**TO IMPLEMENT AND DEMONSTRATE THE FIND-S ALGORITHM**

**Aim:**

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.

**Algorithm:**

**process starts with initializing '' with the Most specific hypothesis, generally, it is the first positive example in the data set. We check for each positive example. If the exemple is negative, we will move on to the next example but if It is a positive example we will consider it for the next step. We will check if each attribute in the example is equal to the hypothesis value.**

**If the value matches, then charge are made**

**If the value does not match, the values is changed to ? - are made.**

**we do this until we search the last positive example in the data set.**

**CODE:**

**import pandas as pd**

**import numpy as np**

**data = pd.read\_csv("D:aj.csv")**

**print(data,"\n")**

**d = np.array(data)[:,:-1]**

**print("\nThe attributes are: ",d)**

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

**print("\nThe target is: ",target)**

**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**

**print("\nThe final hypothesis is:",train(d,target))**

**OUTPUT:**

The attributes are: [[nan 'Sunny ' 'Warm ' 'Normal ' 'Strong ' 'Warm ' 'Same ']

[nan 'Sunny ' 'Warm ' 'High ' 'Strong ' 'Warm ' 'Same ']

[nan 'Rainy ' 'Cold ' 'High ' 'Strong ' 'Warm ' 'Change ']

[nan 'Sunny ' 'Warm ' 'High ' 'Strong ' 'Cool ' 'Change ']]

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

The final hypothesis is: ['?' 'Sunny ' 'Warm ' '?' 'Strong ' '?' '?']

2. **CANDIDATE\_ELIMINATION ALGORITHM**

**Aim:**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.

**Algorithm:**

**Step 1 For each training example d. do. If dis positive example**

**Step 2: remove for g any hypothesis h inconsistent with d**

**Step 3. for each hypothesis is Snot consistent Goal R...**

**Step 4 Remove &from 8.**

**Step 5. Add to S all minimal generalization of consistent with d and**

**having a generalization in G.**

**Step 6. Remove from s any hypothesis with a more specific h in s.**

**Step 7. In d is negative example Remove from 8. my hypotheses**

**h in consistent with d.**

**Stop 8. Remove g from G.**

**Stop 9. Add to G all minimal specialization of g consistent with d and having a specialization is s.**

**Step 10. Remove from G any hypothers having generalhypothesis is G.**

**CODE:**

import numpy as np

import pandas as pd

data = pd.read\_csv("D:\AI LAB\sport new.csv")

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

print("\nInstances are:\n",concepts)

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

print("\nTarget Values are: ",target)

def learn(concepts, target):

specific\_h = concepts[0].copy()

print("\nInitialization of specific\_h and genearal\_h")

print("\nSpecific Boundary: ", specific\_h)

general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

print("\nGeneric Boundary: ",general\_h)

for i, h in enumerate(concepts):

print("\nInstance", i+1 , "is ", h)

if target[i] == "yes":

print("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("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("Specific Bundary after ", i+1, "Instance is ", specific\_h)

print("Generic Boundary after ", i+1, "Instance is ", general\_h)

print("\n")

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")

**OUTPUT:**

Instances are:

[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

['sunny' 'warm' 'high' 'strong' 'warm' 'same']

['rainy' 'cold' 'high' 'strong' 'warm' 'change']

['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values are: ['yes' 'yes' 'no' 'yes']

Initialization of specific\_h and genearal\_h

Specific Boundary: ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Instance is Positive

Specific Bundary after 1 Instance is ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is ['sunny' 'warm' 'high' 'strong' 'warm' 'same']

Instance is Positive

Specific Bundary after 2 Instance is ['sunny' 'warm' '?' 'strong' 'warm' 'same']

Generic Boundary after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 3 is ['rainy' 'cold' 'high' 'strong' 'warm' 'change']

Instance is Negative

Specific Bundary after 3 Instance is ['sunny' 'warm' '?' 'strong' 'warm' 'same']

Generic Boundary after 3 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

Instance 4 is ['sunny' 'warm' 'high' 'strong' 'cool' 'change']

Instance is Positive

Specific Bundary after 4 Instance is ['sunny' 'warm' '?' 'strong' '?' '?']

Generic Boundary after 4 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h:

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

Final General\_h:

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

3. **DECISION TREE BASED I3 ALGORITHM**

**Aim:**

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.

