NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

Pattern Processing Using AI (COCSE60)
PRACTICAL FILE

Submitted by : -
AJIT KUMAR
2019UCO1700
COE 3
8th sem

INDEX

EXPERIMENT 1

Write a Python program to implement a chatbot

Code:

```
from chatterbot import ChatBot from
chatterbot.trainers import
ChatterBotCorpusTrainer
chatbot=ChatBot('corona bot')
trainer = ChatterBotCorpusTrainer(chatbot)
trainer.train("chatterbot.corpus.english.greetings",
"chatterbot.corpus.english.conversations" )
response = chatbot.get_response('What is your Number')print(response)
response = chatbot.get_response('Who are you?')print(response)
```

Output:

```
Training greetings.yml: [####################] 100%
Training conversations.yml: [#######             ] 33%

[nltk_data] Downloading package stopwords to /home/nikhil/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/nikhil/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!

Training conversations.yml: [####################] 100%
I don't have any number
I am just an artificial intelligence.
```

EXPERIMENT 2

Write a program to implement K-Means clustering from scratch

Code:

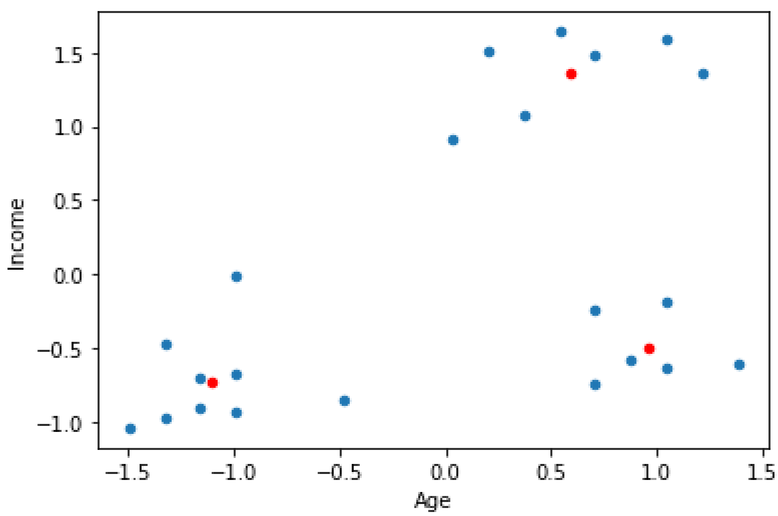
```
import pandas as pd import numpy
as np import matplotlib.pylab
import matplotlib.pyplot as plt
df = pd.read_excel(r"K Means Clustering.xlsx")X = df[["Age",
"Income"]]
ax = X.plot.scatter(x="Age", y="Income")X = (X -
X.mean()) / X.std()
def kMeansInitCentroids(X, K):
    randidx = np.random.permutation(len(X))centroids =
    X.iloc[randidx[0:K]]
    return centroids
def findClosestCentroids(X, centroids):K = len(centroids)
    idx = [0] * len(X)
    for i in range(len(X)): distance_array = [0] *
        Kfor j in range(K):
            distance_array[j] = np.sqrt(sum(pow(X.iloc[i] -centroids.iloc[j], 2)))
        idx[i] = np.argmin(distance_array)return idx
def computeMeans(X, idx, K, centroids):for k in range(K):
          points = [i for i, element in enumerate(idx) if element ==k]
        centroids.iloc[k] = X.iloc[points].mean()return
iterations = 10
def runKMeans(X, K):
     centroids = kMeansInitCentroids(X, K)for iter in
     range(iterations):
          idx = findClosestCentroids(X, centroids)computeMeans(X,
          idx, K, centroids)
```

```
        return centroids, idxinertiaList =
[]

for K in range(1, len(X) + 1): centroids, idx =
      runKMeans(X, K)inertia = 0
      for k in range(K):
                points = [i for i, element in enumerate(idx) if element ==k]
                inertia = (inertia + ((X.iloc[points] - centroids.iloc[k])
        ** 2).sum(axis=1).sum())
      inertiaList.append(inertia)
y = np.array(inertiaList)
x = np.array(range(1, len(X) + 1))plt.xticks(x)
plt.plot(x, y)plt.show()
K = 3
centroids, idx = runKMeans(X, K)
ax = X.plot.scatter(x="Age", y="Income") centroids.plot.scatter(x="Age", y="Income",
color="Red", ax=ax)
```

