COP528 Applied Machine Learning (LABs)

LAB Day 04. Unsupervised learning 02

Introduction

The aim of this session is to understand two classical clustering algorithms: Agglomerative Clustering and DBSCAN. The experiment includes **2** tutorial examples and **4** tasks. You may follow the tutorial to learn the basic functions and then complete the tasks.

Tutorial 01. Agglomerative Clustering

(1) Use the following code to call one of the unsupervised learning methods - Agglomerative Clustering in sklearn to group iris data into 3 classes. Write several lines of codes to compare the results with the ground truth. Please use this example to re-think the difference between supervised learning and unsupervised learning. (How do you think about the unsupervised learning method?)

```
#Importing required libraries
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
import numpy as np
import matplotlib.pyplot as plt

#Getting the data ready
data = load_iris()
df = data.data
#Selecting certain features based on which clustering is done
df = df[:,1:3]

#Creating the model
agg_clustering = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'ward')
#predicting the labels
labels = agg_clustering.fit_predict(df)
```

(2) Use the following code to build the dendrogram of the iris data based on the first two features. Here we use the linkage and dendrogram functions in scipy.cluster.hierarchy.

```
#Importing libraries
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram , linkage

#Getting the data ready

data = load_iris()
df = data.data
#Selecting certain features based on which clustering is done
df = df[:,1:3]

#Linkage Matrix
Z = linkage(df, method = 'average')

plt.figure(figsize=(10,6))
#plotting dendrogram
dendro = dendrogram(Z)

plt.title('Dendrogram')
plt.ylabel('Euclidean distance')
plt.show()
```

Task 01. Agglomerative Clustering 01

MSc Data Science students have taken 4 modules and got their coursework marks. These have been saved in a csv file.

(1) Use the following code to load the data.

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

from scipy.cluster.hierarchy import dendrogram , linkage
df = pd.read_csv('https://raw.githubusercontent.com/lborohfang/COP528AML_Files/main/student.csv')

print(df.head(10))
student = np.asarray(df.iloc[:,1:])
```

(2) Use the function in sklrean to calculate the pairwise distance table (hint: Day02-Tutorial03), and use the agglomerative clustering to cluster the students into classes.

```
# *Solutions Example
from sklearn.metrics import DistanceMetric
dist = DistanceMetric.get_metric('euclidean')
print(dist.pairwise(student))

from sklearn.metrics import DistanceMetric
dist = DistanceMetric.get_metric('manhattan')
print(dist.pairwise(student))


#Agglomerative Clustering
from sklearn import cluster
clst=cluster.AgglomerativeClustering(n_clusters=2)
y_pred_Agglome=clst.fit_predict(student)

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

(3) Please explain what is in the Z matrix in Tutorial 01-(2).

```
# *Solutions Example
# Linkage Matrix
Z = linkage(student, method = 'single')
print(Z)
#plotting dendrogram
dendro = dendrogram(Z)
plt.title('Dendrogram')
plt.ylabel('Euclidean distance')
plt.show()

# In the Z matrix, each row represent the merge process
# the first and second elements of each row represent the cluster(node) id for the merging
# the third element represents the distance between the two clusters
# the fourth value represents the number of original observations in the newly formed cluster.
```

Task 02. Agglomerative Clustering 02

Given the following dataset, using Agglomerative algorithm to cluster the data and evaluate the performance.

```
from sklearn import datasets
df=datasets.make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0,
n_repeated=0,n_classes=4, n_clusters_per_class=1, weights=None, flip_y=0.01, class_sep=1.0, hypercube=True,
shift=0.0, scale=1.0,shuffle=True, random_state=None)
```

