1.**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 and show the output for test cases.** Develop an interactive program by Comparing the result by implementing LIST

THEN ELIMINATE algorithm.

Find-s Algorithm : 1. Load Data set

2. Initialize h to the most specific hypothesis in H

3. For each positive training instance x • For each attribute constraint ai in h If the constraint ai in h is satisfied by x then do nothing else replace ai in h by the next more general constraint that is satisfied by x 4. Output hypothesis h

WORKING PROGRAM

import csv

def loadCsv(filename):

lines = csv.reader(open(filename, "r"))

dataset = list(lines)

for i in range(len(dataset)):

dataset[i] = dataset[i]

return dataset

attributes = ['Sky','Temp','Humidity','Wind','Water','Forecast']

print('Attributes =',attributes)

num\_attributes = len(attributes)

filename = "finds.csv"

dataset = loadCsv(filename)

print(dataset)

hypothesis=['0'] \* num\_attributes

print("Intial Hypothesis")

print(hypothesis)

print("The Hypothesis are")

for i in range(0, len(dataset)):

if dataset[i][num\_attributes] == 'yes':

print ("\nInstance ", i+1, "is", dataset[i], " and is Positive Instance")

for j in range(0, num\_attributes):

if hypothesis[j] == '0' or hypothesis[j] == dataset[i][j]:

hypothesis[j] = dataset[i][j]

else:

hypothesis[j] = '?'

print(i+1, hypothesis, "\n")

if dataset[i][num\_attributes] == 'no':

print ("\nInstance ", i+1, "is", dataset[i], " and is Negative Instance Hence Ignored")

print("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")

**OUTPUT**:

Intial Hypothesis

['0', '0', '0', '0', '0', '0']

The Hypothesis are

Instance 2 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'] and is Positive Instance

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

Instance 3 is ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'] and is Positive Instance

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

Instance 4 is ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'] and is Negative Instance Hence Ignored

The hypothesis for the training instance 4 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Instance 5 is ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes'] and is Positive Instance

5 ['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instance is ['sunny', 'warm', '?', 'strong', '?', '?']

1. **For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Eliminationalgorithm. Output a description of the set of all hypotheses consistent with the training examples.**

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

import numpy as np

import pandas as pd

data = pd.read\_csv('finds.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', '?', '?', '?', '?']]

1. **Identify and Delete Rows that Contain Duplicate Data by considering an appropriate dataset.**

import pandas as pd

# Load the automobile dataset into a DataFrame

df = pd.read\_csv('Automobile\_data.csv')

# Identify duplicate rows

duplicates = df[df.duplicated()]

# View the duplicate rows

print("Duplicate Rows:")

print(duplicates)

# Delete duplicate rows

df.drop\_duplicates(inplace=True)

# Verify the deletion

print("Cleaned DataFrame:")

print(df)

**Identify and Delete Columns That Contain a Single Value by considering an appropriate dataset.**

import pandas as pd

# Load the automobile dataset into a DataFrame

df = pd.read\_csv('Automobile\_data.csv')

# Identify columns with a single value

single\_value\_columns = []

for column in df.columns:

if df[column].nunique() == 1:

single\_value\_columns.append(column)

# Delete the identified columns

df.drop(single\_value\_columns, axis=1, inplace=True)

# Verify the deletion

print("Updated DataFrame:")

print(df)

**OUTPUT:**

Duplicate Rows:

company fuel-type num of doors body-style num of cylinders rating

3 bmw gas 2 sedan four 5

8 benz diesel 4 wegon five 5

Cleaned DataFrame:

company fuel-type num of doors body-style num of cylinders rating

0 audi gas 2 seden five 5

1 audi gas 6 wegon five 5

2 bmw gas 2 sedan four 5

4 audi gas 4 sedan six 5

5 bmw gas 2 wegon five 5

6 bmw gas 2 sedan five 5

7 benz diesel 4 wegon five 5

9 benz diesel 4 sedan five 5

Updated DataFrame:

company fuel-type num of doors body-style num of cylinders

0 audi gas 2 seden five

1 audi gas 6 wegon five

2 bmw gas 2 sedan four

3 bmw gas 2 sedan four

4 audi gas 4 sedan six

5 bmw gas 2 wegon five

6 bmw gas 2 sedan five

7 benz diesel 4 wegon five

8 benz diesel 4 wegon five

9 benz diesel 4 sedan five

**4. 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.**

import numpy as np

import math

import csv

def read\_data(filename):

with open(filename, 'r') as csvfile:

datareader = csv.reader(csvfile, delimiter=',')

headers = next(datareader)

metadata = []

traindata = []

for name in headers:

metadata.append(name)

for row in datareader:

traindata.append(row)

return (metadata, traindata)

class Node:

def \_\_init\_\_(self, attribute):

self.attribute = attribute

self.children = []

self.answer = ""

def \_\_str\_\_(self):

return self.attribute

def subtables(data, col, delete):

dict = {}

items = np.unique(data[:, col])

count = np.zeros((items.shape[0], 1), dtype=np.int32)

for x in range(items.shape[0]):

for y in range(data.shape[0]):

if data[y, col] == items[x]:

count[x] += 1

for x in range(items.shape[0]):

dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

pos = 0

for y in range(data.shape[0]):

if data[y, col] == items[x]:

dict[items[x]][pos] = data[y]

pos += 1

if delete:

dict[items[x]] = np.delete(dict[items[x]], col, 1)

return items, dict

def entropy(S):

items = np.unique(S)

if items.size == 1:

return 0

counts = np.zeros((items.shape[0], 1))

sums = 0

for x in range(items.shape[0]):

counts[x] = sum(S == items[x]) / (S.size \* 1.0)

for count in counts:

sums += -1 \* count \* math.log(count, 2)

