Travaux pratiques - K-means

(correspond à 1 séance de TP)

Références externes utiles :

- Documentation NumPy
- Documentation SciPy
- Documentation MatPlotLib
- Site scikit-learn
- Site langage python

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 des données générées de façon contrôlée et ensuite des données réelles vues en cours, sans ou avec un pré-traitement permettant de renforcer la séparation entre groupes.

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].

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 :

```
>>> import numpy as np    # si pas encore fait
>>> from numpy import newaxis

>>> d1 = np.random.randn(100,3) + [3,3,3]  # génération 100 points 3D suivant loi normale centrée en [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]
>>> c1 = np.ones(100)      # génération des étiquettes du groupe 1
>>> c2 = 2 * np.ones(100)  # génération des étiquettes du groupe 2
>>> c3 = 3 * np.ones(100)
>>> c4 = 4 * np.ones(100)
>>> c5 = 5 * np.ones(100)
>>> data1 = np.hstack((d1,c1[:,newaxis]))  # ajout des étiquettes comme 4ème colonne
>>> data2 = np.hstack((d2,c2[:,newaxis]))
>>> data3 = np.hstack((d3,c3[:,newaxis]))
>>> data4 = np.hstack((d4,c4[:,newaxis]))
>>> data5 = np.hstack((d5,c5[:,newaxis]))
>>> data = np.concatenate((data1,data2,data3,data4,data5))  # concaténation des données dans une matrice
```

TABLE DES MATIÈRES

RECHERCHE

Go

```
>>> data.shape
(500, 4)
>>> np.random.shuffle(data)   # permutation aléatoire des lignes de la matrice data
```

Visualisez les groupes de départ :

```
>>> import matplotlib.pyplot as plt
>>> from mpl_toolkits.mplot3d import Axes3D
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111, projection='3d')
>>> ax.scatter(data[:,0], data[:,1], data[:,2], c=data[:,3])
<mpl_toolkits.mplot3d.art3d.Patch3DCollection object at 0x...>
>>> 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++` :

```
>>> from sklearn.cluster import KMeans
>>> kmeans = KMeans(n_clusters=5, n_init=1, init='k-means++').fit(data[:,:3])
```

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

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=kmeans.labels_)
<mpl_toolkits.mplot3d.art3d.Patch3DCollection object at 0x...>
>>> 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(kmeans.labels_, data[:,3])
```

L'appel à `metrics.adjusted_rand_score()` compare le partitionnement obtenu par la classification automatique (étiquettes de groupe de `kmeans.labels_`) avec le partitionnement correspondant aux groupes définis au départ (étiquettes de la dernière colonne de `data`).

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[:,:3])
>>> metrics.adjusted_rand_score(kmeans.labels_, data[:,3])
```

Question :

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.

Correction :

La cohérence est bien plus forte lorsque l'initialisation est faite par la méthode `k-means++`. Cette méthode d'initialisation mène à des résultats plus stables.

Si on emploie le paramètre par défaut `n_init = 10` plutôt que `n_init = 1`, la variance des résultats diminue significativement, y compris pour l'initialisation `init='random'`.

Question :

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.

Correction :

La cohérence est plus forte lorsque le nombre de groupes demandés à l'algorithme de classification automatique correspond au nombre de groupes « naturels » dans les données. En effet, chercher dans les données un nombre de groupes différent revient à chercher à obtenir un modèle inadapté aux données, la variance des résultats sera donc plus forte (la cohérence plus faible). Pour des données réelles, en dimension élevée (donc non

directement visualisables), une bonne stabilité des résultats indique que le nombre de groupes est adapté aux données.

Question :

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`.

Correction :

Comme indiqué dans la documentation (voir lien plus haut), la valeur finale de l'inertie est à chaque fois dans `kmeans.inertia_`. Le graphique montre que la valeur finale atteinte par le coût diminue avec l'augmentation du nombre de groupes. Ces valeurs ne constituent donc pas une bonne mesure pour comparer des solutions avec des nombres de groupes différents. On constate également que l'inertie finale diminue fortement lorsque le nombre de groupes augmente de 1 à 5, ensuite elle diminue nettement moins vite. Cela indique aussi que chercher 5 groupes est pertinent pour ces données.

Question :

($\rightarrow$ **Compte-rendu**) 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.

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 :

```
[nom@machine ~]$ wget 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 :

($\rightarrow$ **Compte-rendu**) 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.