*Solutions Example

```
# from sklearn import datasets
df=datasets.make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0,
n_repeated=0,n_classes=4, n_clusters_per_class=1, weights=None, flip_y=0.01, class_sep=1.0, hypercube=True,
shift=0.0, scale=1.0,shuffle=True, random_state=None)
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram , linkage
#Linkage Matrix
Z = linkage(df[0], method = 'average')
#plotting dendrogram
dendro = dendrogram(Z)
plt.title('Dendrogram')
plt.ylabel('Euclidean distance')
plt.show()


#The number of clusters that can be observed is 4
from sklearn.cluster import AgglomerativeClustering
#Creating the model

agg_clustering = AgglomerativeClustering(n_clusters = 4, affinity = 'euclidean', linkage = 'ward')

#predicting the labels
labels = agg_clustering.fit_predict(df[0])

# plt.scatter(df[0][0],df[0][1] ,marker='o')
plt.scatter(df[0][:,0],df[0][:,1],c=labels, s=40, cmap='viridis')
plt.show()
```

Tutorial 02. DBSCAN

DBSCAN stands for density-based spatial clustering of applications with noise. It is able to find arbitrary shaped clusters and clusters with noise (i.e. outliers). The main idea behind DBSCAN is that a point belongs to a cluster if it is close to many points from that cluster.

(1) Generation of data sets for testing DBSCAN.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
%matplotlib inline
X1, y1=datasets.make_circles(n_samples=5000, factor=.6,
                              noise=.05)
X2, y2 = datasets.make_blobs(n_samples=1000, n_features=2, centers=[[1.2,1.2]],
cluster_std=[[.1]],
          random_state=9)

X = np.concatenate((X1, X2))
plt.scatter(X[:, 0], X[:, 1], marker='o')
plt.show()
```

(2) Clustering using the DBSCAN function in sklearn. Please observe the distribution of the data set and the clustering results.

```
from sklearn.cluster import DBSCAN
y_pred = DBSCAN().fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.show()
```

(3) Re-clustering by adjusting the eps and min_samples of DBSCAN and observing the adjusted clustering results:

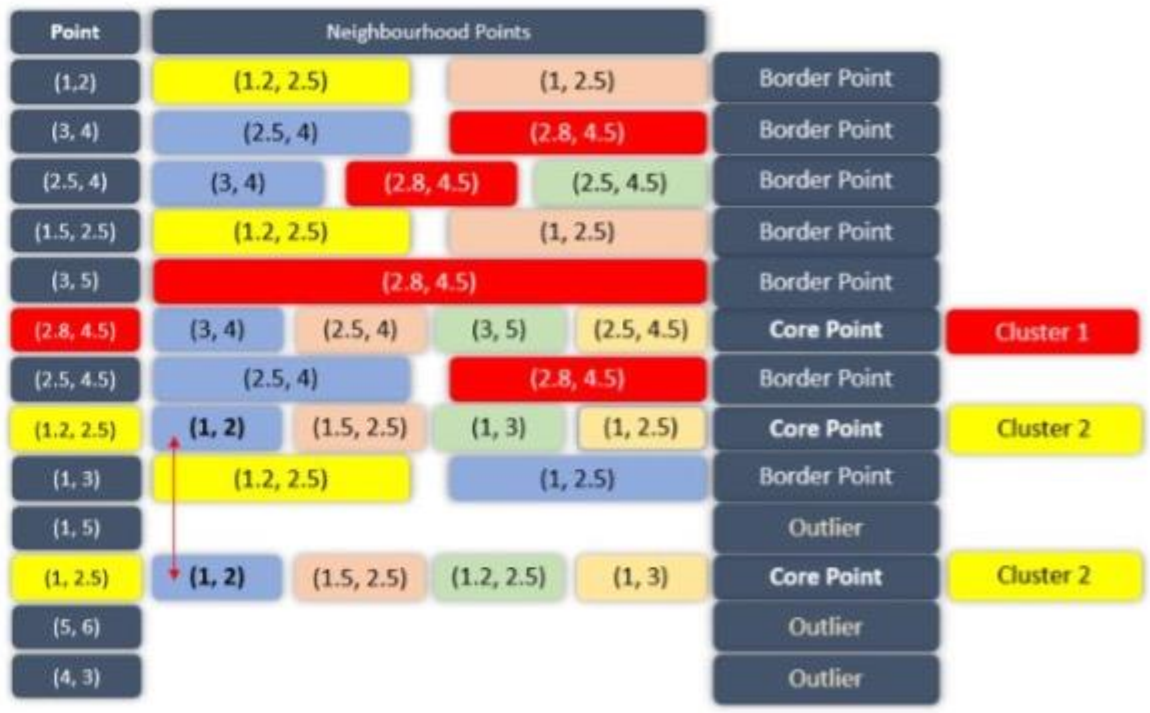
```
y_pred = DBSCAN(eps = 0.1, min_samples = 10).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.show()
```

(4) Review the KMeans algorithm and perform KMeans clustering on this dataset. Observe the clustering results and reflect on the differences.

Task 03. Clustering

Task 01. You are given the following points (1, 2), (3, 4), (2.5, 4), (1.5, 2.5), (3, 5), (2.8, 4.5), (2.5, 4.5), (1.2, 2.5), (1, 3), (1, 5), (1, 2.5), (5, 6), (4, 3). Assume that eps = 0.6 and MinPts = 4, please find out which are core points, which are border points, and which are outlier points. Then please build the clusters based on direct density reachable and density connected relationship.

*Solutions Example



Point	Neighbourhood Points					
(1,2)	(1.2, 2.5)	(1, 2.5)			Border Point	
(3, 4)	(2.5, 4)	(2.8, 4.5)			Border Point	
(2.5, 4)	(3, 4)	(2.8, 4.5)	(2.5, 4.5)		Border Point	
(1.5, 2.5)	(1.2, 2.5)	(1, 2.5)			Border Point	
(3, 5)	(2.8, 4.5)				Border Point	
(2.8, 4.5)	(3, 4)	(2.5, 4)	(3, 5)	(2.5, 4.5)	Core Point	Cluster 1
(2.5, 4.5)	(2.5, 4)	(2.8, 4.5)			Border Point	
(1.2, 2.5)	(1, 2)	(1.5, 2.5)	(1, 3)	(1, 2.5)	Core Point	Cluster 2
(1, 3)	(1.2, 2.5)	(1, 2.5)			Border Point	
(1, 5)					Outlier	
(1, 2.5)	(1, 2)	(1.5, 2.5)	(1.2, 2.5)	(1, 3)	Core Point	Cluster 2
(5, 6)					Outlier	
(4, 3)					Outlier	

Task 04. Experiments with different types of clustering algorithms

Based on the provided data, use different type of clustering algorithms, i.e., DBSCAN and compare them with the Agglomerative clustering results.

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
df = pd.read_csv('https://raw.githubusercontent.com/lborohfang/COP528AML_Files/main/faithful_data.csv')
print(df.head(10))
faithful_data = np.asarray(df.iloc[:,1:])
```

