donnee.predict(x))
  plt.scatter(L[:, 0], L[:, 1],  marker='o', c=color)
    plt.show()
```

On obtient le résultat suivant :

```
1.0  1.0  -1.0  -1.0  1.0  1.0  1.0
```

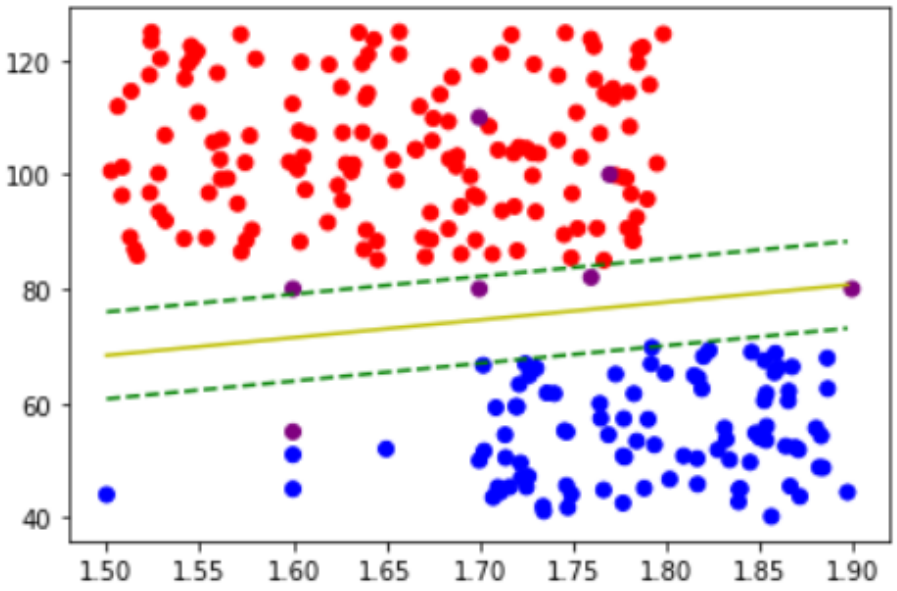

Fig. 2.3 – Ajout des points de prédiction

Ce qui montre que les individus 3 et 4 sont sains contrairement aux autres individus qui sont hypertendus.
On peut facilement vérifier par le calcul que ses résultats sont précis. D'ou notre modèle fournis des prédictions bien fiables et illustrées correctement dans la figure.

NB : L'individu 3 a été choisi en exprès sur la limite de la classification pour montrer la fiablité du modèle pour les cas limites.

Chapitre 3

Creuser d'avantage

3.1 Problème

Dans certain cas, les données sont non linéairement séparables et même l'approche a marge souple s'expose des limitations vis-à-vis ce problème.
La motivation derrière l'utilisation des fonction Kernel est la possibilité de projeter les valeur des donneés vers un autre espace d'une dimension superieure ou la séparation linéaire est possible, ce qui mêne a utilisé ces approches pour classifier. [5]

Par exemple, dans le scénario ci-dessous, nous ne pouvons pas avoir d'hyperplan linéaire entre les deux classes, alors comment le SVM classifie-t-il ces deux classes ?