**ALGORITHM:**

Step 1: Declare the target - attribute is the attribute whose value is to be predicted by the tree, Attributes in a list of other attributes that mауbetested. by the Learned decision tree example attribute are returned. by a decision tree that correctly classifies the given example.

Step 2: execute the for ID3(Example, Target\_attribute, Attribute)

Step 3: create a Root node for the dree if all example aro Positive Action the single-node bree Roob, with label =+ if all excample are negative, Return the Single-node tree Root, with label =- If athibutes is empty, return the Single -node bree Root, with label = most common value of Torget-attribute in example.

Step4: otherwise begin

At the attribute from Attalbute that best "classifies example

The decision adtribute for Root &-A for each possiblevalues vi, of A,

Add a new tree branch below Root, corresponding to the test A=vi

Let excaples vis be the subst of Examples that have value vi for A.

If excaples vi, is exepty.

Then below this. branch add a Leaf node with label= Most common vale of Parget - attribute in Examples.

else

below this new branch add the Subtree ID3 (Examples vi,Target attribute, Attributes - {A}))

END

Stop 5 Return Root

**Code:**

**import pandas as pd**

**import math**

**import numpy as np**

**data = pd.read\_csv("D:LAB\exp3.csv")**

**features = [feat for feat in data]**

**features.remove("answer")**

**class Node:**

**def \_\_init\_\_(self):**

**self.children = []**

**self.value = ""**

**self.isLeaf = False**

**self.pred = ""**

**def entropy(examples):**

**pos = 0.0**

**neg = 0.0**

**for \_, row in examples.iterrows():**

**if row["answer"] == "yes":**

**pos += 1**

**else:**

**neg += 1**

**if pos == 0.0 or neg == 0.0:**

**return 0.0**

**else:**

**p = pos / (pos + neg)**

**n = neg / (pos + neg)**

**return -(p \* math.log(p, 2) + n \* math.log(n, 2))**

**def info\_gain(examples, attr):**

**uniq = np.unique(examples[attr])**

**#print ("\n",uniq)**

**gain = entropy(examples)**

**#print ("\n",gain)**

**for u in uniq:**

**subdata = examples[examples[attr] == u]**

**#print ("\n",subdata)**

**sub\_e = entropy(subdata)**

**gain -= (float(len(subdata)) / float(len(examples))) \* sub\_e**

**#print ("\n",gain)**

**return gain**

**def ID3(examples, attrs):**

**root = Node()**

**max\_gain = 0**

**max\_feat = ""**

**for feature in attrs:**

**#print ("\n",examples)**

**gain = info\_gain(examples, feature)**

**if gain > max\_gain:**

**max\_gain = gain**

**max\_feat = feature**

**root.value = max\_feat**

**#print ("\nMax feature attr",max\_feat)**

**uniq = np.unique(examples[max\_feat])**

**#print ("\n",uniq)**

**for u in uniq:**

**#print ("\n",u)**

**subdata = examples[examples[max\_feat] == u]**

**#print ("\n",subdata)**

**if entropy(subdata) == 0.0:**

**newNode = Node()**

**newNode.isLeaf = True**

**newNode.value = u**

**newNode.pred = np.unique(subdata["answer"])**

**root.children.append(newNode)**

**else:**

**dummyNode = Node()**

**dummyNode.value = u**

**new\_attrs = attrs.copy()**

**new\_attrs.remove(max\_feat)**

**child = ID3(subdata, new\_attrs)**

**dummyNode.children.append(child)**

**root.children.append(dummyNode)**

**return root**

**def printTree(root: Node, depth=0):**

**for i in range(depth):**

**print("\t", end="")**

**print(root.value, end="")**

**if root.isLeaf:**

**print(" -> ", root.pred)**

**print()**

**for child in root.children:**

**printTree(child, depth + 1)**

**root = ID3(data, features)**

**printTree(root)OUTPUT:**

outlook

overcast -> ['yes']

rain

wind

strong -> ['no']

weak -> ['yes']

sunny

humidity

high -> ['no']

normal -> ['yes']

**4. Back propagation algorithm**

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

**ALGORITHM:**

**1..create a feed forward network n2 input h hidden units. n output units**

**2.initialize all network device to small random units**

**3.for each training sample do propogate the input forward through the network input x to the network and compute output of every unit is network**

