TP 2 SVM

Reprenons l'exemple du cours : $\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 2 & 0 \\ 3 & 0 \end{bmatrix}$ et $y = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$

I. Exercice 1 – Plan séparateur

Utiliser la fonction aff_donnees($\mathbf{X}, y, bornex, borney, s$) qui affiche les données sur une figure en mettant un symbole différent pour les exemples positifs et négatifs. $bornex$, $borney$ représentent les limites des axes que l'on pourra fixer à min-1 et max+1. s représente la taille des symboles (utile dans la dernière partie du TP) que l'on pourra fixer à 50.

```
def aff_donnees(X,y,bornex,borney,s):
    plt.scatter(X[:, 0], X[:, 1], c=y, s=s, cmap='winter');
    plt.xlim(bornex);
plt.ylim(borney);
```

Ecrire une fonction affichePlan($\mathbf{w}, b, bornex$) qui affiche un hyperplan dans le plan 2D en passant en argument les paramètres de l'hyperplan $\boldsymbol{w}$ et b ainsi que $bornex$. On pourra pour cela faire varier x dans l'intervalle défini par $bornex$ et calculer les y correspondants. Afficher l'hyperplan de paramètre $\boldsymbol{w}^T = [1{,}0.1]$ et $b = -1$.

Questions
• S'agit-il d'un hyperplan séparateur ?

II. Exercice 2 – SVM linéaire dans le primal

On a vu en cours que le problème des SVM peut être mis sous la forme d'un problème quadratique dont la solution est obtenue avec la librairie cvxopt. Ecrire une fonction ResoudPrimal($\mathbf{X}$,y) qui renvoie les paramètres $\boldsymbol{w}$ et b de l'hyperplan séparateur.

Rq : la fonction sol = cvxopt.solvers.qp(P, q, G, h) résout le problème :

$$\min_z 0.5 z^T P z + q^T z \qquad sc \quad Gz \leq h$$

P, q, G, h doivent être des matrices de décimaux de type cvxopt.matrix. La conversion est réalisée avec P= cvxopt.matrix(P). La solution au problème est trouvée avec z=sol['x'].
Compléter pour cela le code suivant :

```
def Resoud_primal(X,y):
N= ???
n= ???
q = ???
P1=np.concatenate((np.zeros((1,1)),np.zeros((1,n))),axis=1)
P2=np.concatenate((np.zeros((n,1)),np.eye(n)),axis=1)
P=np.concatenate((P1,P2),axis=0)
P=cvxopt.matrix(P)
for i in range(N):
    g=np.concatenate((np.reshape(-y[i],(1,1)), np.reshape(-y[i]*X[i][:],(1,2))),axis=1)
    if i==0:
        G=g
    else:
        G=np.concatenate((G, g), axis=0)
G=cvxopt.matrix(G+0.)
```

Pour concaténer des matrices A et B horizontalement, on fera :
np.concatenate((A,B),axis=1)
pour les concatener verticalement,
np.concatenate((A,B),axis=0)
Pour remplir une matrice de taille *dy*x*dx* avec des uns ou des zéros, on fera
np.zeros((dy,dx)) ou np.ones((dy,dx))
Une matrice identité de taille *dx*x*dx* sera obtenue avec
np.eye(dx)
Vérifier votre programme en traçant cet hyperplan.

Questions
- Est-ce que l'hyperplan obtenu vous parait correct ?
- Que se passe-t-il si on ajoute le point $\mathbf{x} = \begin{bmatrix} 2.1 \\ 2.5 \end{bmatrix}$ d'étiquette 1 ? Vérifier en faisant tourner le programme.
- Même question avec le point $\mathbf{x} = \begin{bmatrix} 1.5 \\ 2.5 \end{bmatrix}$

III. Exercice 3 – SVM à marge souple dans le primal

Programmer un SVM à marge souple sur les données de départ, ResoudPrimal(X,y,C).

Questions
- En ajoutant le point $\begin{bmatrix} 2.1 \\ 2.5 \end{bmatrix}$ d'étiquette 1, que se passe-t-il en testant valeurs de C ?
- Même question avec le point $\begin{bmatrix} 2.1 \\ 2.5 \end{bmatrix}$.
- Même question avec le point $\begin{bmatrix} 1.5 \\ 2.5 \end{bmatrix}$.
- Conclusion sur l'effet du paramètre C

IV. Exercice 4 – SVM dans le dual

Pour étudier les SVM dans le dual, utiliser la librairie scikit-learn (from sklearn import svm). On pourra alors directement apprendre un SVM avec :

```
model = svm.SVC(kernel='linear', C=1)

model.fit(X, y)
```

Apprendre un SVM sur les données initiales **x**,y.

Récupérer les valeurs des $\alpha_i y_i$ (model.dual_coef_), les vecteurs support (model.support_vectors_) et les indices des vecteurs support (model.support_) du SVM appris et retrouver les valeurs de l'hyperplan optimal $\boldsymbol{w}$ et b.

Questions
- Comment retrouver l'hyperplan optimal en fonction des $\alpha_i y_i$?
- Retrouver les résultats de l'exercice précédent.

V. Exercice 5 – SVM avec Kernel

Pour afficher la fonction de décision du SVM appris en 4.1, utiliser la fonction

```
def aff_frontiere(X,y,bornex,borney,model):
```

```
    aff_donnees(X,y,bornex,borney,50)
    xx, yy = np.meshgrid(np.linspace(bornex[0], bornex[1],50), np.linspace(borney[0], borney[1],50))
    xy =  np.concatenate((np.reshape(xx,(xx.shape[0]*xx.shape[1],1)), np.reshape(yy,(yy.shape[0]*yy.shape[1],1))),axis=1)

    P = model.predict(xy)

    aff_donnees(xy,P,bornex,borney,1)
```

Vérifier que l'on retrouve bien les hyperplans de l'exercice 4.

Apprendre des SVM avec des noyaux linéaires ou rbf. Que remarquez-vous sur les nouvelles fonctions de décisions ?

Ajouter le point $\begin{bmatrix} 2.1 \\ 2.5 \end{bmatrix}$ et tester plusieurs noyaux. Conclusion ?

Même question avec le point $\begin{bmatrix} 1.5 \\ 2.5 \end{bmatrix}$.

Utiliser maintenant les données stockées dans data.npz et tester les différents noyaux (gaussien, polynomial et linéaire). Lequel amène aux meilleurs résultats ?

```
f = np.load(data.npz')
X=f['arr_0']
y=f['arr_1']
```

Questions

- Expliquer le fonctionnement de la fonction aff_frontiere
- Peut-on apprendre des frontières non linéaires avec un noyau linéaire ?
- Que donne un SVM avec noyau rbf sur les données linéairement séparable ? Est-il bien adapté ?