Pattern Processing using AI Practical File

COSCE60

Nikhil Gupta
2019UCO1719
COE Section 3

Q1 Write a python program to implement a simple Chatbot.

Code:

```python
# %%
import json
import requests

# %%
def chatbot():
    print("Hi! I'm a weather prediction chatbot")
    api_key = "f37d0c3cb6c045218e1152633232504"
    while(1):
        print("Which city would you like the weather forcast for? Type 'exit' to quit")
        city = input().lower()
        if(city=="exit"):
            break
        url = f"https://api.weatherapi.com/v1/forecast.json?key={api_key}&q={city}"
        response = requests.get(url)
        data = json.loads(response.text)
        if len(data)>1:
            temp = data["current"]["temp_c"]
            description = data["current"]["condition"]["text"]
            humidity = data["current"]["humidity"]
            wind_speed = data["current"]["wind_kph"]
            print(f"The weather in {city.title()} is {description}, with a temperature of {temp}°C , humidity of {humidity}%, and wind speed of {wind_speed} km/h.")
            print()
        else:
            print("City not found. Please try again.")
            print()
    print("Thank you for using the chatbot.")


# %%
chatbot()
```

```
# %%
```

Output:

```
Hi! I'm a weather prediction chatbot
Which city would you like the weather forcast for? Type 'exit' to quit
The weather in New Delhi is Mist, with a temperature of 32.0°C , humidity of 26%, and wind speed of 9.0 km/h.

Which city would you like the weather forcast for? Type 'exit' to quit
The weather in Mumbai is Overcast, with a temperature of 31.0°C , humidity of 59%, and wind speed of 13.0 km/h.

Which city would you like the weather forcast for? Type 'exit' to quit
The weather in Landon is Partly cloudy, with a temperature of 5.0°C , humidity of 81%, and wind speed of 19.1 km/h.

Which city would you like the weather forcast for? Type 'exit' to quit
The weather in Dwarka is Sunny, with a temperature of 31.9°C , humidity of 59%, and wind speed of 16.9 km/h.

Which city would you like the weather forcast for? Type 'exit' to quit
City not found. Please try again.

Which city would you like the weather forcast for? Type 'exit' to quit
The weather in Mexico is Partly cloudy, with a temperature of 21.0°C , humidity of 23%, and wind speed of 13.0 km/h.

Which city would you like the weather forcast for? Type 'exit' to quit
Thank you for using the chatbot.
```

Q2 Write a program to implement k-means clustering from scratch.

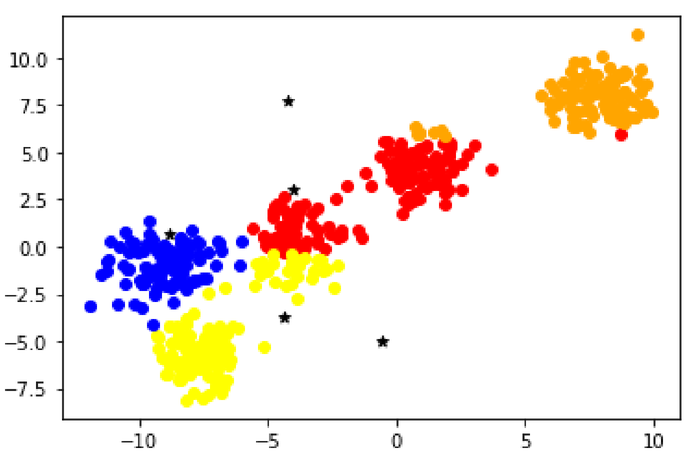
Code:

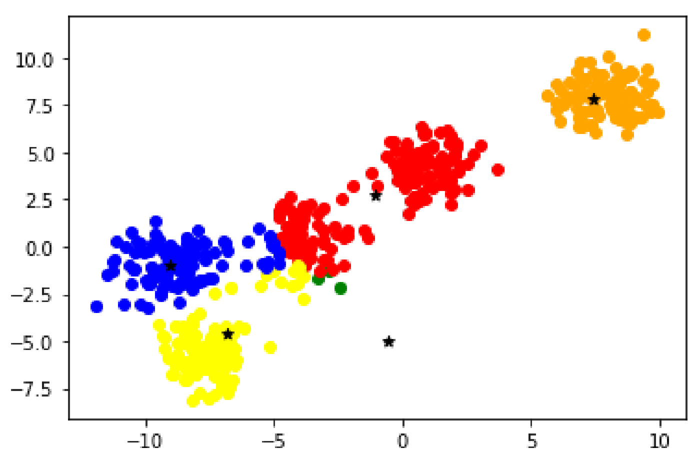
```
# %%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# %%
X,Y = make_blobs(n_samples=500,n_features=2,centers=5,random_state=3)

# %%
plt.figure(0)
plt.scatter(X[:,0],X[:,1],c=Y)
plt.show()
```

