TP 7
Algorithmes à noyaux

One Class SVM (OCSVM)

Les OCSVM sont des estimateurs de support de densité pour des données multidimensionnelles. L'idée derrière l'implémentation est de trouver l'hyperplan le plus éloigné de l'origine qui sépare les données de l'origine.

Dans scikit-learn, les *One-Class* SVM sont implémentés dans le module `sklearn.svm`, n'hésitez pas à consulter la documentation de ce module pour plus de détails. Nous utilisons un noyau gaussien pour faire la détection des *outliers* dans un échantillon de données en deux dimensions. Pour cet exemple, nous générons deux clusters gaussiens auxquels nous ajoutons 10% de données anormales, tirées au hasard uniformément dans l'espace à deux dimensions considéré :

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

from sklearn import svm

# On créé deux groupes séparés (échantillons de gaussiennes)
N = 200
data1 = 0.3 * np.random.randn(N // 2, 2) + [2,2]
data2 = 0.3 * np.random.randn(N // 2, 2) - [2,2]

# On créé 10% de données anormales (*outliers*)
outliers = np.random.uniform(size=(N // 10, 2), low=-6, high=6)

# Les données = groupes + anomalies
X = np.concatenate((data1, data2, outliers))

plt.scatter(X[:,0], X[:,1]) and plt.show()
```

Nous pouvons ensuite créer le modèle de *one-class* SVM avec `sklearn`. Le paramètre `nu` correspond à la proportion maximale d'erreurs autorisées, c'est-à-dire au pourcentage maximal de points du jeu de données que l'on acceptera d'exclure de notre classe. Cette fraction doit peu ou prou correspondre au pourcentage de données anormales attendu dans le jeu de données. Dans notre cas, nous savons qu'il y a environ 10% d'"*outliers* donc nous pouvons choisir `nu=0.1`.

```
# Construction du modèle (noyau RBF)
clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.05)
clf.fit(X)
```

Le code suivant permet alors de tracer les frontières (plus exactement les lignes de niveaux) de la fonction de décision de la `OneClassSVM` ainsi entraînée :

```
# Afficher les points et les vecteurs les plus proches du plan de séparation
xx, yy = np.meshgrid(np.linspace(-7, 7, 500), np.linspace(-7, 7, 500))
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
y_pred = clf.predict(X)

# Choix du jeu de couleurs
plt.set_cmap(plt.cm.Paired)
# Trace le contour de la fonction de décision
plt.contourf(xx, yy, Z)
# Affiche les points considérés comme "inliers"
plt.scatter(X[y_pred>0,0],    X[y_pred>0,1],    c='white',    edgecolors='k',
label='inliers')
# Affiche les points considérés comme "outliers"
plt.scatter(X[y_pred<=0,0], X[y_pred<=0,1], c='black', label='outliers')
plt.legend()
plt.show()
```

Question

Testez plusieurs valeurs pour le paramètre `gamma`. Pour quelle valeur le résultat semble meilleur (moins de *outliers* incorrectement classés) ? En pratique on ne connait pas les *outliers*, l'utilité des OCSVM est de les détecter. Le paramètre `nu` doit aussi avoir une bonne valeur pour ne pas sous-estimer (ou sur-estimer) le support de la distribution.

Question

Remplacez le noyau `rbf` par un noyau linéaire. Quel problème constatez-vous ?

SVM pour la régression

Dans le cas de la régression, l'objectif est d'apprendre un modèle qui prédit les valeurs d'une fonction à partir des valeurs des variables d'entrée. L'idée est de trouver la fonction la plus « lisse » qui passe par les (ou à proximité des) données d'apprentissage. Scikit-learn implémente le modèle SVR (epsilon-regression) dans le module Python `sklearn.svm.SVR` dont vous pouvez bien sûr consulter la documentation.
Dans cette partie nous présenterons la régression dans le cas unidimensionnel en comparant plusieurs noyaux avec scikit-learn. Le module `sklearn.svm.SVR` permet de faire varier tous les paramètres.
Il faut d'abord importer les modules :

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

from sklearn.svm import SVR
```

Données synthétiques

Dans un premier temps, nous allons travailler sur des données générées. Notre objectif sera de reproduire une sinusoïde, comme dans le TP sur les arbres de décision :

```
X = np.sort(5 * np.random.rand(40, 1), axis=0)
y = np.sin(X).ravel()
```

On ajoute un bruit aléatoire sur 20% des observations. Nos échantillons d'entraînement correspondent donc à la courbe suivante :

```
y[::5] += 3 * (0.5 - np.random.rand(8))
plt.show(X, y)
```

Nous pouvons facilement entraîner trois modèles de SVM pour la régression grâce à la classe `SVR` de scikit-learn :

```
# Création des SVM
C = 1e3
svr_rbf = SVR(kernel='rbf', C=C, gamma=0.1)
svr_lin = SVR(kernel='linear', C=C)
svr_poly = SVR(kernel='poly', C=C, degree=2)

# Entraînement des SVM sur les observations bruitées
y_rbf = svr_rbf.fit(X, y).predict(X)
y_lin = svr_lin.fit(X, y).predict(X)
y_poly = svr_poly.fit(X, y).predict(X)
```

Afficher les résultats :

```
plt.scatter(X, y, color='darkorange', label='Données')
plt.plot(X, y_rbf, color='navy', lw=2, label='RBF')
plt.plot(X, y_lin, color='c', lw=2, label='Linéaire')
plt.plot(X, y_poly, color='cornflowerblue', lw=2, label='Polynomial')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Support Vector Regression')
plt.legend()
plt.show()
```

Question :

Pourquoi employer une valeur aussi grande pour le parametre C (ici, $C = 1000$) ?

Diabetes dataset

`sklearn` contient plusieurs jeux de données réels d'exemple. Concentrons-nous sur le jeu de données *Diabetes* consistant à prévoir la progression de la maladie (représentée par un indice quantitatif) à partir de différentes variables : âge, sexe, pression artérielle, IMC et six valeurs de prélèvements sanguins.

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

diabetes = load_diabetes()
X_train, X_test, y_train, y_test = train_test_split(diabetes.data,
diabetes.target, test_size=0.30, random_state=0)
```

Question

Chargez la base de données Diabetes du module `sklearn.datasets` et faites une partition aléatoire en partie apprentissage et partie test (70% apprentissage, 30% test). Construisez un modèle de SVM de régression sur cette base et calculez l'erreur quadratique moyenne sur l'ensemble de test. Utilisez `GridSearchCV` pour déterminer le meilleur noyau à utiliser.