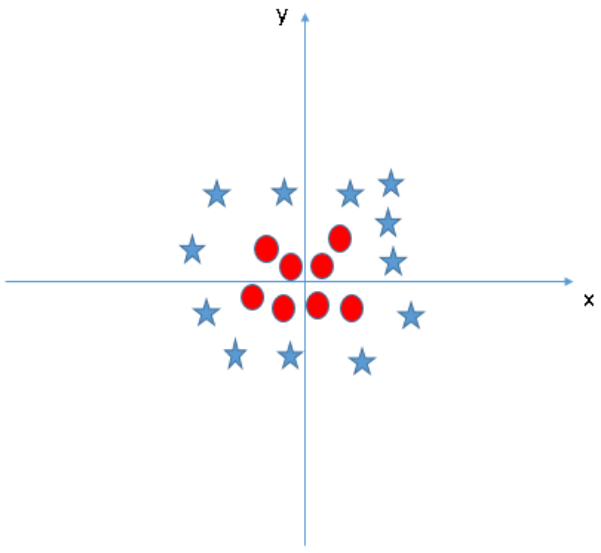

Fig. 3.1 – Exemple du problème de la non séparabilité linéaire

3.2 Idée des polynômes de Kernel

Il résout ce problème en introduisant une caractéristique supplémentaire. Ici, nous allons ajouter une nouvelle caractéristique

$$z = x^2 + y^2. \tag{3.1}$$

Maintenant, traçons les points de données sur les axes x et z :

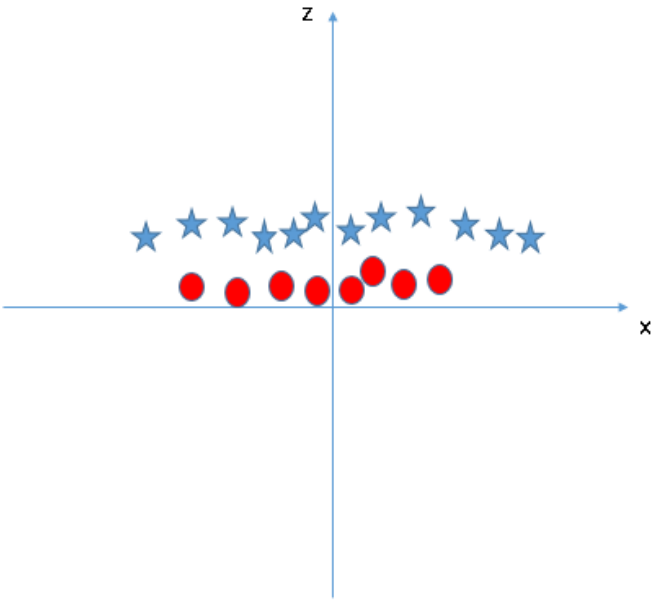

Fig. 3.2 – Transformation des données par un polynôme de Kernel

Dans le graphique ci-dessus, les points à considérer sont :
Toutes les valeurs de z sont toujours positives car z est la somme au carré de x et de y.

Dans le graphique original, les cercles rouges apparaissent près de l'origine des axes x et y, ce qui entraîne une valeur plus faible de z, et les étoiles relativement éloignées de l'origine entraînent une valeur plus élevée de z.

Dans le classifieur SVM, il est facile d'avoir un hyperplan linéaire entre ces deux classes. L'algorithme SVM dispose d'une technique appelée **"kernel trick"** permettant d'ajouter les caractéristiques pour avoir un hyperplan.

Le noyau SVM est une fonction qui prend un espace d'entrée de faible dimension et le transforme en un espace de dimension supérieure, c'est-à-dire qu'il convertit un problème non séparable en un problème séparable. Il est surtout utile dans les problèmes de séparation non linéaires. En d'autres termes, il effectue des transformations de données extrêmement complexes, puis trouve le processus permettant de séparer les données en fonction des étiquettes ou des sorties que vous avez définies.
Lorsque nous regardons l'hyperplan dans l'espace d'entrée original, il ressemble à un cercle :

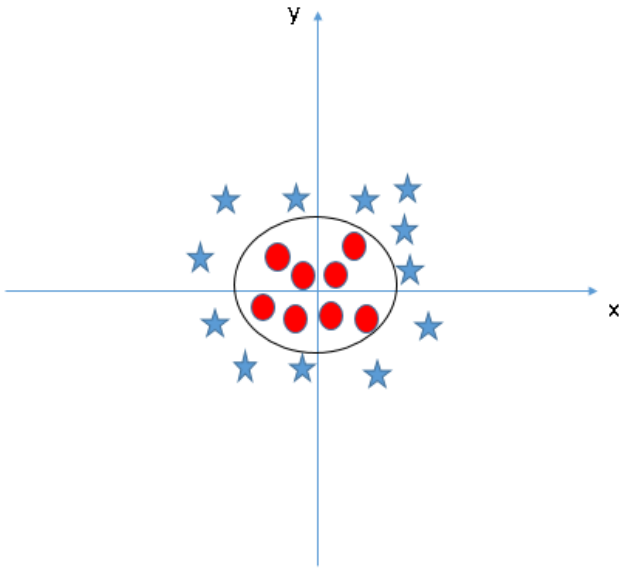

Fig. 3.3 – Classification par Kernel vue dans l'espace d'origine

En somme, dans le cas général, on visualise les propriétés de la dataset étudiée, et on décide selon sa séparabilité linéaire si on applique directement notre démarche de classification ou bien on effectue en premier ordre quelques transformations à l'aide des noyaux de Kernel pour avoir une classification linéaire possible puis on fait le retour vers la dimension de départ.