**4.propogate the error backward through te network unit k calculate its error sk.**

**sk<-->Ok(subscript)(1-ok)(k-ok)**

**for each network unit h, calculate the error sk**

**sk<--ok(1-on)Ewk,ksk**

**5.update each network weight wji**

**Wij<--- Wji + Wji**

**where**

**Wji = nfixji**

**CODE:**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0)

y = y/100

def sigmoid (x):

return 1/(1 + np.exp(-x))

def derivatives\_sigmoid(x):

return x \* (1 - x)

epoch=5

lr=0.1

inputlayer\_neurons = 2

hiddenlayer\_neurons = 3

output\_neurons = 1

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

for i in range(epoch):

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+bout

output = sigmoid(outinp)

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr

wh += X.T.dot(d\_hiddenlayer) \*lr

print ("-----------Epoch-", i+1, "Starts----------")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

RESULT:

-----------Epoch- 5 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.74602983]

[0.73586461]

[0.74682761]]

-----------Epoch- 5 Ends----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.74602983]

[0.73586461]

[0.74682761]]

​

5 **NAÏVE BAYESIANCLASSIFICATION**

**Aim:** Write a program to implement the Naive Bayesian Classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

**ALGORTIHM:**

Step 1: convert the dataset into a frequency table.

Step 2: Compute the "prior probabilities for each the class

Step 3: compute the probabilities of evidence.

Step4: Compute the probability of like li hood of evidence Stups Substitute all the vales into the Name Bayer.

Formula to get the probability

p(4/₂c) = P(x/y) & p(y) PO

Step6: The class with the highest posterior probability is the outcome of prediction.

**CODE:**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**import seaborn as sns**

**dataset = pd.read\_csv("D:\LAB\exp6.csv")**

**X = dataset.iloc[:, [0,1]].values**

**y = dataset.iloc[:, 2].values**

**from sklearn.model\_selection import train\_test\_split**

**X\_train, X\_test, y\_train, y\_test =train\_test\_split(X,y,test\_size= 0.25, random\_state=0)**

**from sklearn.preprocessing import StandardScaler**

**sc\_X = StandardScaler()**

**X\_train = sc\_X.fit\_transform(X\_train)**

**X\_test = sc\_X.fit\_transform(X\_test)**

**from sklearn.naive\_bayes import BernoulliNB**

**from sklearn.naive\_bayes import GaussianNB**

**classifer1 = GaussianNB()**

**classifer1.fit(X\_train, y\_train)**

**y\_pred1 = classifer1.predict(X\_test)**

**from sklearn.metrics import accuracy\_score**

**print(accuracy\_score(y\_test,y\_pred1))**

**OUTPUT**

0.91

6 **NAÏVE BAYESIANCLASSIFIER**

**Aim:**

By assuming a set of documents that need to be classified, use the naive Bayesian classifier model to perform this task. Built in java classes / API can be used to write the program. Calculate the accuracy, precision and recall for your data set.

**ALGORTIHM:**

**1**. convert the dataset into frequency table

2.complete the prior probabilities for each the classes

3.compute the probability of evidence

4.compute the probability of livelihood of evidence

5.substitute all the values into the naive bayes formula

p(xy/x)=p(x/y)/p(x)\*p(y)

6.the class with the highest posterior probability is the outcome of prediction

7.compute the accuracy for the given dataset

accuracy=TN+IP/TN+FP+TP+FN

**CODE:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

dataset = pd.read\_csv("D:\LAB\exp6.csv")

X = dataset.iloc[:, [0,1]].values

y = dataset.iloc[:, 2].values

# training and testing data

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X,y,test\_size= 0.25, random\_state=0)

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.fit\_transform(X\_test)

from sklearn.naive\_bayes import BernoulliNB

from sklearn.naive\_bayes import GaussianNB

classifer1 = GaussianNB()

classifer1.fit(X\_train, y\_train)

y\_pred1 = classifer1.predict(X\_test)

from sklearn.metrics import accuracy\_score

print(accuracy\_score(y\_test,y\_pred1))

from sklearn.metrics import accuracy\_score, confusion\_matrix, precision\_score, recall\_score

print('Accuracy Metrics: \n')

print('Accuracy: ', accuracy\_score(y\_test, y\_pred1))

print('Recall: ', recall\_score(y\_test, y\_pred1))

print('Precision: ', precision\_score(y\_test, y\_pred1))

print('Confusion Matrix: \n', confusion\_matrix(y\_test, y\_pred1))

output:

0.91

Accuracy Metrics:

Accuracy: 0.91

Recall: 0.84375

Precision: 0.8709677419354839

Confusion Matrix:

[[64 4]

[ 5 27]]

7 **BAYESIAN NETWORK FOR MEDICAL DATASET**

**Aim:**

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.