*Solutions Example

```
# *Solutions Example
#load the dataset
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()
# Data analysis and standardisation
# MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(faithful_data)
X_MinMax=scaler.transform(faithful_data)
plt.scatter(X_MinMax[:, 0], X_MinMax[:, 1], marker='o')
plt.show()
# DBSCAN
from sklearn.cluster import DBSCAN
y_pred_DBSCAN = DBSCAN(eps = 0.2, min_samples = 20).fit_predict(X_MinMax)
plt.scatter(X_MinMax[:, 0], X_MinMax[:, 1], c=y_pred_DBSCAN)
plt.show()
silhouetteScore=metrics.silhouette_score(X_MinMax, y_pred_DBSCAN, metric='euclidean', sample_size=None,
random_state=None)
print(silhouetteScore)

# Agglomerative
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram , linkage
#Linkage Matrix
Z = linkage(X_MinMax, method = 'average')
#plotting dendrogram
dendro = dendrogram(Z)
plt.title('Dendrogram')
plt.ylabel('Euclidean distance')
plt.show()
from sklearn import cluster
clst=cluster.AgglomerativeClustering(n_clusters=2)
y_pred_Agglome=clst.fit_predict(X_MinMax)
plt.scatter(X_MinMax[:, 0], X_MinMax[:, 1], c=y_pred_Agglome)
plt.show()
silhouetteScore=metrics.silhouette_score(X_MinMax, y_pred_Agglome, metric='euclidean', sample_size=None,
random_state=None)
print(silhouetteScore)
```