Output:

	Name	Age	Income
0	N1	27	105000
1	N2	29	135000
2	N3	29	91500
3	N4	28	90000
4	N5	42	225000
5	N6	39	232500
6	N7	41	240000
7	N8	38	243000
8	N9	36	234000
9	N10	35	195000
10	N11	37	205500
11	N12	26	67500
12	N13	27	72000
13	N14	28	76500
14	N15	29	74250
15	N16	32	79500
16	N17	40	97500
17	N18	41	94500
18	N19	43	96000
19	N20	39	120000
20	N21	41	123000
21	N22	39	87000



EXPERIMENT 3

Generating samples of Normal distribution and plotting them

Code:

import numpy as np
import matplotlib.pyplot as pltfrom scipy.stats
import norm import statistics

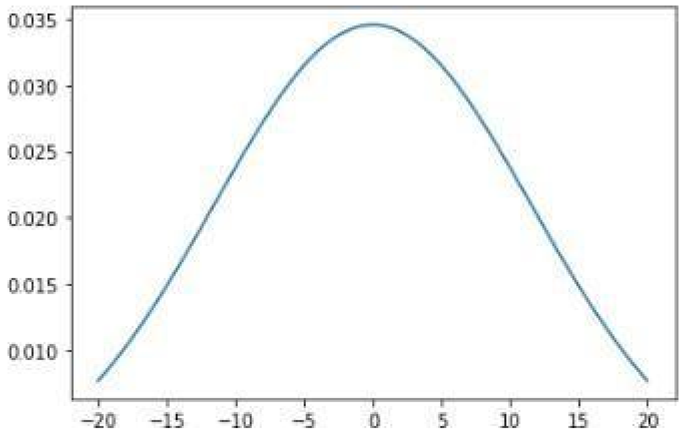
# Plot between -10 and 10 with .001 steps.
x_axis = np.arange(-20, 20, 0.01)

Calculating mean and standard deviation

```
mean = statistics.mean(x_axis)sd =
statistics.stdev(x_axis)

plt.plot(x_axis, norm.pdf(x_axis, mean, sd))plt.show()
```

Output:



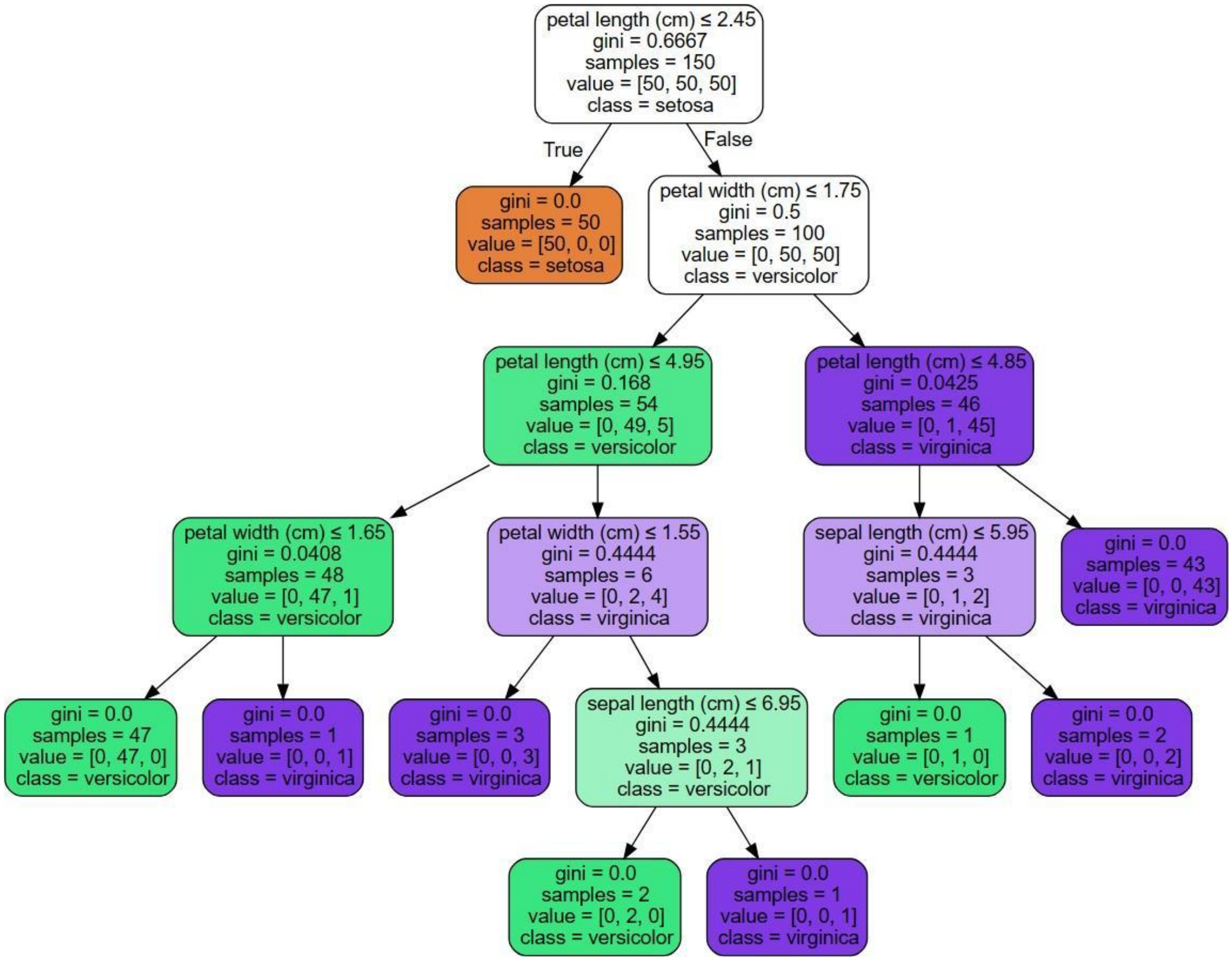
EXPERIMENT 4

Implement Decision Tree Algorithms

Code:

```
from sklearn.datasets import load_irisfrom sklearn import tree
iris = load_iris()
X, y = iris.data, iris.target
clf = tree.DecisionTreeClassifier()clf = clf.fit(X, y)
```

Output:



