IPST-CNAM 10 Décembre 2021
RCP208

TP 3

K-means

L'objectif de cette séance de TP est de présenter l'utilisation des fonctionnalités de scikit-learn concernant la classification automatique avec *k-means*, ainsi que de contribuer à une meilleure compréhension de cette méthode et de l'impact sur les résultats de la distribution des données ou de la technique d'initialisation (initialisation aléatoire ou *k-means++*). Pour cela, sont d'abord examinées dans l'application 1 des données générées de façon contrôlée et ensuite des données réelles, sans ou avec un pré-traitement permettant de renforcer la séparation entre groupes. Nous traitons par la suite à travers l'application 2 un autre exemple pour mieux comprendre et maîtriser K-menas avec le langage de programmation Python.

I. La classification automatique dans scikit-learn :

Des méthodes de classification automatique variées sont disponibles dans scikit-learn. Parmi ces méthodes nous examinerons de plus près les K-moyennes (*K-means*) et ultérieurement la classification spectrale (*spectral clustering*).

Pour chaque méthode, l'implémentation permet d'obtenir un « modèle » (par ex., un ensemble de *k* centres de groupes) à partir des données (sous forme de tableau de `n_samples x n_features` ainsi que, pour certaines méthodes, de matrice de similarités `n_samples x n_samples`) et ensuite d'obtenir le numéro de groupe (*cluster*) pour les données initiales ou pour de nouvelles données.

La description de l'implémentation de la méthode des K-moyennes (*K-means*) se trouve dans http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

Nous nous servirons également de l'indice de Rand ajusté, voir http://scikit-learn.org/stable/modules/clustering.html#adjusted-rand-index, pour évaluer la cohérence entre des classifications différentes d'un même ensemble de données. Pour d'autres méthodes permettant de comparer les résultats de différentes classifications voir [WW07], [VEB10].

II. Application 1 :

1. Classification avec K-means de données générées

Démarrez par la génération de cinq groupes de 100 vecteurs chacun dans l'espace 3D, chacun suivant une loi normale (de moyenne nulle et de variance unitaire). Appliquez à chaque groupe une translation différente dans l'espace, générez les étiquettes de groupe, construisez l'ensemble total de données et mélangez ensuite les lignes de cet ensemble :

```python
import numpy as np    # si pas encore fait
from sklearn.utils import shuffle

# génération 100 points 3D suivant loi normale centrée
# chaque groupe est translaté d'un vecteur [3,3,3]
d1 = np.random.randn(100,3) + [3,3,3]
d2 = np.random.randn(100,3) + [-3,-3,-3]
d3 = np.random.randn(100,3) + [-3,3,3]
d4 = np.random.randn(100,3) + [-3,-3,3]
d5 = np.random.randn(100,3) + [3,3,-3]

# génération des étiquettes de chaque groupe
c1 = np.ones(100)
c2 = 2 * np.ones(100)
c3 = 3 * np.ones(100)
c4 = 4 * np.ones(100)
c5 = 5 * np.ones(100)

# concaténation des données dans une matrice
data = np.concatenate((d1,d2,d3,d4,d5))
labels = np.concatenate((c1, c2, c3, c4, c5))
print(data.shape)
# permutation aléatoire des lignes de la matrice data
data, labels = shuffle(data, labels)
```

Visualisez les groupes de départ :

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# La couleur des points dépend de leur étiquette (label)
ax.scatter(data[:,0], data[:,1], data[:,2], c=labels)
plt.show()
```

Appliquez la classification automatique avec *K-means*, d'abord avec un seul essai (une seule initialisation suivie d'une seule exécution de *K-means*, `n_init = 1`) utilisant la méthode d'initialisation `k-means++` :

```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=5, n_init=1, init='k-means++').fit(data)
```

Examinez les paramètres, les attributs et les méthodes de la classe `sklearn.cluster.KMeans` en suivant le lien indiqué plus haut.

On peut obtenir les groupes prédits pour les données à l'aide de la méthode `predict(X)` :

```
pred = kmeans.predict(data)
```

Les groupes associés aux exemples d'apprentissage sont également stockés dans l'attribut `kmeans.labels_` :

```
print(kmeans.labels_)
```

Visualisez les résultats de cette classification :

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data[:,0], data[:,1], data[:,2], c=pred)
plt.show()
```

Il est possible d'évaluer la cohérence entre les groupes de départ et le partitionnement trouvé par *K-means* en utilisant l'indice de Rand ajusté (voir le cours et la documentation) :

```
from sklearn import metrics
metrics.adjusted_rand_score(pred, labels)
```

L'appel à `metrics.adjusted_rand_score()` compare le partitionnement obtenu par la classification automatique (étiquettes de groupe de `pred`) avec le partitionnement correspondant aux groupes définis au départ (étiquettes stockées dans `labels`).

Appliquez maintenant la classification automatique avec *K-means* avec un seul essai (`n_init = 1`) utilisant la méthode d'initialisation `random` :

```
kmeans = KMeans(n_clusters=5, n_init=1, init='random').fit(data)
metrics.adjusted_rand_score(kmeans.labels_, labels)
```

Question 1:

Répétez plusieurs fois la classification avec chacune de ces deux méthodes d'initialisation et examinez à chaque fois la cohérence des groupes obtenus avec les groupes de départ. Que constatez-vous ? Expliquez.