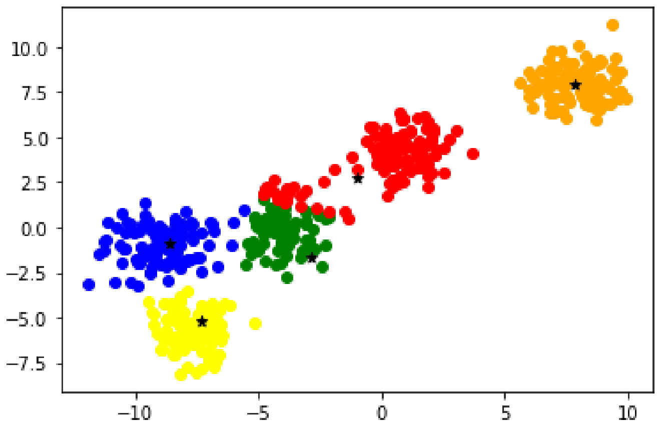
```python
# %%
k = 5
color = ["green","red","yellow","blue","orange"]

clusters = {}

for i in range(k):
    center = 10*(2*np.random.random((X.shape[1],))-1)
    points = []
    cluster = {
        "center":center,
        "points":points,
        "color":color[i]
    }

    clusters[i] = cluster


# %%
print(clusters)

# %%
def distance(x1,x2):
    return np.sqrt(np.sum((x1-x2)**2))

def assignPointsToCluster(clusters):
    for i in range(X.shape[0]):
        clust_x = X[i]
        dist = []
        for kx in range(k):
            d = distance(clust_x,clusters[kx]['center'])
            dist.append(d)

        idx = np.argmin(dist)
        clusters[idx]['points'].append(clust_x)

def updateCluster(clusters):
    for kx in range(k):
        pts = np.array(clusters[kx]['points'])
```

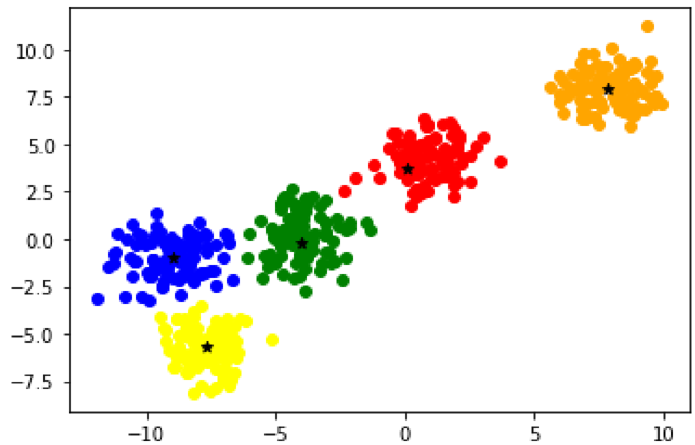
```
        if(pts.shape[0]>0):
            new_centers = np.mean(pts,axis=0)
            clusters[kx]['center'] = new_centers
            clusters[kx]['points'] = []

def plotClusters(clusters):

    plt.figure()
    for kx in range(k):
        pts = np.array(clusters[kx]['points'])

        try:
            plt.scatter(pts[:,0],pts[:,1],color=clusters[kx]['color'])
        except:
            pass

        cent = clusters[kx]['center']
        plt.scatter(cent[0],cent[1],color='black',marker="*")

# %%
epoch = 5
for i in range(epoch):
    assignPointsToCluster(clusters)
    plotClusters(clusters)
    updateCluster(clusters)

# %%
```

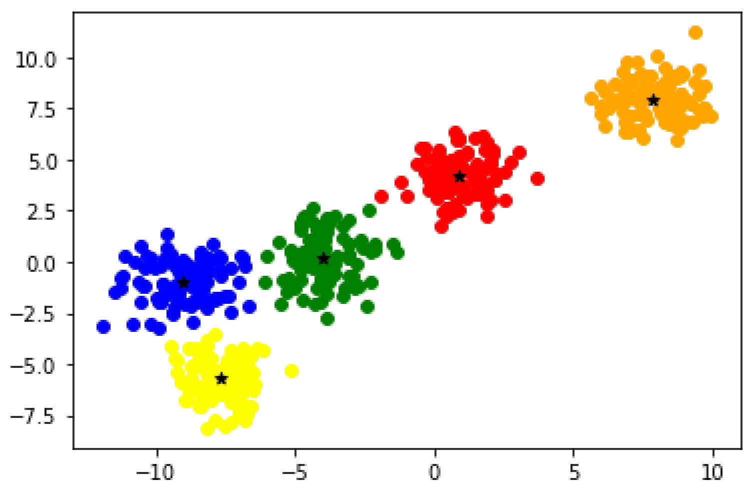
Output:











