TP 5

SVM linéaires

Les machines à vecteurs de support (SVM : *Support Vector Machines*) sont une classe de méthodes d'apprentissage statistique basées sur le principe de la maximisation de la marge (séparation des classes). Il existe plusieurs formulations (linéaires, versions à noyaux) qui peuvent s'appliquer sur des données séparables (linéairement) mais aussi sur des données non séparables.
Les avantages des SVM :

- Très efficaces en dimension élevée.
- Ils sont aussi efficaces dans le cas où la dimension de l'espace est plus grande que le nombre d'échantillons d'apprentissage.
- Pour la décision, n'utilisent pas tous les échantillons d'apprentissage, mais seulement une partie (les vecteurs de support). En conséquence, ces algorithmes demandent moins de mémoire.

Désavantages :

- Si le nombre d'attributs est beaucoup plus grand que le nombre d'échantillons, les performances sont moins bonnes.
- Comme il s'agit de méthodes de discrimination entre les classes, elles ne fournissent pas d'estimations de probabilités.

Jeu de données Iris

Dans Scikit-learn, les SVM sont implémentées dans le module `sklearn.svm`. Dans cette partie nous allons nous intéresser à la version linéaire (Scikit-learn utilise les bibliothèques libLinear et libSVM).

Nous allons utiliser le jeu de données Iris déjà rencontré dans les séances précédentes. Pour pouvoir afficher les résultats, on va utiliser seulement les premiers deux attributs (longueur et largeur des sépales).

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split

# Chargement des données
iris = datasets.load_iris()
```

Pour commencer, nous ne conservons que les deux premiers attributs du jeu de données :

```
X, y = iris.data[:, :2], iris.target
# On conserve 50% du jeu de données pour l'évaluation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
```

Nous pouvons maintenant entraîner une machine à vecteur de support linéaire :

```
C = 1.0 # paramètre de régularisation
lin_svc = svm.LinearSVC(C=C)
lin_svc.fit(X_train, y_train)
```

Question

Calculez le score d'échantillons bien classifiés sur le jeu de données de test.

Visualisons maintenant la surface de décision apprise par notre modèle :

```
# Créer la surface de décision discretisée
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
# Pour afficher la surface de décision on va discrétiser l'espace avec un
pas h
h = max((x_max - x_min) / 100, (y_max - y_min) / 100)
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Surface de décision
Z = lin_svc.predict(np.c_[xx.ravel(), yy.ravel()])
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_train[:, 0], X_train[:, 1], label="train", edgecolors='k',
c=y_train, cmap=plt.cm.coolwarm)
plt.scatter(X_test[:, 0], X_test[:, 1], label="test", marker='*', c=y_test,
cmap=plt.cm.coolwarm)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title("LinearSVC")
```

Question :

Testez différentes valeurs pour le paramètre C. Comment la frontière de décision évolue en fonction de C ?

Question

D'après la visualisation ci-dessus, ce modèle vous paraît-il adapté au problème ? Si non, que peut-on faire pour l'améliorer ?

Nous verrons dans le prochain TP que scikit-learn permet de manipuler des machines à vecteurs de support avec des noyaux non-linéaires dans la classe `SVC`.

Les modèles linéaires `LinearSVC()` et `SVC(kernel='linear')`, comme nous l'avons déjà dit, produisent des résultats légèrement différents à cause du fait qu'ils optimisent des fonctions de coût différentes mais aussi à cause du fait qu'ils gèrent les problèmes multi-classe de manière différente (linearSVC utilise *One-vs-All* et SVC utilise *One-vs-One*).

```
lin_svc = svm.LinearSVC(C=C).fit(X_train, y_train)
svc = svm.SVC(kernel='linear', C=C).fit(X_train, y_train)

titles = ['SVC with linear kernel', 'LinearSVC (linear kernel)']

fig = plt.figure(figsize=(12, 4.5))

for i, clf in enumerate((svc, lin_svc)):
    plt.subplot(1, 2, i + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    # Utiliser une palette de couleurs
    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_train[:, 0], X_train[:, 1], label="train", edgecolors='k',
c=y_train, cmap=plt.cm.coolwarm)
    plt.scatter(X_test[:,  0],  X_test[:,  1],  label="test",  marker='*',
c=y_test, cmap=plt.cm.coolwarm)
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.title(titles[i])
plt.show()
```

Pour l'instant, nous n'avons exploité que deux variables explicatives. Néanmoins, l'intérêt des machines à vecteur de support linéaires est qu'il est souvent plus facile de trouver des hyperplans séparateurs dans des espaces de grande dimension.

Question

Réalisez l'optimisation d'une nouvelle machine à vecteur de support linéaire mais en utilisant les quatre attributs du jeu de données Iris. Le score de classification en test a-t-il augmenté ? Pourquoi ?

Jeu de données Digits

Le jeu de données Digits est une collection d'images de chiffres manuscrits (nous l'avons déjà utilisé dans le TP sur les forêts aléatoires). Elles peuvent se charger directement depuis scikit-learn :

```
from sklearn.datasets import load_digits
digits = load_digits()
X, y = digits.data, digits.target
```

Question :

Utilisez les données Digits pour construire un classifieur LinearSVC et évaluez-le. Si le temps d'apprentissage est trop long, sélectionnez une partie plus petite de la base d'apprentissage (par exemple 10000 échantillons). Pour quelle valeur de C on obtient le meilleurs résultats de généralisation ?