Ex:1

Date:

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Aim: To demonstrate the FIND-S algorithm for finding the most specific hypothesis for sample data set.

Procedure:

Step 1: Import necessary Python libraries

Step 2: Read csv file using pandas library

Step 3: Segregate positive and negative example

Step 4: Identify the hypothesis from the dataset by comparing all the positive example

Step 5 : Print the hypothesis

Python Code:

```
import pandas as pd
import numpy as np

#to read the data in the csv file
data = pd.read_csv("finds.csv")
print(data,"n")

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)

#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
```

```
print("n The target is: ",target)

#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis
 #obtaining the final hypothesis
print("n The final hypothesis is:",train(d,target))
```

Sample Data:

Time	Weather	Temperature	Company	Humidity	Wind	Goes
Morning	Sunny	Warm	Yes	Mild	Strong	Yes
Evening	Rainy	Cold	No	Mild	Normal	No
Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
Evening	Sunny	Cold	Yes	High	Strong	Yes

Sample Output:

Time Weather Temperature Company Humidity Wind Goes

0 Morning Sunny Warm Yes Mild Strong Yes

1 Evening Rainy Cold No Mild Normal No

2 Morning Sunny Moderate Yes Normal Normal Yes

3 Evening Sunny Cold Yes High Strong Yes n

n The attributes are: [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']

['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']

['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']

['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]

The target is: ['Yes' 'No' 'Yes' 'Yes']

The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']

RESULT:

Thus the program to FIND-S algorithm for finding the most specific hypothesis using a sample data set is verified and executed successfully.

Ex:2

Date:

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Aim: To implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses.

Procedure:

Step 1: Import necessary Python libraries

Step 2: Read dataset in CSV format

Step 3: Initialize General Hypothesis and Specific Hypothesis.

Step4: For each training example
Step5: If example is positive example
```
if attribute_value == hypothesis_value:
        Do nothing
else:
        replace attribute value with '?' (Basically generalizing it)
```
Step6: If example is Negative example
Make generalize hypothesis more specific.

Python Code:

```
import numpy as np

import pandas as pd


data = pd.read_csv('enjoysport.csv')

concepts = np.array(data.iloc[:,0:-1])

print(concepts)

target = np.array(data.iloc[:,-1])

print(target)

def learn(concepts, target):
```

```
specific_h = concepts[0].copy()
print("initialization of specific_h and general_h")
print(specific_h)
general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
print(general_h)

for i, h in enumerate(concepts):
    print("For Loop Starts")
    if target[i] == "yes":
        print("If instance is Positive ")
        for x in range(len(specific_h)):
            if h[x]!= specific_h[x]:
                specific_h[x] ='?'
                general_h[x][x] ='?'

    if target[i] == "no":
        print("If instance is Negative ")
        for x in range(len(specific_h)):
            if h[x]!= specific_h[x]:
                general_h[x][x] = specific_h[x]
            else:
                general_h[x][x] = '?'

    print(" steps of Candidate Elimination Algorithm",i+1)
    print(specific_h)
    print(general_h)
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
for i in indices:
    general_h.remove(['?', '?', '?', '?', '?', '?'])
return specific_h, general_h
```

```
s_final, g_final = learn(concepts, target)


print("Final Specific_h:", s_final, sep="\n")

print("Final General_h:", g_final, sep="\n")
```

Sample Data:

enjoysport.csv

sky	airtemp	humidity	wind	water	forcast	enjoysport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Sample Output:

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 2
```

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

For Loop Starts

If instance is Negative

steps of Candidate Elimination Algorithm 3

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

For Loop Starts

If instance is Positive

steps of Candidate Elimination Algorithm 4

['sunny' 'warm' '?' 'strong' '?' '?']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:

['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

RESULT:

Thus the program for candidate-Elimination algorithm using the set of all hypotheses is verified and executed successfully.

Ex:3

Date:

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Aim: To demonstrate the working of the decision tree based ID3 algorithm.

Procedure:

Step 1: Import necessary Python libraries

Step 2: Download dataset from machine learning library

Step 3: Calculate entropy for dataset.

Step 4: For each attribute/feature.

4.1. Calculate entropy for all its categorical values.

4.2. Calculate information gain for the feature.

Step 5: Find the feature with maximum information gain.

Step 6: Repeat it until we get the desired tree.

Python Code:

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
#from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
```

```
# Function importing Dataset
def importdata():
	balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
	sep= ',', header = None)

	# Printing the dataswet shape
	print ("Dataset Length: ", len(balance_data))
	print ("Dataset Shape: ", balance_data.shape)

	# Printing the dataset obseravtions
	print ("Dataset: ",balance_data.head())
	return balance_data

# Function to split the dataset
def splitdataset(balance_data):

	# Separating the target variable
	X = balance_data.values[:, 1:5]
	Y = balance_data.values[:, 0]

	# Splitting the dataset into train and test
	X_train, X_test, y_train, y_test = train_test_split(
	X, Y, test_size = 0.3, random_state = 100)

	return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
```

```
def train_using_gini(X_train, X_test, y_train):

        # Creating the classifier object
        clf_gini = DecisionTreeClassifier(criterion = "gini",
                        random_state = 100,max_depth=3, min_samples_leaf=5)

        # Performing training
        clf_gini.fit(X_train, y_train)
        return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

        # Decision tree with entropy
        clf_entropy = DecisionTreeClassifier(
                        criterion = "entropy", random_state = 100,
                        max_depth = 3, min_samples_leaf = 5)

        # Performing training
        clf_entropy.fit(X_train, y_train)
        return clf_entropy



# Function to make predictions
def prediction(X_test, clf_object):

        # Predicton on test with giniIndex
        y_pred = clf_object.predict(X_test)
        print("Predicted values:")
        print(y_pred)
```

```
        return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

        print("Confusion Matrix: ",
                confusion_matrix(y_test, y_pred))

        print ("Accuracy : ",
        accuracy_score(y_test,y_pred)*100)

        print("Report : ",
        classification_report(y_test, y_pred))

# Driver code
def main():

        # Building Phase
        data = importdata()
        X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
        clf_gini = train_using_gini(X_train, X_test, y_train)
        clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

        # Operational Phase
        print("Results Using Gini Index:")

        # Prediction using gini
        y_pred_gini = prediction(X_test, clf_gini)
        cal_accuracy(y_test, y_pred_gini)
```

```
        print("Results Using Entropy:")
        # Prediction using entropy
        y_pred_entropy = prediction(X_test, clf_entropy)
        cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if __name__=="__main__":
        main()
```

Sample Data:

https://archive.ics.uci.edu/ml/machine-learning-'+'databases/balance-scale/balance-scale.data

Sample Output:

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
Results Using Gini Index:
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
```

'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[0 6 7]
[0 67 18]
[0 19 71]]

Accuracy : 73.40425531914893

Report : precision recall f1-score support

	precision	recall	f1-score	support
B	0.00	0.00	0.00	13
L	0.73	0.79	0.76	85
R	0.74	0.79	0.76	90
accuracy			0.73	188
macro avg	0.49	0.53	0.51	188
weighted avg	0.68	0.73	0.71	188

Results Using Entropy:

Predicted values:

['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'

```
'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy :  70.74468085106383

Report :               precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.71      0.74      0.72        85
           R       0.71      0.78      0.74        90

    accuracy                           0.71       188
   macro avg       0.47      0.51      0.49       188
weighted avg       0.66      0.71      0.68       188
```

RESULT:

Thus the program of the decision tree based ID3 algorithm is verified and executed successfully.

Ex:4

Date:

Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

.**Aim:** To Build an Artificial Neural Network by implementing the Back propagation algorithm.

Procedure:

Step 1: Import necessary Python libraries

Step 2: Initialize Network input, weight

Step 3: Forward Propagate.

Step 4: Back Propagate Error.

Step 5: Train Network.

Step 6: Predict.

Python Code:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # two inputs [sleep,study]
y = np.array(([92], [86], [89]), dtype=float) # one output [Expected % in Exams]
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100
#Sigmoid Function
def sigmoid (x):
   return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
   return x * (1 - x)

#Variable initialization
epoch=5000    #Setting training iterations
lr=0.1        #Setting learning rate
inputlayer_neurons = 2          #number of features in data set
hiddenlayer_neurons = 3    #number of hidden layers neurons
output_neurons = 1         #number of neurons at output layer
#weight and bias initialization
```

```
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons)) #weight of the link
from input node to hidden node
bh=np.random.uniform(size=(1,hiddenlayer_neurons)) # bias of the link from input node to
hidden node
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons)) #weight of the link
from hidden node to output node
bout=np.random.uniform(size=(1,output_neurons)) #bias of the link from hidden node to
output node
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):

#Forward Propogation
   hinp1=np.dot(X,wh)
   hinp=hinp1 + bh
   hlayer_act = sigmoid(hinp)
   outinp1=np.dot(hlayer_act,wout)
   outinp= outinp1+ bout
   output = sigmoid(outinp)

#Backpropagation
   EO = y-output
   outgrad = derivatives_sigmoid(output)
   d_output = EO* outgrad
   EH = d_output.dot(wout.T)

#how much hidden layer weights contributed to error
   hiddengrad = derivatives_sigmoid(hlayer_act)
   d_hiddenlayer = EH * hiddengrad

# dotproduct of nextlayererror and currentlayerop
wout += hlayer_act.T.dot(d_output) *lr
wh += X.T.dot(d_hiddenlayer) *lr

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Sample Data:
[2, 9], [1, 5], [3, 6] (Sleep, study)
[92], [86], [89] (Expected output)
Sample Output:

```
       Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
```

```
 [0.89]]
Predicted Output:
 [[0.87805512]
 [0.85929315]
 [0.87512372]]
```

RESULT:

Thus the program to Build an Artificial Neural Network by implementing the Back propagation algorithm is verified and executed successfully.

Ex:5

Date:

Write a program t implement the Naive Bayesian classifier for a sample training data set stored as a CSV file. Compute the accuracy of the classifier, considering few data set.

Aim: To compute accuracy of Naive Bayesian classifier for sample data set.

Procedure:

Step 1: Import necessary Python libraries

Step 2: Download dataset from sklearn

Step 3: Build Gaussian model for Naïve Bayesian classifier

Step 4: Assign data to build model

Step 5 : Print the metrics for classifier using metric library

Python Code:

```
from sklearn import datasets
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
dataset = datasets.load_iris()
model = GaussianNB()
model.fit(dataset.data, dataset.target)
expected = dataset.target
predicted = model.predict(dataset.data)
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

Sample Data:

Iris Dataset

Sample Output:

	precision	recall	f1-score	support
0	**1.00**	**1.00**	**1.00**	**50**
1	**0.94**	**0.94**	**0.94**	**50**
2	**0.94**	**0.94**	**0.94**	**50**
accuracy			**0.96**	**150**
macro avg	**0.96**	**0.96**	**0.96**	**150**
weighted avg	**0.96**	**0.96**	**0.96**	**150**

RESULT:

Thus the program to compute accuracy of Native Bayesian Classifier for sample data set is verified and executed successfully.

Ex:6
Date:

Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set (You can use Java/Python ML library classes/API)..

Aim: To construct a Bayesian network considering medical data.

Procedure:

Step 1: Import necessary Python libraries

Step 2: Download dataset from sklearn

Step 3: Build Bayesian network model for medical data

Step 4: Use the model to diagnosis heart patients data

Step 5 : Print the result

Python Code:

```
import numpy as np
from urllib.request import urlopen
import urllib
import matplotlib.pyplot as plt # Visuals
import seaborn as sns
import sklearn as skl
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
Cleveland_data_URL =
'http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.hun
garian.data'
names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak',
'slope', 'ca', 'thal', 'heartdisease']
heartDisease = pd.read_csv(urlopen(Cleveland_data_URL), names = names)
heartDisease = heartDisease.replace('?', np.nan)
#display the data
print('Few examples from the dataset are given below')

print(heartDisease.head())
```

```
model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),
                        ('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),
                       ('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])

# Learing CPDs using Maximum Likelihood Estimators
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)# Learing CPDs using Maximum Likelihood Estimators

from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)

# Computing the probability of bronc given smoke.
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q['heartdisease'])
```

Sample Data:

Heart Disease Dataset

Sample Output:

Few examples from the dataset are given below

	age	sex	cp	trestbps	chol	fbs	...	exang	oldpeak	slope	ca	thal	heartdisease
0	**28**	**1**	**2**	**130**	**132**	**0**	**...**	**0**	**0.0**	**NaN**	**NaN**	**NaN**	**0**
1	**29**	**1**	**2**	**120**	**243**	**0**	**...**	**0**	**0.0**	**NaN**	**NaN**	**NaN**	**0**
2	**29**	**1**	**2**	**140**	**NaN**	**0**	**...**	**0**	**0.0**	**NaN**	**NaN**	**NaN**	**0**
3	**30**	**0**	**1**	**170**	**237**	**0**	**...**	**0**	**0.0**	**NaN**	**NaN**	**6**	**0**
4	**31**	**0**	**2**	**100**	**219**	**0**	**...**	**0**	**0.0**	**NaN**	**NaN**	**NaN**	**0**

[5 rows x 14 columns]

heartdisease	phi(heartdisease)
heartdisease_0	0.6333
heartdisease_1	0.3667

RESULT:

Thus the program to construct a Bayesian network considering medical data is verified and executed successfully.