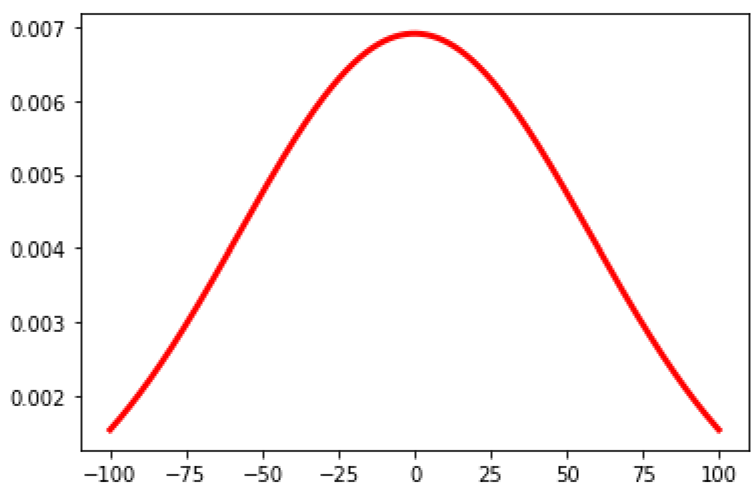
Q3 Generating samples of Gaussian (normal) distributions and plotting them for visualization

Code:

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

```
# %%
x_axis = np.arange(-100,100,0.1)
print(x_axis)

# %%
mean = np.mean(x_axis)
std = np.std(x_axis)
print(mean,std)

# %%
y_axis = 1/(std * np.sqrt(2 * np.pi)) * np.exp( - (x_axis - mean)**2 / (2
* std**2) )

# %%
plt.plot(x_axis,y_axis,linewidth=3, color='r')
plt.show()

# %%
```

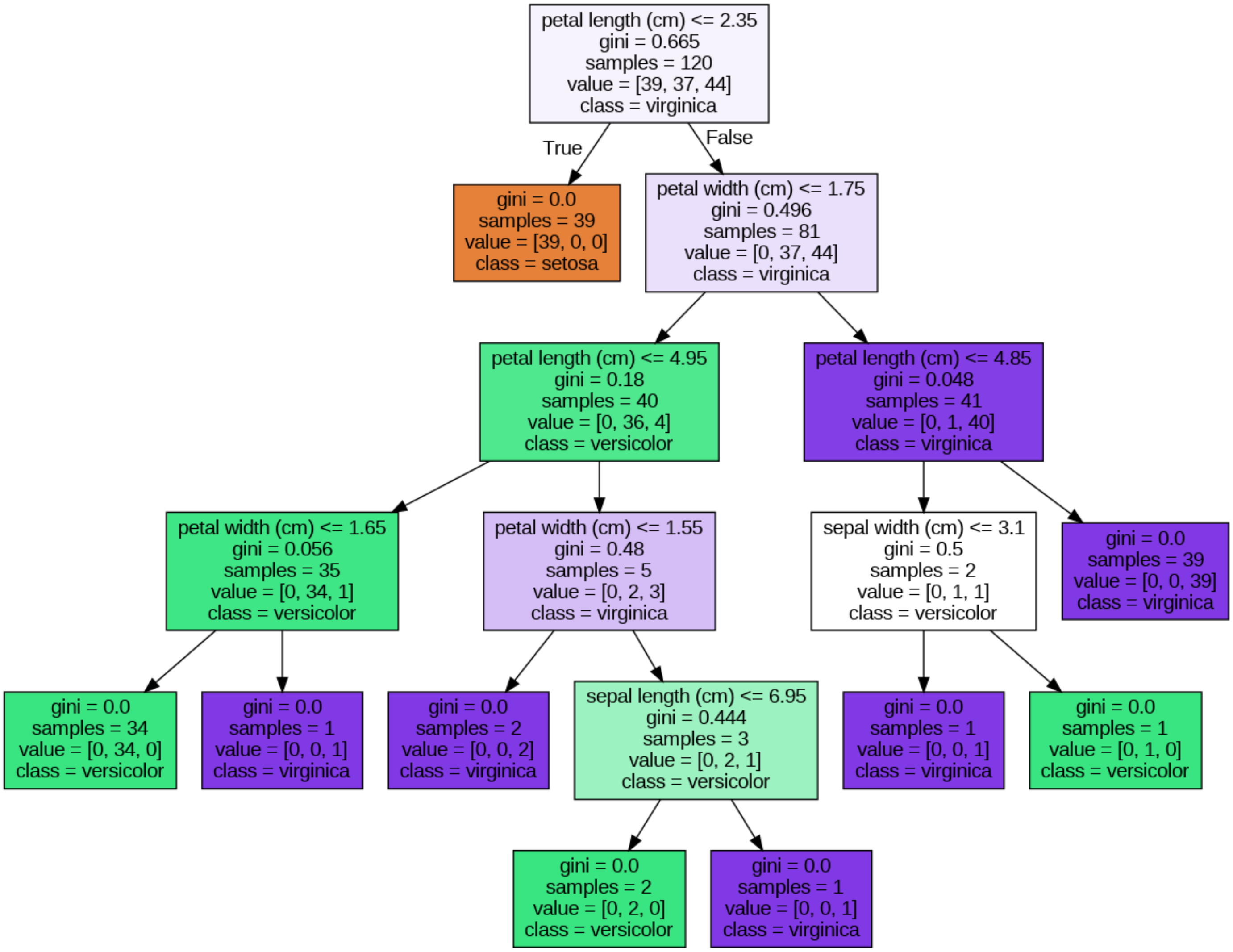
Output:



Q4 Implement Decision Tree algorithms.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import graphviz
from sklearn.tree import DecisionTreeClassifier

iris = datasets.load_iris()

