Travaux pratiques de
Machine learning
Cycle pluridisciplinaire d'études supérieures
Université Paris sciences et lettres

Joon Kwon

vendredi 28 avril 2023

Ce TP doit être rédigé et sera évalué. Il doit être rendu sous la forme d'un notebook Jupyter. Il est possible de travailler en binôme. Les noms doivent apparaître dans le document ainsi que dans le nom du fichier de la façon suivante : `TP-NOMDEFAMILLE1_NOMDEFAMILLE2.ipynb`. Il doit être envoyé le mercredi 17 mai 2023 au plus tard à l'adresse `joon.kwon@inrae.fr`.

On travaille à nouveau à avec le jeu de données sur le vin.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data =
 ↪  pd.read_csv('https://joon-kwon.github.io/cpes-ml/wine_dataset.csv')
data.head()
```

Nous ne gardons cette fois-ci que deux variables explicatives.

```
X = data[['chlorides','total_sulfur_dioxide']]
y = data['style']
```

Par ailleurs, nous allons appliquer à chaque variable explicative une transformation affine de sorte que ses valeurs appartiennent à $[-1, 1]$ (dans la pratique, le temps de calcul des SVM, ainsi que la performance des prédicteurs obtenus, ont tendance à être meilleurs lorsque les variables explicatives sont normalisées).

```
from sklearn.preprocessing import MinMaxScaler
X = MinMaxScaler(feature_range=(-1,1)).fit(X).transform(X)
```

QUESTION 1. — Séparer le jeu de données en un échantillon d'apprentissage et un échantillon de test.

Nous allons d'abord effectuer une régression logistique.

```
from sklearn.linear_model import LogisticRegressionCV
```

La classe `LogisticRegressionCV` (`CV` pour validation croisée) choisit automatiquement le meilleur hyperparamètre `C` par validation croisée et renvoie le prédicteur construit avec la valeur sélectionnée et entraîné sur tout l'échantillon d'apprentissage. Les arguments à fournir sont `Cs` : une liste ou un `array` d'hyperparamètres à considérer, et `cv` : le paramètre k de la k-validation croisée. On pourra si besoin consulter la documentation.

QUESTION 2. — Utiliser `LogisticRegressionCV` avec une grille assez large d'hyperparamètres (par exemple `np.logspace(-6,6,20)`) pour construire un prédicteur (nommé `logreg`) et afficher son score sur l'échantillon de test.

Puisque nous n'avons que deux variables explicatives, nous pouvons représenter dans le plan les entrées, ainsi que la frontière de décision du prédicteur `logreg`.

```
plt.scatter(X[:,0],X[:,1],c=y,edgecolors='k')
figure = plt.gca()
xlim = figure.get_xlim()
ylim = figure.get_ylim()
xx = np.linspace(xlim[0], xlim[1], 100)
yy = np.linspace(ylim[0], ylim[1], 100)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
P = logreg.decision_function(xy).reshape(XX.shape)
plt.contour(XX, YY, P, levels=[0], alpha=1)
plt.show()
```

Question 3. — En s'inspirant du code ci-dessus, définir une fonction `plot_decision_boundary` qui prend un argument un prédicteur et qui produit une figure représentant les entrées ainsi que la frontière de décision du prédicteur.

Nous entraînons à présent un SVM à l'aide de la classe `LinearSVM`. Contrairement à la régression logistique, il n'existe pas de version de `LinearSVM` qui choisisse automatiquement par validation croisée l'hyperparamètre de régularisation. Mais la fonction `GridSearchCV` permet de parvenir facilement au même résultat :

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
grid = {'C':np.logspace(-6,6,20)}
svc =
 ↪  GridSearchCV(LinearSVC(),cv=5,param_grid=grid,verbose=3)
print('Score
 ↪  :',svc.fit(X_train,y_train).score(X_test,y_test))
plot_decision_boundary(svc)
```

où `cv` spécifie le paramètre k de la k-validation croisée, où `param_grid` est un *dictionnaire* qui spécifie les valeurs d'arguments à considérer (en l'occurrence, on fait seulement varier la valeur de `C`, mais on peut spécifier des listes de valeurs pour plusieurs arguments), et où `verbose=3` permet d'avoir des messages qui indiquent où en sont les calculs (intéressant et rassurant lorsque les calculs prennent un certain temps). On peut ensuite consulter la meilleure combinaison de paramètres retenue via `svc.best_params_`.

```
print(svc.best_params_)
```

Nous souhaitons à présent faire appel à des SVM avec des noyaux polynômiaux de la forme :

$$\mathrm{K}(x, x') = (\gamma \langle x, x' \rangle + r)^{\delta}, \qquad x, x' \in \mathbb{R}^d.$$

Par exemple :

```
from sklearn.svm import SVC
svcpoly =
 ↪  SVC(kernel='poly',C=1,coef0=1,degree=2,gamma='auto')
```

où `C` correspond à l'inverse du paramètre de régularisation λ, `coef0` correspond à r, `degree` correspond à δ, et enfin `gamma='auto'` correspond à choisir $\gamma = 1/d$, c'est-à-dire $\gamma = 1/2$ en l'occurrence.

Question 4. — Entraîner des SVM avec noyaux polynomiaux de degrés allant de 2 à 8 en gardant les autres hyperparamètres constants et tracer pour chaque prédicteur la frontière de décision.

Question 5. — En utilisant `GridSearchCV`, entraîner un SVM avec noyau polynômial dont le degré a été choisi parmi $\{2, \dots, 8\}$ par validation croisée. Observer le degré sélectionné ainsi que le score du prédicteur obtenu sur l'échantillon de test.