Atelier d’optimisation :

Support Vector Machine SVM

Réalisé par : Abdelkader MILADI

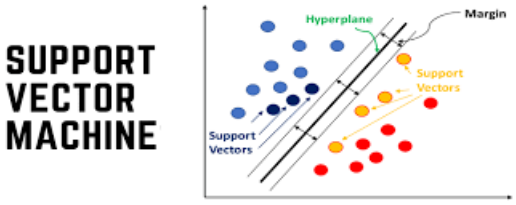
INDP 2 F

Objectifs :

- Comprendre la méthode SVM.
- Utiliser la méthode SVM pour résoudre un cas de discrimination linéaire.
- Implémenter la solution à travers un code en Python.

SVM :

SVM (Support Vector Machine ou Machine à vecteurs de support) : Les SVMs sont une famille d'algorithmes d'apprentissage automatique qui permettent de résoudre des problèmes tant de classification que de régression ou de détection d'anomalie. Ils sont connus pour leurs solides garanties théoriques, leur grande flexibilité ainsi que leur simplicité d'utilisation même sans grande connaissance de data mining.



La résolution de ces deux problèmes passe par la construction d'une fonction qui à un vecteur d'entrée **x** associe une sortie **y**.
Les observations sur le bord qui nous aident à dessiner la marge sont appelées « vecteurs de support ».
Il est possible de pratiquer l'apprentissage automatique avec le langage Python, en utilisant la bibliothèque « Scikit-Learn » qui propose un module « sklearn.svm » ou bien en écrivant un code qui estime à partir de zéro les SVM.

Cas linéaire et hyperplan séparateur :

Le SVM linéaire est utilisé pour les données linéairement séparables : si un ensemble de données peut être classé en deux classes à l'aide d'une seule ligne droite, ces données sont appelées « données linéairement séparables ».
Le cas d'une fonction discriminante linéaire, obtenue par combinaison linéaire du vecteur d'entrée $x = (x1, x2)^T \in \mathbb{R}2\backslash\{0\}$ avec un vecteur de poids $\omega = (\omega1, \omega2)^T \in \mathbb{R}2\backslash\{0\}$ et $\omega3 \in \mathbb{R}$: $g(x) = \omega^T x - \omega3$
La frontière de décision d'équation $g(x) = 0$ est un « hyperplan séparateur ».

Programmation :

```
#Importation des bibliothèques nécessaires
import numpy as np
import matplotlib.pyplot as plt
import random
```

```
#Fonction qui calcule les contraintes gi(ω)
def contrainte(x, y, w, i):
    g = - y[i] * (np.dot(w[:2], x[i]) - w[-1]) + 1
    return g
```

```
#Fonction qui calcule les paramètres ω et λ
def parameters(x, y):
    lamda = np.array([0.1] * x.shape[0])
    pas = 0.0005
    eps = 1e-5
    i = 0
    while(i < 70000):
        w = np.zeros(3)
        gather = lamda.copy()
        Margin = []
        L = []
        #Détermination de ω1 et ω2
        w[:2] = np.dot(y * lamda, x)
        #Détermination des points sur les vecteurs supports tels que λ > 0
        for l in range(x.shape[0]):
            if gather[l] > 0:
                Margin.append(x[l])
        #Détermination de ω3
        if Margin != []:
            for m in Margin:
                w[-1] += np.dot(m, w[:2])
            w[-1] /= len(Margin)
        #Calcul des contraintes gi(ω) pour chaque individu
        for j in range(len(x)):
            L.append(contrainte(x, y, w, j))
        g = np.array(L)
        #Calcul de λ selon la méthode d'Uzawa
        lamda += pas * g
        for j in range(lamda.shape[0]):
            if lamda[j] < 0:
                lamda[j] = 0

        if np.linalg.norm(lamda - gather, 2) < eps:
           return w
        i += 1
    return w
```

```python
#Fonction qui représente graphiquement la solution
def show(x, w):
    w = w.tolist()
    #X est l'axe des abscisses et y est l'axe des ordonnées
    X = x[:, 0]
    y = x[:, 1]
    #Détermination des 3 droites à dessiner : D0, D1 et D-1
    d0 = (- w[0] * X + w[2]) / w[1]
    d1 = (- w[0] * X + w[2] + 1) / w[1]
    d_1 = (- w[0] * X + w[2] - 1) / w[1]
    plt.title("Support Vector Machine")
    plt.xlabel('Taille(m)')
    plt.ylabel('Poids(Kg)')
    plt.scatter(X, y, marker="o", label = 'Individu', color = 'saddlebrown')
    plt.plot(X, d0, label='D0', linestyle='dashed', color = 'red')
    plt.plot(X, d1, label='D1', color = 'darkblue')
    plt.plot(X, d_1, label='D-1', color = 'yellow')
    plt.legend()
    plt.grid()
    plt.show()
```

```python
#Génération des données de test
individu = np.array([[1.5, 72], [1.6, 77], [1.65, 80], [1.75, 92], [1.6, 60], [1.83, 69], [1.7, 65], [1.8, 72]])
classification = np.array([1, 1, 1, 1, -1, -1, -1, -1])
for i in range(12):
    data = [random.uniform(1.5, 2.0), random.uniform(50, 100)]
    individu = np.concatenate((individu, np.array([data])))
    imc = data[1] / data[0] ** 2
    if imc < 24.9:
        classification = np.concatenate((classification, np.array([-1])))
    else:
        classification = np.concatenate((classification, np.array([1])))
```

```python
#Test de la fonction qui calcule les paramètres ω et λ
W = parameters(individu, classification)
```

```python
#Test de la fonction qui génère la représentation graphique
show(individu, W)
```