X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =
0, test_size = 0.2)

tree = DecisionTreeClassifier(max_depth = 5)
tree.fit(X_train, y_train)
tree_predictions = tree.predict(X_test)

cm = confusion_matrix(y_test, tree_predictions)
print(cm)

score =accuracy_score(tree_predictions, y_test)
print(score)

tree_formed = export_graphviz(tree, out_file = None, feature_names =
iris.feature_names, class_names = iris.target_names, filled = True)
graph = graphviz.Source(tree_formed, format="png")

graph
```

Output:



petal length (cm) <= 2.35
gini = 0.665
samples = 120
value = [39, 37, 44]
class = virginica
True
False
gini = 0.0
samples = 39
value = [39, 0, 0]
class = setosa
petal width (cm) <= 1.75
gini = 0.496
samples = 81
value = [0, 37, 44]
class = virginica
petal length (cm) <= 4.95
gini = 0.18
samples = 40
value = [0, 36, 4]
class = versicolor
petal length (cm) <= 4.85
gini = 0.048
samples = 41
value = [0, 1, 40]
class = virginica
petal width (cm) <= 1.65
gini = 0.056
samples = 35
value = [0, 34, 1]
class = versicolor
petal width (cm) <= 1.55
gini = 0.48
samples = 5
value = [0, 2, 3]
class = virginica
sepal width (cm) <= 3.1
gini = 0.5
samples = 2
value = [0, 1, 1]
class = versicolor
gini = 0.0
samples = 39
value = [0, 0, 39]
class = virginica
gini = 0.0
samples = 34
value = [0, 34, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
sepal length (cm) <= 6.95
gini = 0.444
samples = 3
value = [0, 2, 1]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

Q5 Implement SVM.

Code:

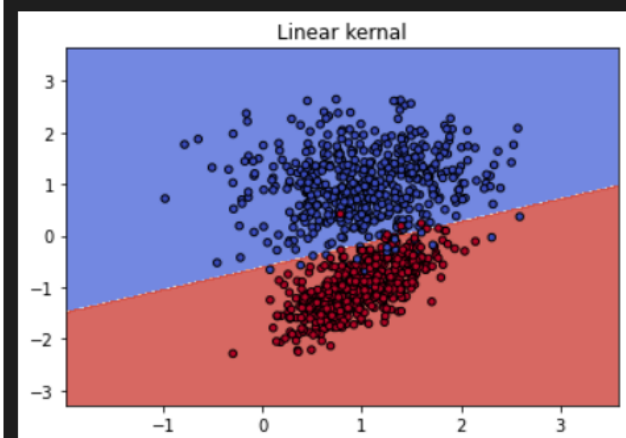
```
# %% [markdown]
# ### Generate Dataset

# %%
from sklearn.datasets import  make_classification
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# %%
X,Y =
make_classification(n_classes=2,n_samples=1000,n_clusters_per_class=1,rand
om_state=3,n_features=2,n_informative=2,n_redundant=0)

# %%
plt.scatter(X[:,0],X[:,1],c=Y)
plt.show()

# %%
plt.scatter(X[:,0],X[:,1],c=Y)
plt.show()

# %%
def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out
```

```
def plot(model, X,Y,title):
    fig, ax = plt.subplots()

    xx, yy = make_meshgrid(X[:,0],X[:,1])

    plot_contours(ax, model, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.coolwarm, s=20,
edgecolors='k')
    ax.set_title(title)
    plt.show()


# %%
from sklearn import svm

# %%
svc = svm.SVC(kernel='linear')
svc.fit(X,Y)
print(svc.score(X,Y))
plot(svc,X,Y,"Linear kernal")

# %%
# RBF Kernel
svc = svm.SVC()
svc.fit(X,Y)
print(svc.score(X,Y))
plot(svc,X,Y,"RBF kernal")

# %%
# Polynomial Kernel
svc = svm.SVC(kernel='poly')
svc.fit(X,Y)
print(svc.score(X,Y))
plot(svc,X,Y,"Poly kernal")

# %%
def custom_kernel(x1,x2):
    return np.square(np.dot(x1,x2.T))

svc = svm.SVC(kernel=custom_kernel)
```

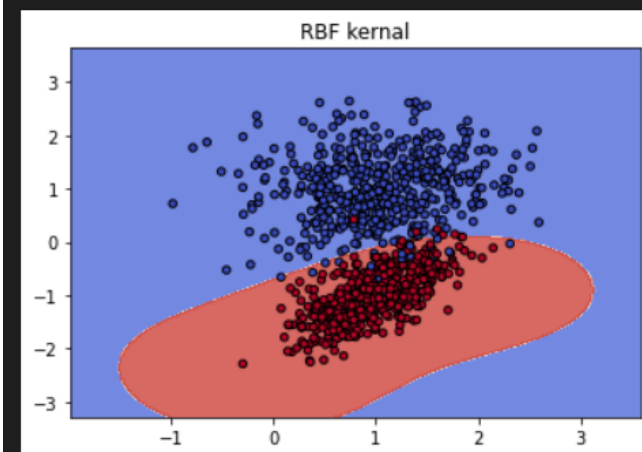
```
svc.fit(X,Y)
print(svc.score(X,Y))
plot(svc,X,Y,"Custom kernal")