ALGORITHM:

Step 1: Import the libraries &Import the model for bayession Model.

Step 2: print the sample instances from the dataset.

Step 3: calculate the posterior conditional probability distribution of each of the possible in observed

causes given the observed evidence.

Step4: calculate the probability of heart disease given evidence for resteg attributes.

Step 5: Calculate probability of heart disease given evidence for up attribute.

**CODE:**

**import numpy as np**

**import pandas as pd**

**import csv**

**from pgmpy.estimators import MaximumLikelihoodEstimator**

**from pgmpy.models import BayesianModel**

**from pgmpy.inference import VariableElimination**

**heartDisease = pd.read\_csv('D:\AI LAB\Expt7.csv')**

**heartDisease = heartDisease.replace('?',np.nan)**

**print('Sample instances from the dataset are given below')**

**print(heartDisease.head())**

**print('\n Attributes and datatypes')**

**print(heartDisease.dtypes)**

**model= BayesianModel([('age','heartdisease'),('gender','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])**

**print('\nLearning CPD using Maximum likelihood estimators')**

**model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)**

**print('\n Inferencing with Bayesian Network:')**

**HeartDiseasetest\_infer = VariableElimination(model)**

**print('\n 1. Probability of HeartDisease given evidence= restecg')**

**q1=HeartDiseasetest\_infer.query(variables=['heartdisease'],evidence={'restecg':1})**

**print(q1)**

**print('\n 2. Probability of HeartDisease given evidence= cp ')**

**q2=HeartDiseasetest\_infer.query(variables=['heartdisease'],evidence={'cp':2})**

**print(q2)**

**output**

Sample instances from the dataset are given below

age gender cp trestbps chol fbs restecg thalach exang oldpeak \

0 63 1 1 145 233 1 2 150 0 2.3

1 67 1 4 160 286 0 2 108 1 1.5

2 67 1 4 120 229 0 2 129 1 2.6

3 37 1 3 130 250 0 0 187 0 3.5

4 41 0 2 130 204 0 2 172 0 1.4

slope ca thal heartdisease

0 3 0 6 0

1 2 3 3 2

2 2 2 7 1

3 3 0 3 0

4 1 0 3 0

Attributes and datatypes

age int64

gender int64

cp int64

trestbps int64

chol int64

fbs int64

restecg int64

thalach int64

exang int64

oldpeak float64

slope int64

ca object

thal object

heartdisease int64

dtype: object

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence= restecg

+-----------------+---------------------+

| heartdisease | phi(heartdisease) |

+=================+=====================+

| heartdisease(0) | 0.1012 |

+-----------------+---------------------+

| heartdisease(1) | 0.0000 |

+-----------------+---------------------+

| heartdisease(2) | 0.2392 |

+-----------------+---------------------+

| heartdisease(3) | 0.2015 |

+-----------------+---------------------+

| heartdisease(4) | 0.4581 |

+-----------------+---------------------+

2. Probability of HeartDisease given evidence= cp

+-----------------+---------------------+

| heartdisease | phi(heartdisease) |

+=================+=====================+

| heartdisease(0) | 0.3610 |

+-----------------+---------------------+

| heartdisease(1) | 0.2159 |

+-----------------+---------------------+

| heartdisease(2) | 0.1373 |

+-----------------+---------------------+

| heartdisease(3) | 0.1537 |

+-----------------+---------------------+

| heartdisease(4) | 0.1321 |

+-----------------+---------------------+

**8. EM algorithm and K means algorithm**

**Aim:** To apply EM algorithm to cluster a set of data stored in a .csv file. Use the same dataset for

clustering using k-means algorithm.

ALGORITHM:

Step 1: Import the Libraries & include the model for k-meas and gaussian mixture.

Step 2: Label the attribute

Step 3: Generate the colour map.