3.3 D'autres Classes ? ?

Dans son type le plus simple, le SVM ne supporte pas nativement la classification multi-classes. Il supporte la classification binaire et la séparation des points de données en deux classes. Pour la classification multi-classe, le même principe est utilisé après avoir décomposé le problème de multi-classification en plusieurs problèmes de classification binaire.[6]

L'idée est de faire correspondre les points de données à un espace de haute dimension pour obtenir une séparation linéaire mutuelle entre deux classes. C'est ce qu'on appelle l'approche One-to-One, qui décompose le problème de multiclassification en plusieurs problèmes de classification binaire. Un classificateur binaire pour chaque paire de classes.

Une autre approche que l'on peut utiliser est l'approche One-to-Rest. Dans cette approche, la décomposition est fixée à un classificateur binaire pour chaque classe.

Un seul SVM fait de la classification binaire et peut faire la différence entre deux classes. Ainsi, selon les deux approches de ventilation, pour classer les points de données de l'ensemble de données de m classes :

Prenons l'exemple d'un problème de classification de 3 classes : vert, rouge et bleu, comme dans l'image suivante :

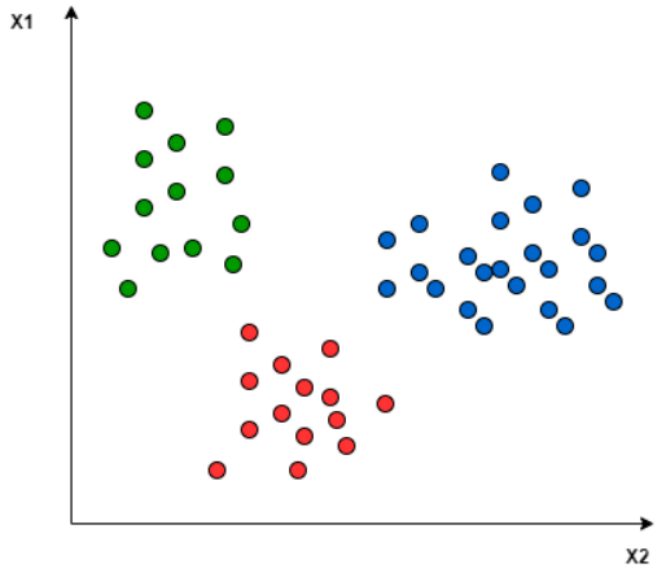

Fig. 3.4 – Exemple d'une classification Multi-classes

Dans l'approche One-to-One, nous avons besoin d'un hyperplan pour séparer entre chaque deux classes, en négligeant les points de la troisième classe. Cela signifie que la séparation ne prend en compte que les points des deux classes dans la séparation actuelle. Par exemple, la ligne rouge-bleu essaie de maximiser la séparation uniquement entre les points bleus et rouges. Elle ne tient pas compte des points verts :

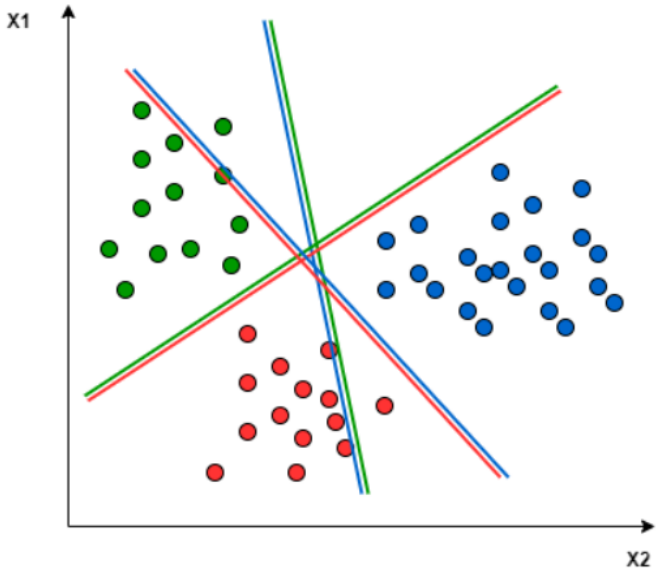

Fig. 3.5 – SVM multi-classes One-to-One

Dans l'approche One-to-Rest, nous avons besoin d'un hyperplan pour séparer une classe de toutes les autres à la fois. Cela signifie que la séparation prend en compte tous les points, en les divisant en deux groupes : un groupe pour les points de la classe et un groupe pour tous les autres points. Par exemple, la ligne verte essaie de maximiser la séparation entre les points verts et tous les autres points à la fois :

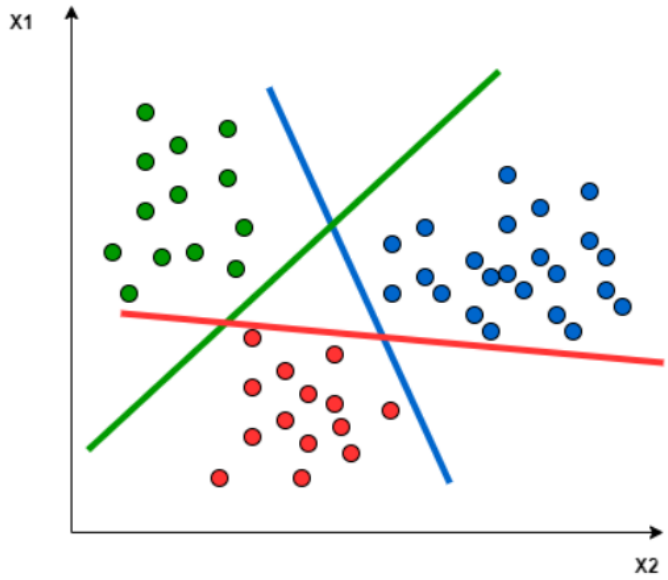