# %%
```

Output:

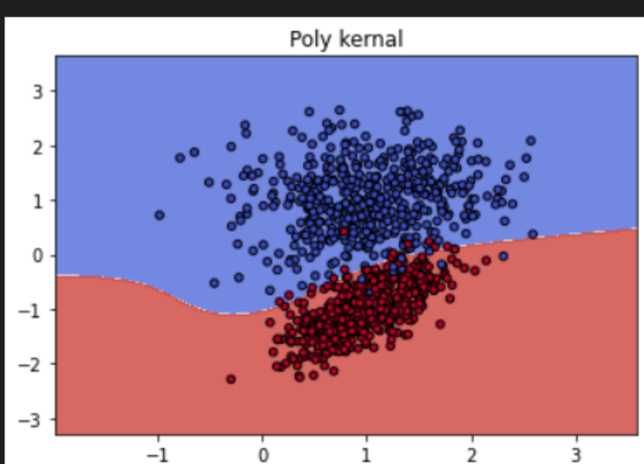
```
0.969
```



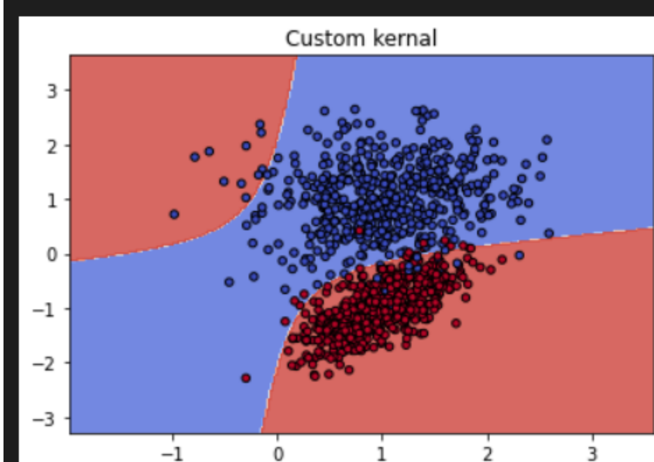

```
0.971
```




```
0.969
```




```
0.957
```

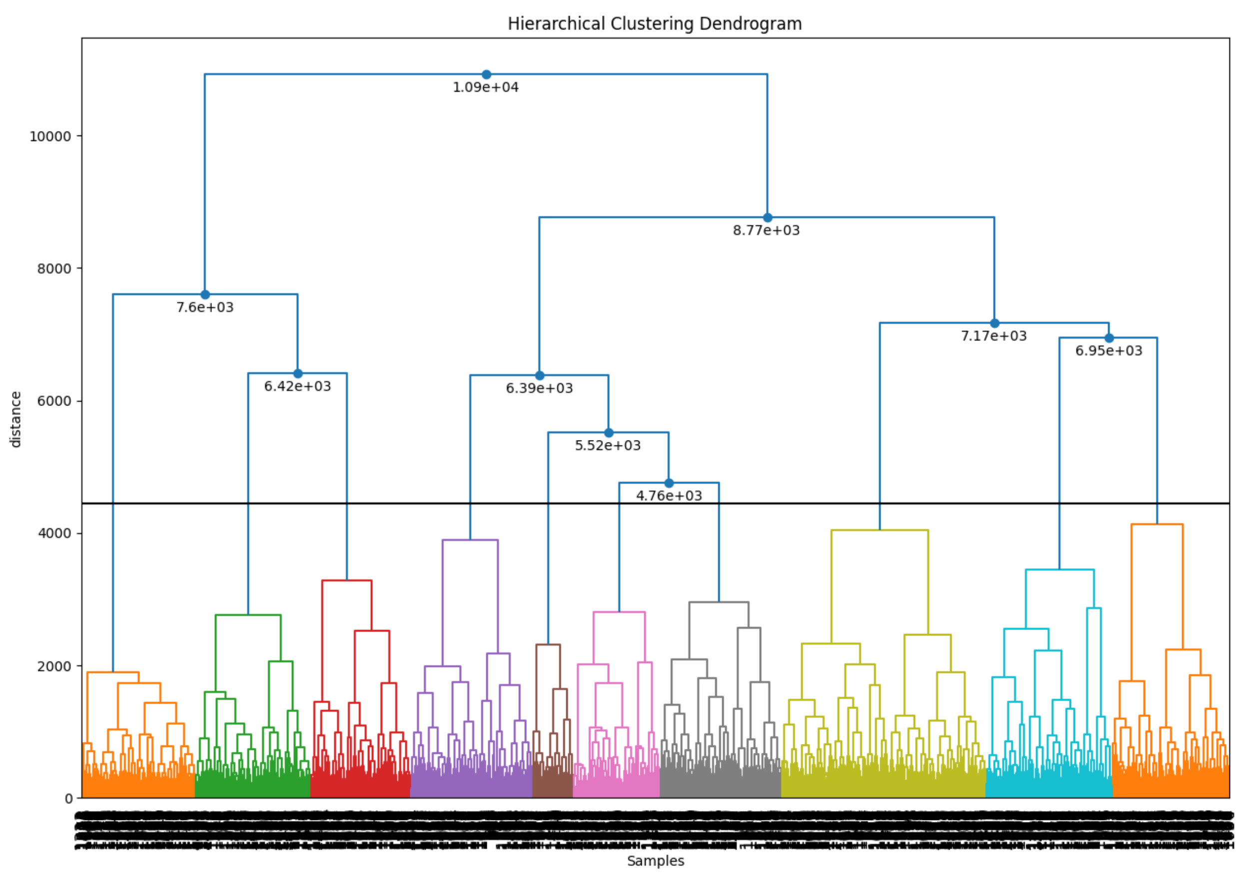


Q6 Implement agglomerative Hierarchical clustering.

Code:

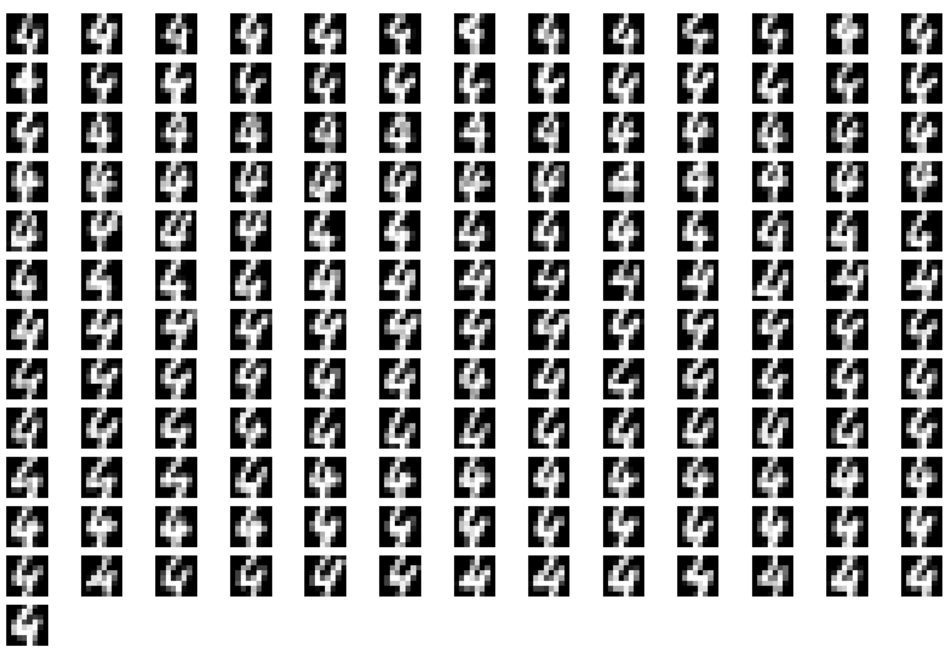
```
from clustimage import Clustimage

cl = Clustimage()

X = cl.import_example(data='mnist')

result = cl.fit_transform(X, cluster='agglomerative')

cl.dendrogram()

cl.plot(cmap='binary', labels=[1,2])
```

Output:



Images in cluster 1

Images in cluster 2



Q7 Implement Maximum-Likelihood estimation.

Code:

```
# %%
import numpy as np
import math
from scipy.optimize import minimize

# %%
mean = 10
std = 20

# %%
s = np.random.normal(mean, std, 3000)

# %%
def likelihood(mean, std, x):
    return (1 / math.sqrt(2 * math.pi * std**2)) * np.exp(-(x - mean)**2 /
(2 * std**2))

def log_likelihood(mean, std, data):
    return sum(np.log(likelihood(mean, std, x)) for x in data)

# %%
neg_log_likelihood = lambda mean: -log_likelihood(mean, std, s)

# %%
result = minimize(neg_log_likelihood, x0=0.0)
mean_mle = result.x[0]
print(mean_mle)

# %%
print("Difference between original mean and new mean", mean-mean_mle)

# %%
```

Output:

```
result = minimize(neg_log_likelihood, x0=0.0)
mean_mle = result.x[0]
print(mean_mle)
```

✓ 2.8s

```
9.437499921115245
```

```
print("Difference between original mean and new mean", mean-mean_mle)
```

✓ 0.1s