EXPERIMENT 5

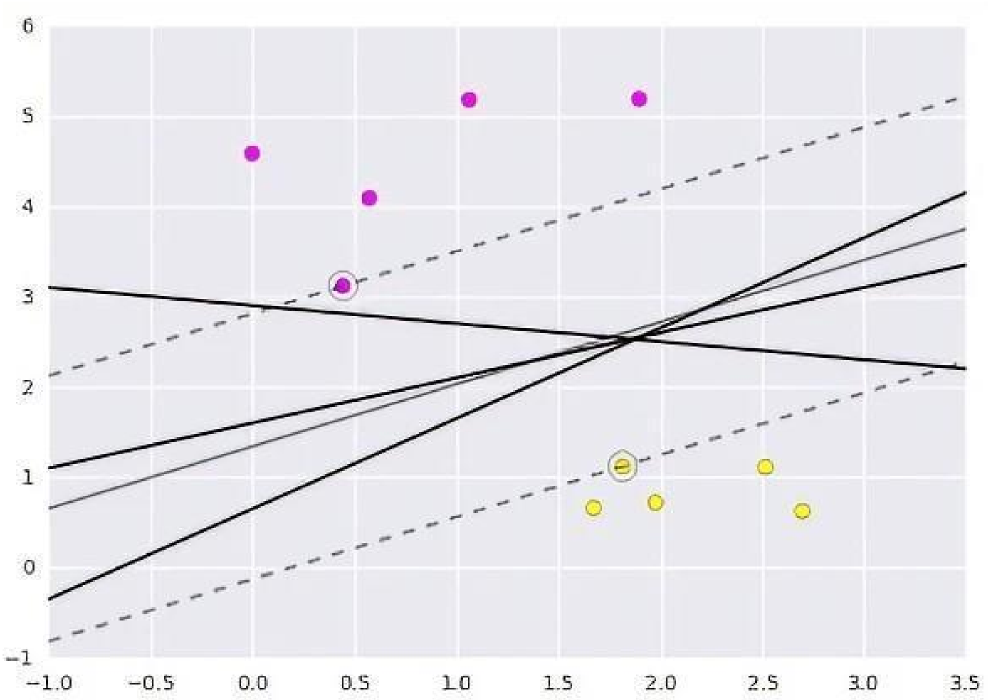
Implement SVM

Code:

```
import numpy as np import
pandas as pd
import matplotlib.pyplot as plt
x = pd.read_csv("C:\...\cancer.csv")a = np.array(x)
y = a[:,30]
```

```
x = np.column_stack((x.malignant,x.benign))from sklearn.svm
import SVC
clf = SVC(kernel='linear')clf.fit(x, y)
clf.predict([[120, 990]])
clf.predict([[85, 550]])
```

Output:



EXPERIMENT 6

Implement PCA and use it for unsupervised learning

Code:

```
import pandas as pdurl =
"https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
```

df = pd.read_csv(url, names=['sepal length','sepalwidth','petal length','petal width','target']) from sklearn.preprocessing import StandardScaler features = ['sepal length', 'sepal width', 'petal length', 'petal width'] x = df.loc[:, features].values y = df.loc[:,['target']].values

x = StandardScaler().fit_transform(x) from sklearn.decomposition import PCA pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x) principalDf = pd.DataFrame(data

= principalComponents
, columns = ['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, df[['target']]],axis = 1)

Output:

	sepal length	sepal width	petal length	petal width
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977

PCA
(2 components)

	principal component 1	princial component 2
0	-2.264542	0.505704
1	-2.086426	-0.655405
2	-2.367950	-0.318477
3	-2.304197	-0.575368
4	-2.388777	0.674767

EXPERIMENT 7

Implement Maximum Likelihood estimation

Code:

import numpy as np
from scipy.optimize import minimize
np.random.seed(123)

```
mu_true = 2
sigma_true = 1.5
data = np.random.normal(mu_true, sigma_true, 100)def
normal_likelihood(params, data):
        mu, sigma = params
        ll = -np.sum(np.log(sigma) + 0.5 * np.log(2 *np.pi) + ((data - mu)
** 2) / (2 * sigma ** 2))
        return ll
def neg_log_likelihood(params,   data): return -
        normal_likelihood(params, data)
params_init = [1, 1] # Initial guess for mu and sigmaresults =
minimize(neg_log_likelihood, params_init, args=(data,))
mu_hat, sigma_hat = results.x print(f"Estimated mu:
{mu_hat:.2f}") print(f"Estimated sigma: {sigma_hat:.2f}")
```

Output:

```
Estimated mu: 1.95
Estimated sigma: 1.44
```

EXPERIMENT 8

Implement agglomerative hierarchical clustering

Code:

```
import pandas as pdimport numpy
as np
```

```
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClusteringimport
scipy.cluster.hierarchy as sch
dataset = pd.read_csv('./data.csv')
X = dataset.iloc[:, [3, 4]].values
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))
model = AgglomerativeClustering(n_clusters=5, affinity='euclidean',linkage='ward')
model.fit(X)
labels = model.labels_
plt.scatter(X[labels==0, 0], X[labels==0, 1], s=50, marker='o',color='red')
plt.scatter(X[labels==1, 0], X[labels==1, 1], s=50, marker='o',color='blue')
plt.scatter(X[labels==2, 0], X[labels==2, 1], s=50, marker='o',color='green')
plt.scatter(X[labels==3, 0], X[labels==3, 1], s=50, marker='o',color='purple')
plt.scatter(X[labels==4, 0], X[labels==4, 1], s=50, marker='o',color='orange') plt.show()
```

Output: