

TD n° 5 : SVM avec scikit-learn

L'objectif de cette séance de travaux pratiques est de montrer l'utilisation des SVM pour les problèmes de classification et de régression en python (`sklearn`).

Exercice 1 : (Prise en main des SVM en Scikit-learn)

1. Chargement des données. Comme les données à traiter sont dans un fichier tabulé, nous allons utiliser la bibliothèque `pandas` :

```
import pandas
df = pandas.read_csv("TD5-data_svm.txt",sep="\t",header=0,index_col=0)
print(df.shape)
# création des deux sous-data.frame des deux classes : y=+1 (p) et y=-1 (n)
dfpos = df[df['y']=='p']
dfneg = df[df['y']=='n']
```

2. Visualisation. Nous allons représenter sur le plan, chacun des points représentés par leurs numéros dans la structure de données :

```
import matplotlib.pyplot as plt
# affichage des instances
def myscatter(df,dfpositif,dfnegatif):
    # nuage de points « blanc » pour définir les dimensions du graphique
    plt.scatter(df.iloc[:,0],df.iloc[:,1],color="white")
    # affichage des instances positives
    for i in dfpositif.index:
        plt.annotate(i,xy=(df.loc[i,'x1'],df.loc[i,'x2']),xytext=(-3,-3),
                    textcoords='offset points',color='red')
    #affichage des instances negatives
    for i in dfnegatif.index:
        plt.annotate(i,xy=(df.loc[i,'x1'],df.loc[i,'x2']),xytext=(-3,-3),
                    textcoords='offset points',color='blue')
    return None

# visualisation
myscatter(df,dfpos,dfneg)
plt.show()
```

3. Apprentissage du modèle SVM :

```
#importation de la classe SVC
from sklearn.svm import SVC
svm = SVC(kernel='linear')
svm.fit(df.values[:,0:2],df.values[:,2])
```

4. On peut connaître le nombre de points supports (`svm.support_.shape`), leurs numéros (`df.index[svm.support_]`) et leurs poids (`svm.dual_coef_`), les abscisses et ordonnées des vecteurs supports (`svm.support_vectors_`). pour mettre en évidence des points supports dans la représentation des points, on utilisera :

```
myscatter(df,dfpos,dfneg)
plt.scatter(abs,ord,marker="s", s=200,facecolors='none',edgecolors='black')
# marker: forme (o pour rond, s pour square ...)
# s : taille
# facecolors: couleur de l'intérieur
# edgecolors: couleur du contour
plt.show()
```

où `abs` et `ord` sont les listes des abscisses et ordonnées des points à mettre en évidence.

5. Visualisation de la frontière entre les classes, et de la marge. La frontière de partage est donnée par l'équation $\beta_1 x + \beta_2 y + \theta = 0$ où (β_1, β_2) est donné par `svm.coef_` et θ est donné par `svm.intercept_`.

On peut donc tracer 3 droites chacune à l'aide de 2 points d'abscisses : `xx = np.array([1,13])`

— les ordonnées de la frontière sont données par :

```
yf = -svm.coef_[0][0]/svm.coef_[0][1]*xx-svm.intercept_/svm.coef_[0][1]
```

— les ordonnées de la marge du côté bas sont données par :

$$y_b = -\text{svm.coef_}[0][0]/\text{svm.coef_}[0][1]*xx - (\text{svm.intercept_}-1.0)/\text{svm.coef_}[0][1]$$

— les ordonnées de la marge du côté haut sont données par :

$$y_h = -\text{svm.coef_}[0][0]/\text{svm.coef_}[0][1]*xx - (\text{svm.intercept_}+1.0)/\text{svm.coef_}[0][1]$$

Pour la visualisation, on écrira

```
myscatter(df,dfpos,dfneg)
plt.scatter(abs,ord,marker='s',s=200,facecolors='none',edgecolors='black')
plt.plot(xx,yf,c='green')
plt.plot(xx,yb,c='orange')
plt.plot(xx,yh,c='orange')
plt.show()
```

Exercice 2: (Utilisation de noyaux sur les données Iris)

Nous allons réutiliser le jeu de données iris.

1. Comme précédemment, nous allons importer les données :

```
from sklearn import datasets
iris = datasets.load_iris()
y = iris.target # Les labels associés à chaque enregistrement
```

2. se restreindre à 2 caractéristiques, par exemple les 2 premières : $X = \text{iris.data[:, :2]}$
3. calculer les min et max des deux caractéristiques choisis
4. construire les classifieurs suivants :

```
clf = svm.LinearSVC(C=C,max_iter=20000).fit(X, y)
clf = svm.SVC(kernel='linear', C=C).fit(X, y)
clf = svm.SVC(kernel='rbf', gamma=gamma, max_iter=10000000, C=C).fit(X, y)
clf = svm.SVC(kernel='poly', degree=3, max_iter=10000000, gamma=gamma,C=C).fit(X, y)
```

5. construire les prédictions pour un maillage du plan des caractéristiques choisies :

```
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,y_max, h))
# ou h est le pas du maillage...
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
# ravel permet d'"aplatir" le tableau obtenu
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
# Afficher aussi les points d'apprentissage
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
```

6. La fonction noyau est l'une des suivantes :

linéaire : $\langle x, x' \rangle$ polynomial : $(\gamma \langle x, x' \rangle + r)^d$ rbf : $\exp(-\gamma \|x - x'\|^2)$.

Deux paramètres jouent un rôle important : C et γ . le paramètre C promeut des fonctions lisses pour de petites valeurs, tout comme le paramètre γ qui définit le voisinage des points.

Intuitivement, le paramètre γ définit la portée de l'influence d'un seul exemple d'entraînement, les valeurs basses signifiant "loin" et les valeurs élevées signifiant "proche".

Le paramètre C établit un compromis entre la classification correcte des exemples d'entraînement et la maximisation de la marge de la fonction de décision. Pour des valeurs grandes de C , une marge plus petite sera acceptée si la fonction de décision permet de classer correctement tous les points d'entraînement. Un C plus faible encouragera une marge plus grande, donc une fonction de décision plus simple, au détriment de la précision de l'entraînement. En d'autres termes, C se comporte comme un paramètre de régularisation dans le SVM.

Expérimentez l'influence de ces paramètres sur la forme des frontières.

On pourra aller voir la page <https://remi.flamary.com/demos/svmreg.fr.html> qui montre sur un exemple, l'influence de ces 2 paramètres.