```
Difference between original mean and new mean 0.5625000788847547
```

Q8 Implement Principal component analysis and use it for unsupervised learning

Code:

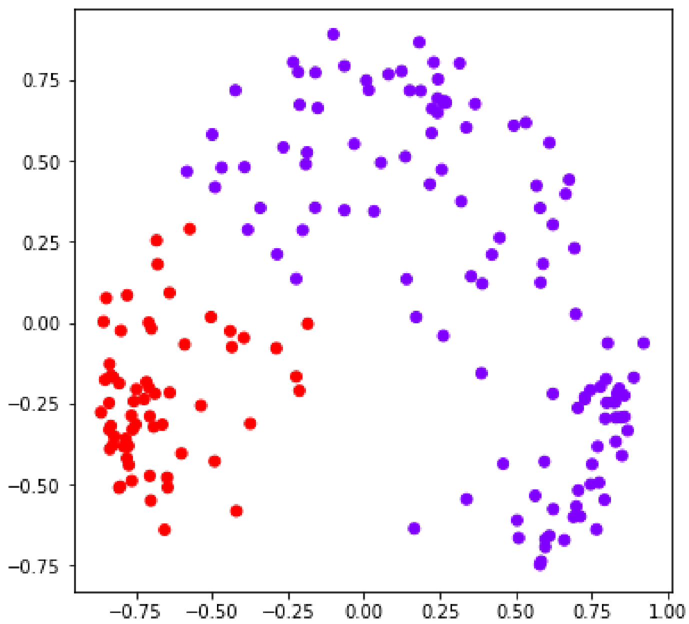
```
# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc

# %%
X = pd.read_csv('wine-clustering.csv')

X.fillna(method ='ffill', inplace = True)

# %%
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_normalized = normalize(X_scaled)
```

```python
X_normalized = pd.DataFrame(X_normalized)

# %%
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X_normalized)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']

# %%

plt.figure(figsize =(8, 8))
plt.title('Visualising the data')
Dendrogram = shc.dendrogram((shc.linkage(X_principal, method ='ward')))

# %%

ac2 = AgglomerativeClustering(n_clusters = 2)

# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
           c = ac2.fit_predict(X_principal), cmap ='rainbow')
plt.show()

# %%
ac3 = AgglomerativeClustering(n_clusters = 3)

plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
           c = ac3.fit_predict(X_principal), cmap ='rainbow')
plt.show()

# %%

ac4 = AgglomerativeClustering(n_clusters = 4)

plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
            c = ac4.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```

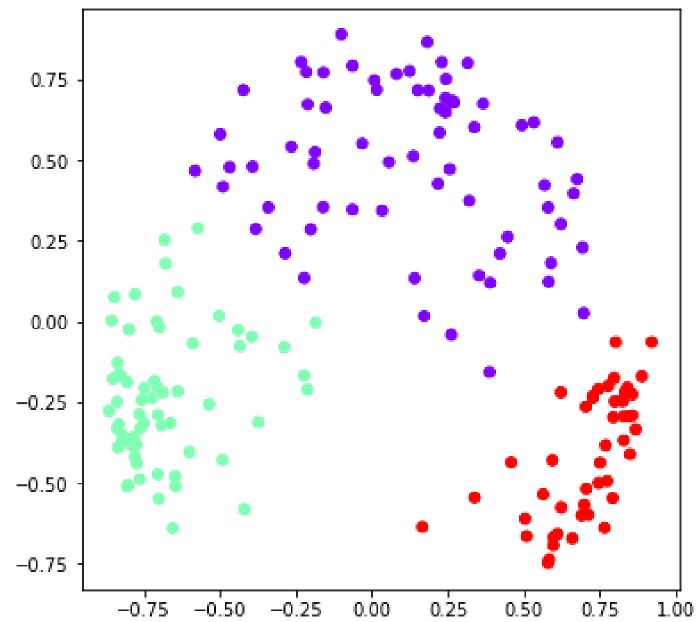
```python
# %%
ac5 = AgglomerativeClustering(n_clusters = 5)

plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
            c = ac5.fit_predict(X_principal), cmap ='rainbow')
plt.show()

# %%
ac6 = AgglomerativeClustering(n_clusters = 6)

plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
            c = ac6.fit_predict(X_principal), cmap ='rainbow')
plt.show()

# %%

k = [2, 3, 4, 5, 6]

# Appending the silhouette scores of the different models to the list
silhouette_scores = []
silhouette_scores.append(
        silhouette_score(X_principal, ac2.fit_predict(X_principal)))
silhouette_scores.append(
        silhouette_score(X_principal, ac3.fit_predict(X_principal)))
silhouette_scores.append(
        silhouette_score(X_principal, ac4.fit_predict(X_principal)))
silhouette_scores.append(
        silhouette_score(X_principal, ac5.fit_predict(X_principal)))
silhouette_scores.append(
        silhouette_score(X_principal, ac6.fit_predict(X_principal)))

# Plotting a bar graph to compare the results
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 20)
plt.ylabel('S(i)', fontsize = 20)
plt.show()
```

```
# %%
```

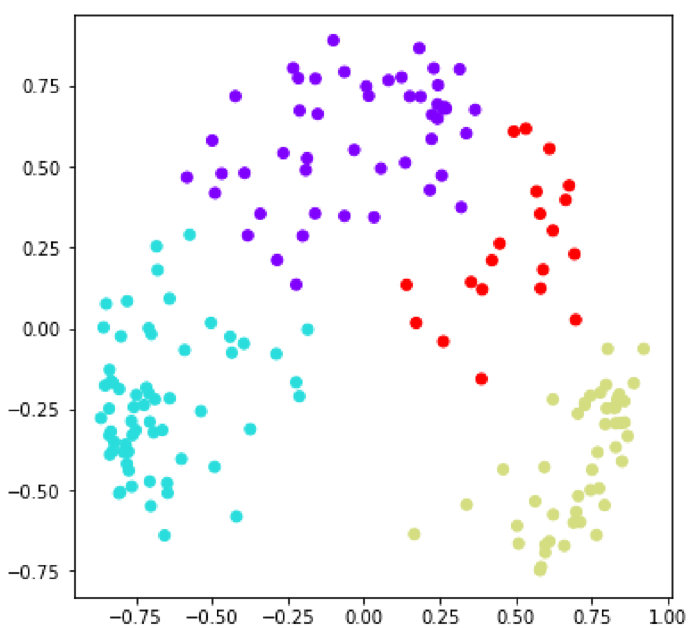
Output:



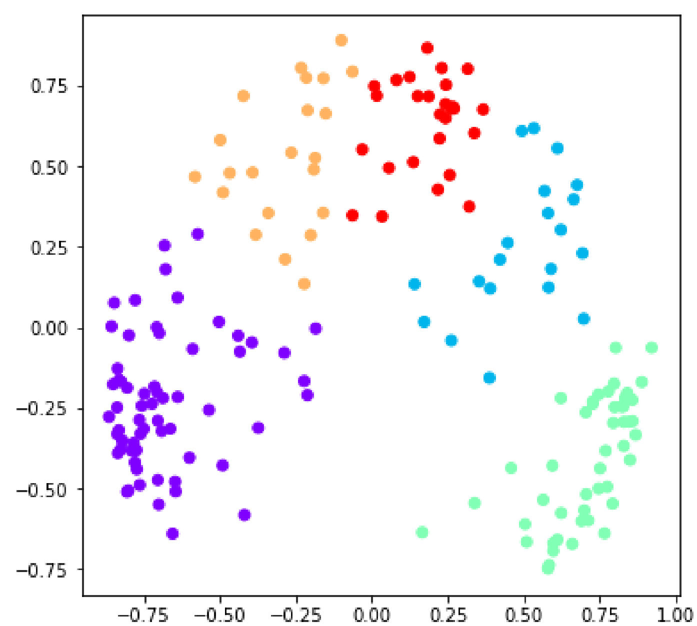
For n_clusters = 2



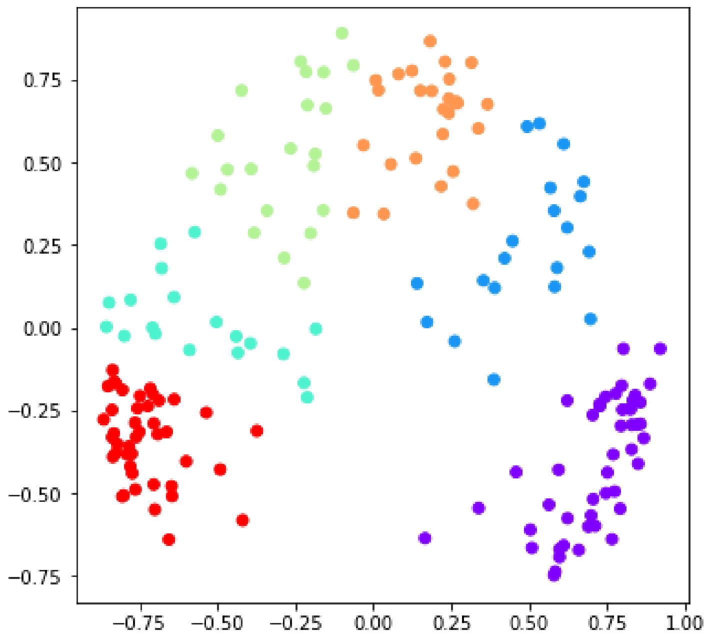
For n_clusters = 3



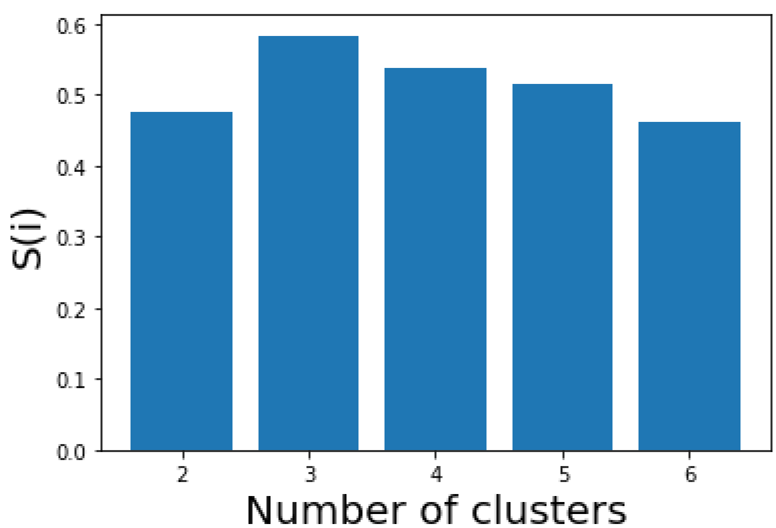
For n_clusters = 4



For n_clusters = 5



For n_clusters = 6



Plot of n_clusters and score