Loughborough University

COP528 Applied Machine Learning (LABs)

LAB Day 03. Unsupervised learning 01

Introduction

The aim of this session is to understand a couple of clustering methods, including the KMean and GMM algorithms, and evaluate them based on a couple of evaluation metrics. The experiments include **3** tutorial examples (one of these is optional) and **2** tasks. You may follow the tutorial to learn the functions and then complete the tasks.

Tutorial 01. Clustering model evaluation

Clustering model evaluation is important in order to objectively assess how good a clustering model is, especially when visualisation of high latitude data is not sufficient for evaluation.

Clustering model evaluation metrics provided by the metrics module

Method name	Ground-truth	Best value	Sklearn functions
Purity score	Required	1.0	Self-defined
Silhouette Scores	Not required	1.0	silhouette_score

(1) Self-defined Purity score function

```
import numpy as np
from sklearn import metrics

def purity_score(y_true, y_pred):
    # compute contingency matrix (also called confusion matrix)
    contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
    # return purity
    return np.sum(np.amax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)
```

(2) The following is examples of the Silhouette Scores calculation.

```
from sklearn import metrics
silhouetteScore=metrics.silhouette_score(X, y_pred, metric='euclidean', sample_size=None,
random_state=None)
```

Tutorial 02. k-means clustering algorithm

(1) Let's make our own KMeans clustering. Follow the code below, especially the function definitions and loops, to understand the KMeans clustering process. (Optional)

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

# Distance between two points
def distance(e1, e2):
    return np.sqrt((e1[0]-e2[0])**2+(e1[1]-e2[1])**2)

#  Pick K random points in arr
def Pick_K_random_points(arr,k):

    r = np.random.randint(arr.__len__() - 1, size=(k))
    K_random_point= arr[r]

    return K_random_point

# Get the collection centre
def means(arr):
        return np.array([np.mean([e[0] for e in arr]), np.mean([e[1] for e in arr])])

#get the teat dataset
array_test = np.random.randint(100, size=(10, 1, 2))[:, 0, :]
print(array_test)
plt.scatter(array_test[:, 0], array_test[:, 1], marker='o')
plt.show()

# Set the number of clusters K
K=3
# Pick K random points，  Pick K random points as cluster
k_Centre=Pick_K_random_points(array_test,K)
print(k_Centre)
## Initialize the clustering array
cla_arr = [[]]
for i in range(K-1):
    cla_arr.append([])
```

```
## Iterative clustering
D=np.zeros(K)
D_stop=np.ones(K)
#Determining if the centre point has changed


cla_temp = cla_arr
N=0 #Define an Iterative count N
while all(D!=D_stop) and N<20: #Centre no longer changes or iterates to a certain times of iterations
    # Initialize the cla_temp array

    for e in array_test:    # Clustering each element of the set to the nearest class
        ki = 0                          # Assumes closest to first centre
        min_d = distance(e, k_Centre[ki])
        for j in range(1, k_Centre.__len__()):
            if distance(e, k_Centre[j]) < min_d:    # Find closer clustering centres
                min_d = distance(e, k_Centre[j])
                ki = j
        cla_temp[ki].append(e)

    Clustering=cla_temp.copy()
    print(Clustering)

    # Updating the Clustering Centre
    for k in range(k_Centre.__len__()):
        k_Centre
        new_k_Centre=means(cla_temp[k])
        if all(k_Centre[k] == means(cla_temp[k])):
            D[k]=1.
        else:
            D[k]=0.

        k_Centre[k] = means(cla_temp[k])
        cla_temp[k] = []
    N=N+1
    print(k_Centre)


## Visualisation
col = ['HotPink', 'Aqua', 'Chartreuse', 'yellow', 'LightSalmon']
for i in range(K):
    plt.scatter(k_Centre[i][0], k_Centre[i][1], linewidth=10, color=col[i])
    plt.scatter([e[0] for e in Clustering[i]], [e[1] for e in Clustering[i]], color=col[i])
plt.show()
```

(2) We can also directly use the KMeans function in sklearn for clustering, with the following code.

```
# generate the experimental data
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.datasets import make_blobs
# X is the sample feature, Y is the sample cluster category, 1000 samples, 2 features per sample, 4 clusters, cluster centres in [-1,-1], [0,0], [1,1], [2,2], cluster variances [0.4, 0.2, 0.2, 0.2] respectively
X, y = make_blobs(n_samples=1000, n_features=2, centers=[[-1,-1], [0,0], [1,1], [2,2]], cluster_std=[0.4, 0.2, 0.2, 0.2],
                  random_state =9)
plt.scatter(X[:, 0], X[:, 1], marker='o')
plt.show()


from sklearn.cluster import KMeans
y_pred = KMeans(n_clusters=2, random_state=9).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.show()
```

Task 01. k-means clustering

Use k-means to make clusters of the data and fine-tune the cluster number, use Silhouette Scores to evaluate the above clusters.

Fine-tune the cluster number based on the dataset and examples given in Tutorial 03 and evaluate the clustering results using Silhouette Scores. Review the visualisation operations in Day02 and present the results of the clustering with the evaluation results visually.

Generate the experimental data

```
# generate the experimental data
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.datasets import make_blobs
# X is the sample feature, Y is the sample cluster category, 1000 samples, 2 features per sample, 4 clusters,
cluster centres in [-1,-1], [0,0], [1,1], [2,2], cluster variances [0.4, 0.2, 0.2, 0.2] respectively
X, y = make_blobs(n_samples=1000, n_features=2, centers=[[-1,-1], [0,0], [1,1], [2,2]], cluster_std=[0.4, 0.2, 0.2,
0.2],
                  random_state =9)
plt.scatter(X[:, 0], X[:, 1], marker='o')
plt.show()
```

Tutorial 03. Gaussian Mixed Model (GMM)

Here is an example of the use of the GMM function in sklearn.

(1) First generate the experimental data using the following code

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.datasets import make_blobs
# X is the sample feature, Y is the sample cluster category, 1000 samples, 4 features per sample, 5 clusters, cluster centres in [-1,-1], [-1,2], [0,0], [1,1], [2,2], cluster variances [0.4, 0.3, 0.2, 0.2, 0.1] respectively.
X, y = make_blobs(n_samples=1000, n_features=4, centers=[[-1,-1],[-1,2], [0,0], [1,1], [2,2]], cluster_std=[0.4,0.3, 0.2, 0.2, 0.1],
                  random_state =9)
plt.scatter(X[:, 0], X[:, 1], marker='o')
plt.show()
```

(2) The data is clustered using the GMM and two evaluation scores are calculated.

```
from sklearn.mixture import GaussianMixture as GMM
gmm = GMM(n_components=5).fit(X)
y_pred = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_pred, s=40, cmap='viridis')

from sklearn import metrics
silhouetteScore=metrics.silhouette_score(X, y_pred, metric='euclidean', sample_size=None, random_state=None)
print(silhouetteScore)
```

Task 02. Clustering by using KMean and GMM.

Based on the provided data, use k-mean and GMM to find clusters with the following data:

Classic dataset about geyser eruptions. Each row represents an observed eruption of the Old Faithful Geyser in Yellowstone National Park. The eruptions column represents the duration of the eruption in minutes, and the waiting column represents the duration in minutes until the next eruption.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('https://raw.githubusercontent.com/lborohfang/COP528AML_Files/main/faithful_data.csv')
print(df.head(5))
faithful_data = np.asarray(df.iloc[:,:])
plt.scatter(faithful_data[:, 0], faithful_data[:, 1], marker='o')
plt.show()
```