Question 2 :

Variez le nombre de groupes (`n_clusters`) et faites plusieurs essais pour chaque valeur du nombre de groupes. Examinez de nouveau la stabilité des résultats en utilisant l'indice de Rand ajusté. Expliquez ce que vous constatez.

Question 3 :

Variez le nombre de groupes (`n_clusters`) entre 2 et 20, tracez le graphique d'évolution de la valeur finale atteinte par le coût (l'inertie, voir la documentation) pour chacune des valeurs de `n_clusters`.

Question 4 :

Générez 500 données suivant une distribution **uniforme** dans $[0,1)^3$ (données tridimensionnelles dans le cube unité). Appliquez sur ces données *K-means* avec `n_clusters=5` et initialisation aléatoire (`random`), et examinez la stabilité des résultats en utilisant l'indice de Rand. Appliquez sur ces mêmes données *K-means* avec toujours `n_clusters=5` mais une initialisation `k-means++`, examinez la stabilité des résultats. Attention, vous ne disposez plus de groupes définis au départ ; pour définir les groupes de référence, auxquels vous comparerez ceux issus des autres classifications, vous pouvez appliquer une première fois *K-means* avec `n_clusters=5, n_init=1, init='k-means++'`. Observez-vous des différences par rapport aux résultats obtenus sur les données générées au début de cette section (avec `np.random.randn`) ? Expliquez.

2. Classification avec K-means des données « textures » :

Pour rappel, ces données correspondent à 5500 observations décrites par 40 variables. Chaque observation appartient à une des 11 classes de textures ; chaque classe est représentée par 500 observations. Les données sont issues de https://www.elen.ucl.ac.be/neural-nets/Research/Projects/ELENA/databases/REAL/texture/. Nous appliquerons *K-means* à ces données, avec `n_clusters = 11`, et examinerons dans quelle mesure les groupes issus de la classification automatique se rapprochent des classes présentes.

Si les données ne sont pas dans le répertoire de travail il est nécessaire de les chercher d'abord. En ligne de commande :

```
%%bash
wget -nc http://cedric.cnam.fr/~crucianm/src/texture.dat
```

Mélangez les observations et appliquez *K-means* à ces données (attention, la dernière colonne contient les étiquettes de classe) :

```
textures = np.loadtxt('texture.dat')
np.random.shuffle(textures)
kmeans = KMeans(n_clusters=11).fit(textures[:,:40])
metrics.adjusted_rand_score(kmeans.labels_, textures[:,40])
```

On constate que les groupes issus de la classification automatique ne donnent que peu d'indications sur les classes présentes dans ces données.

Question 5 :

Appliquez l'analyse discriminante à ces données et appliquez de nouveau *K-means* avec `n_clusters = 11` aux données projetées dans l'espace discriminant. Que constatez-vous ? Expliquez. Visualisez les résultats.

III. Application 2 :

L'algorithme de clustering K-means signifie qu'il s'agit d'une technique d'apprentissage automatique non supervisée utilisée pour regrouper des points de données. Nous résoudrons le problème du regroupement des « groupes de revenus » en utilisant l'algorithme K-means. La méthode du coude « elbow » est une technique utilisée pour déterminer le nombre optimal de k.

Exécutez les différentes instructions et commenter chacun d'eux.

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
```

```
df = pd.read_csv("/home/dell/Bureau/income.csv")
print(df.head())
```

```
plt.scatter(df.Age,df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
plt.show()
```

```
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
print(y_predicted)
```

```
df['cluster']=y_predicted
print(df.head())
```

```
print(km.cluster_centers_)
```

```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()
plt.show()
```

```
scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])
```

```
print(df.head())
```

```
plt.scatter(df.Age,df['Income($)'])
plt.show()
```

```
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
print(y_predicted)
```

```
df['cluster']=y_predicted
print(df.head())
```

```
print(km.cluster_centers_)
```

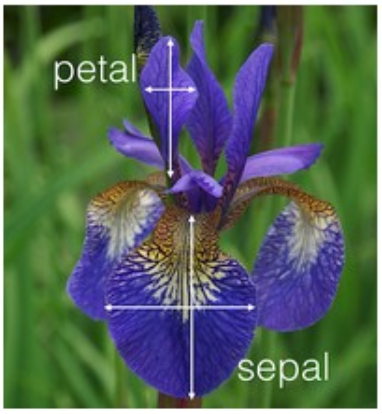
```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.legend()
plt.show()
```

```
## Elbow Plot

sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['Age','Income($)']])
    sse.append(km.inertia_)
```

```
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
plt.show()
```

Exercice :



1. Utilisez l'ensemble de données de fleurs d'iris de la bibliothèque sklearn et essayez de former des grappes de fleurs à l'aide des fonctionnalités de largeur et de longueur des pétales. Supprimez deux autres fonctionnalités pour plus de simplicité.
2. Déterminez si un prétraitement tel que la mise à l'échelle pourrait aider ici.
3. Tracez le tracé du coude et à partir de cela, déterminez la valeur optimale de k.