Step 4: Calculate the accuracy &confussionMadrix for k-mean

Step 5: Generate the Scatter plot for gaussion Mixture model

Step 6: calculate the accuracy & confession matrix of Gmm.

**CODE:**

from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

import sklearn.metrics as metrics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

names = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width', 'Class']

dataset = pd.read\_csv("D:\LAB\exp8.csv", names=names)

X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))

colormap=np.array(['red','lime','black'])

plt.subplot(1,3,1)

plt.title('Real')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y])

model=KMeans(n\_clusters=3, random\_state=0).fit(X)

plt.subplot(1,3,2)

plt.title('KMeans')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[model.labels\_])

print('The accuracy score of K-Mean: ',metrics.accuracy\_score(y, model.labels\_))

print('The Confusion matrixof K-Mean:\n',metrics.confusion\_matrix(y, model.labels\_))

gmm=GaussianMixture(n\_components=3, random\_state=0).fit(X)

y\_cluster\_gmm=gmm.predict(X)

plt.subplot(1,3,3)

plt.title('GMM Classification')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y\_cluster\_gmm])

print('The accuracy score of EM: ',metrics.accuracy\_score(y, y\_cluster\_gmm))

print('The Confusion matrix of EM:\n ',metrics.confusion\_matrix(y, y\_cluster\_gmm))

RESULT:

The accuracy score of K-Mean: 0.24

The Confusion matrixof K-Mean:

[[ 0 50 0]

[48 0 2]

[14 0 36]]

The accuracy score of EM: 0.36666666666666664

The Confusion matrix of EM:

[[50 0 0]

[ 0 5 45]

[ 0 50 0]]

**9. K-Nearest Neighbour Algorithm**

**Aim:**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.

ALGORITHM:

Step 1- Import the Libraries & model for Nearest Neighbour.

Step 2. Specify the names for the attributes.

Step 3. Split the training &testing .

Step4. Specify neighbour count as s.

Step5. Calculate the confussionMadrix, accuracy &classification report.

Step6. print the calculate values.importnumpy as np

**CODE:**

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from sklearn.datasets import load\_iris

iris = load\_iris()

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

df = pd.DataFrame(iris.data,columns=iris.feature\_names)

df['target'] = iris.target

X = df.iloc[:, :-1]

y = df.iloc[:, -1]

print(X.head())

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.10)

classifier = KNeighborsClassifier(n\_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0

print ("\n-------------------------------------------------------------------------")

print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))

print ("-------------------------------------------------------------------------")

for label in ytest:

print ('%-25s %-25s' % (label, ypred[i]), end="")

if (label == ypred[i]):

print (' %-25s' % ('Correct'))

else:

print (' %-25s' % ('Wrong'))

i = i + 1

print ("-------------------------------------------------------------------------")

print("\nConfusion Matrix:\n",metrics.confusion\_matrix(ytest, ypred))

print ("-------------------------------------------------------------------------")

print("\nClassification Report:\n",metrics.classification\_report(ytest, ypred))

print ("-------------------------------------------------------------------------")

print('Accuracy of the classifer is %0.2f' % metrics.accuracy\_score(ytest,ypred))

print ("-------------------------------------------------------------------------")

RESULT:

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

0 5.1 3.5 1.4 0.2

1 4.9 3.0 1.4 0.2

2 4.7 3.2 1.3 0.2

3 4.6 3.1 1.5 0.2

4 5.0 3.6 1.4 0.2

-------------------------------------------------------------------------

Original Label Predicted Label Correct/Wrong

-------------------------------------------------------------------------

2 2 Correct

1 2 Wrong

2 2 Correct

0 2 Wrong

0 2 Wrong

0 2 Wrong

0 2 Wrong

2 2 Correct

2 2 Correct

0 2 Wrong

1 2 Wrong

1 2 Wrong

2 2 Correct

0 2 Wrong

1 2 Wrong

-------------------------------------------------------------------------

Confusion Matrix:

[[6 0 0]

[0 4 0]

[0 1 4]]

-------------------------------------------------------------------------

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 6

1 0.80 1.00 0.89 4

2 1.00 0.80 0.89 5

accuracy 0.93 15

macro avg 0.93 0.93 0.93 15

weighted avg 0.95 0.93 0.93 15

-------------------------------------------------------------------------

Accuracy of the classifer is 0.93

-------------------------------------------------------------------------