Ex:7

Date:

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering (You can add Java/Python ML library classes/API in the program).

Aim: To apply EM model and K-means for clustering a set of data stored in a .CSV file.

Procedure:

Step 1: Import necessary Python libraries

Step 2: Download dataset from sklearn

Step 3: Apply EM model for clustering

Step 4: Apply K-means algorithm for clustering

Step 5 : Compare the accuracy metrics for both the algorithm

Python Code:

```
# k-means clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from matplotlib import pyplot
# define dataset
X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2,
n_redundant=0, n_clusters_per_class=1, random_state=4)
# define the model
model = KMeans(n_clusters=2)
# fit the model
model.fit(X)
```

```
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
	# get row indexes for samples with this cluster
	row_ix = where(yhat == cluster)
	# create scatter of these samples
	pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```

 # gaussian mixture clustering

```
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from matplotlib import pyplot

# define dataset
X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=4)
# define the model
model = GaussianMixture(n_components=2)
# fit the model
model.fit(X)
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
```

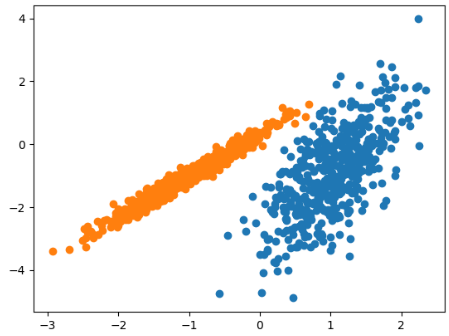
```
# create scatter plot for samples from each cluster
for cluster in clusters:
	# get row indexes for samples with this cluster
	row_ix = where(yhat == cluster)
	# create scatter of these samples
	pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```

Sample Data:

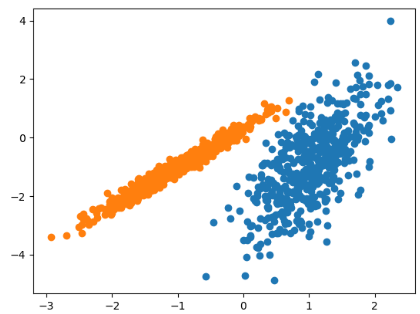
Random data generation

Sample Output:

EM- Gaussian Mixture



K-Means



Analysis:

The K-means algorithm does is just to follow a recipe: alternate between computing the means of each of the K classes (centers of gravity) and assigning each point to the nearest mean. The outcome is such that only points which are close together are deemed to be in the same class.

A Gaussian mixture model, on the other hand, assumes that for each datapoint x_n there is a latent (hidden) variable z_n with values 1, ..., K representing its cluster (or class). Conditional on z_n, x_n is drawn from a Gaussian distribution with mean and co-variance matrix depending on the class z_n. The EM algorithm attempts to find the configuration of the z_n's that maximizes the overall likelihood.

In your example, you're generating data from a mixture of two distributions: orange and blue, so K=2. Orange and blue are not strictly Gaussian, but close enough. Accordingly, in both cases (even with only few data) the mixture model picks up that there is one distribution with a bigger variance (yellow) and one with a smaller variance (purple). By design, K-means has no chance of picking up this pattern.

RESULT:

Thus the program to apply EM model and K-means for clustering a set of data stored in a.CSV file is verified and executed successfully.

Ex:8

Date:

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem

Aim: To implement k-Nearest Neighbour algorithm to classify the iris data set.

Procedure:

Step 1: Import necessary Python libraries

Step 2: Download dataset from sklearn

Step 3: Build k-Nearest Neighbour classifier

Step 4: Print the predictions using metric library

Python Code:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=42)
```

```
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
print(knn.predict(X_test))
```

Sample Data:

Iris Dataset

Sample Output:

[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]

RESULT:

Thus the program to implement the K-Nearest Neighbour algorithm to classify the iris data set is verified and executed successfully.