Travaux pratiques de Machine learning

Cycle pluridisciplinaire d'études supérieures
Université Paris sciences et lettres

Joon Kwon

vendredi 24 mars 2023

Nous allons travailler avec le jeu de données de Ronald Fischer, qui date de 1936, et qui regroupe des données sur des iris.

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

iris = sns.load_dataset("iris")
```

`iris` contient alors les données sous la forme d'une *DataFrame* : il s'agit d'un type de données fourni par la librairie Pandas, qui permet de stocker des données sous la forme d'un tableau dont les colonnes sont nommées, et qui fournit un grand nombre de fonctions pour explorer et manipuler ces données. On peut afficher les 5 premières lignes avec la commande suivante.

```
iris.head()
```

On voit que les 4 premières colonnes correspondent respectivement à la longueur du sépale, la largeur du sépale, la longueur de la pétale et la largeur de

la pétale (toutes ces mesures sont en centimètres). Ces 4 variables vont constituer l'entrée. La cinquième colonne correspond à l'espèce de l'iris (qui peut être : Setosa, Versicolor ou Virginica) et constitue la sortie. Il s'agit donc d'un problème de classification : on souhaite construire un prédicteur prédisant l'espèce en fonction des autres caractéristiques. Par ailleurs, la `DataFrame` comporte 150 lignes : il s'agit du nombre de données.

Nous souhaitons dans un premier temps simplifier le jeu de données en réduisant la dimension de l'espace des entrées de 4 à 2. Il s'agit donc de déterminer quelles sont les deux variables explicatives qui semblent les plus prometteuses pour la prédiction de l'espèce. Pour ce faire, nous allons utiliser la fonction `sns.pairplot` de la librairie Seaborn.

```
sns.set()
sns.pairplot(iris, hue="species")
plt.show()
```

Question 1. — Choisir les deux variables explicatives qui semblent les plus prometteuses pour la prédiction de l'espèce, c'est-à-dire celles qui dans la figure semblent le mieux séparer les différentes espèces. Créer un `array X` contenant les colonnes correspondant aux deux variables explicatives sélectionnées, ainsi qu'un array `y` contenant la colonne correspondant à l'espèce (on pourra faire appel à `iris.values` qui renvoie l'ensemble des données de la `DataFrame iris` sous la forme d'un `array` NumPy).

Nous souhaitons à présent séparer le jeu de données en deux échantillons : un d'apprentissage de taille 90, et un de test de taille 60. Une première idée serait d'utiliser les 90 premières données pour constituer l'échantillon d'apprentissage et les 60 dernières pour l'échantillon de test.

```
X_train = X[:90]
y_train = y[:90]
X_test = X[90:]
y_test = y[90:]
```

Question 2. — Observer les échantillons ainsi obtenus et expliquer en quoi ils sont problématiques.

On peut remédier au problème précédent de la façon suivante.

```
from sklearn.utils import shuffle
X, y = shuffle(X,y)

X_train = X[:90]
y_train = y[:90]
X_test = X[90:]
y_test = y[90:]
```

On peut à présent entraîner un prédicteur kNN, pour $k = 3$ par exemple.

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

La distance utilisée par défaut est la distance euclidienne. Les prédicteurs de classification de `scikit-learn` fournissent une fonction `.score` (donc ici `knn.score`) qui donne la proportion de bonnes prédictions sur un échantillon fourni en argument.

Question 3. — Quelle est la relation entre le score défini ci-dessus et le risque empirique ?

Question 4. — Calculer le prédicteur kNN pour $k \in \{1, \dots, 20\}$. Tracer les courbes des scores calculés sur les échantillons d'apprentissage et de test respectivement. En déduire une valeur de k qui semble avoir produit le meilleur prédicteur.

Question 5. — En reprenant le code écrit précédemment, définir une fonction `best_knn_score` qui prend en argument deux `array` `X` et `y`, construit les échantillons d'apprentissage et de test, calcule les prédicteurs kNN pour $k \in \{1, \dots, 20\}$, et qui finalement renvoie le meilleur score calculé sur l'échantillon de test. Exécuter plusieurs fois cette fonction. Que remarque-t-on ? À quoi cela est-ce dû ?

Question 6. — Écrire une fonction `best_knn_score_avg` qui prend en argument des `array` `X` et `y`, qui exécute 100 fois `best_knn_score(X,y)` et qui renvoie la moyenne des 100 scores obtenus.

L'algorithme kNN est sensible à l'échelle utilisée pour chaque variable explicative. Pour s'en rendre compte, nous allons modifier l'échelle d'une variable

explicative et observer la conséquence sur la qualité du prédicteur obtenu. On commence par créer une copie `X_` de `X`.

```
X_ = X.copy()
```

La commande `X_ = X` n'aurait pas créé un copie, mais simplement un second nom pour le même objet : les modifications sur `X_` auraient alors également affecté `X`.

Question 7. — Modifier l'`array` `X_` en convertissant les données d'une des deux variables explicatives des centimètres aux mètres. L'utilisation de `X_` au lieu de `X` affecte-t-elle la qualité des prédicteurs construits ? Recommencer en convertissant cette fois-ci l'autre variable explicative en mètres (et en gardant la première en centimètres).

Lorsqu'on souhaite s'assurer que l'influence d'une variable explicative ne soit ni trop faible, ni trop élevée en raison de son échelle, il est recommandée de normaliser les données. Autrement dit, pour chaque variable explicative, il s'agit d'appliquer une transformation affine telle que les valeurs qui en résultent aient une moyenne nulle et un écart-type égal à 1. `scikit-learn` propose une fonction pour cela.

```
from sklearn import preprocessing
X_scaled = preprocessing.scale(X)
```

Question 8. — La normalisation des données donne-t-elle ici un meilleur résultat ?

Question 9. — Si on considère l'ensemble des 4 variables explicatives initialement disponibles, le prédicteur obtenu est-il meilleur ?