Fig. 3.6 – SVM multi-classes One-to-Rest

Conclusion

J'ai réussi dans ce travail à se familiariser avec les fondements mathématiques de la méthode SVM et de l'explorer dans le cadre d'une implémentation appliquée dans l'étude des données.

En effet, le travail de documentation que j'ai fait m'a permis de faire face à la problématique de classfication par les SVMs présentée dans le cadre du cours de l'optimisation et d'implémenter l'exemple des individus classés par le seuil de leurs IMC.

De plus, dans la dernière partie du travail j'ai allé encore plus loin dans l'exploration de la méthode dans des exemples plus délicats à l'aide des noyaux de Kernel et des multi-classes.

Pour conclure, les SVM sont très appréciées dans le cadre de dimension élevée. De plus, elles n'utilisent pas tous les échantillons d'apprentissage, mais seulement une partie (les vecteurs de support). En conséquence, elles demandent moins de mémoire.

Par contre, Si le nombre d'attributs est beaucoup plus grand que le nombre d'échantillons, les performances seront moins bonnes. et comme il s'agit de méthodes de discrimination entre les classes, elles ne fournissent pas directement des estimations de probabilités.

Souheib Ben Mabrouk.

Bibliographie

[1] I.M. Guyon B.E. Boser and V.N. Vapnik. *A training algorithm for optimal margin classifiers. In Proceedings of the fifth annual workshop on Computational learning theory.*

[2] Bharat Ratna Dr. B. R. Ambedkar University. Support vector machine-implementation and application in computer sciences-project report. https://www.docsity.com/en/support-vector-machine-implementation-and-application-in-computer-sciences-project-re84747//.

[3] Jamal Kharroubi. Support vector machine-implementation and application in computer sciences-project report. https://pastel.archives-ouvertes.fr/pastel-00001124/document.

[4] ZAIZ Faouzi. Les supports vecteurs machines (svm) pour la reconnaissance des caractères manuscrits arabes report. https://core.ac.uk/download/pdf/35402383.pdf.

[5] Sunil Ray. Understanding support vector machine(svm) algorithm from examples. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/7//.

[6] Multiclass classification using support vector machines. https://www.baeldung.com/cs/svm-multiclass-classification.