return sums

def gain\_ratio(data, col):

items, dict = subtables(data, col, delete=False)

total\_size = data.shape[0]

entropies = np.zeros((items.shape[0], 1))

intrinsic = np.zeros((items.shape[0], 1))

for x in range(items.shape[0]):

ratio = dict[items[x]].shape[0]/(total\_size \* 1.0)

entropies[x] = ratio \* entropy(dict[items[x]][:, -1])

intrinsic[x] = ratio \* math.log(ratio, 2)

total\_entropy = entropy(data[:, -1])

iv = -1 \* sum(intrinsic)

for x in range(entropies.shape[0]):

total\_entropy -= entropies[x]

return total\_entropy / iv

def create\_node(data, metadata):

if (np.unique(data[:, -1])).shape[0] == 1:

node = Node("")

node.answer = np.unique(data[:, -1])[0]

return node

gains = np.zeros((data.shape[1] - 1, 1))

for col in range(data.shape[1] - 1):

gains[col] = gain\_ratio(data, col)

split = np.argmax(gains)

node = Node(metadata[split])

metadata = np.delete(metadata, split, 0)

items, dict = subtables(data, split, delete=True)

for x in range(items.shape[0]):

child = create\_node(dict[items[x]], metadata)

node.children.append((items[x], child))

return node

def empty(size):

s = ""

for x in range(size):

s += " "

return s

def print\_tree(node, level):

if node.answer != "":

print(empty(level), node.answer)

return

print(empty(level), node.attribute)

for value, n in node.children:

print(empty(level + 1), value)

print\_tree(n, level + 2)

metadata, traindata = read\_data("tennis.csv")

data = np.array(traindata)

node = create\_node(data, metadata)

print\_tree(node, 0)

**OUTPUT**:

Outlook

Overcast

b'Yes'

Rain

Wind

b'Strong'

b'No'

b'Weak'

b'Yes'

Sunny

Humidity

b'High'

b'No'

b'Normal'

b'Yes'

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

**5.Demonstrate the working of the Random forest algorithm. Use an appropriate data set**

**for building and apply this knowledge toclassify a new sample.**

import numpy as np

import pandas as pd

import matplotlib as mpl

import matplotlib.pyplot as plt

# get titanic & test csv files as a DataFrame

train = pd.read\_csv("C:\\Users\\Aman\_kumar05\\Desktop/train.csv")

print(train.shape)

#Checking for missing data

NAs = pd.concat([train.isnull().sum()], axis=1, keys=["Train"])

NAs[NAs.sum(axis=1) > 0]

train.pop("Cabin")

train.pop("Name")

train.pop("Ticket")

# Filling missing Age values with mean

train["Age"] = train["Age"].fillna(train["Age"].mean())

# Filling missing Embarked values with most common value

train["Embarked"] = train["Embarked"].fillna(train["Embarked"].mode()[0])

train["Pclass"] = train["Pclass"].apply(str)

# Getting Dummies from all other categorical vars

for col in train.dtypes[train.dtypes == "object"].index:

for\_dummy = train.pop(col)

train = pd.concat([train, pd.get\_dummies(for\_dummy, prefix=col)], axis=1)

train.head()

labels = train.pop("Survived")

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(train, labels, test\_size=0.25)

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

rf.fit(x\_train, y\_train)

y\_pred = rf.predict(x\_test)

from sklearn.metrics import roc\_curve, auc

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_pred)

roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

roc\_auc

n\_estimators = [1, 2, 4, 8, 16, 32, 64, 100, 200]

train\_results = []

test\_results = []

for estimator in n\_estimators:

rf = RandomForestClassifier(n\_estimators=estimator, n\_jobs=-1)

rf.fit(x\_train, y\_train)

train\_pred = rf.predict(x\_train)

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_train, train\_pred)

roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

train\_results.append(roc\_auc)

y\_pred = rf.predict(x\_test)

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_pred)

roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

test\_results.append(roc\_auc)

from matplotlib.legend\_handler import HandlerLine2D

line1, = plt.plot(n\_estimators, train\_results, "b", label="Train AUC")

line2, = plt.plot(n\_estimators, test\_results, "r", label="Test AUC")

plt.legend(handler\_map={line1: HandlerLine2D(numpoints=2)})

plt.ylabel("AUC score")

plt.xlabel("n\_estimators")

plt.show()

**OUTPUT**:

(891, 12)

6.**Write a program to implement the naïve Bayesian classifier for a sample training data**

**set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

import csv

import random

import math

def loadcsv(filename):

lines = csv.reader(open(filename, "r"));

dataset = list(lines)

for i in range(len(dataset)):

#converting strings into numbers for processing

dataset[i] = [float(x) for x in dataset[i]]

return dataset

def splitdataset(dataset, splitratio):

#67% training size

trainsize = int(len(dataset) \* splitratio);

trainset = []

copy = list(dataset);

while len(trainset) < trainsize:

#generate indices for the dataset list randomly to pick ele for training data

index = random.randrange(len(copy));

trainset.append(copy.pop(index))

return [trainset, copy]

def separatebyclass(dataset):

separated = {} #dictionary of classes 1 and 0

#creates a dictionary of classes 1 and 0 where the values are

#the instances belonging to each class

for i in range(len(dataset)):

vector = dataset[i]

if (vector[-1] not in separated):

separated[vector[-1]] = []

separated[vector[-1]].append(vector)

return separated

def mean(numbers):

return sum(numbers)/float(len(numbers))

def stdev(numbers):

avg = mean(numbers)

variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)

return math.sqrt(variance)

def summarize(dataset): #creates a dictionary of classes

summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(\*dataset)];

del summaries[-1] #excluding labels +ve or -ve

return summaries

def summarizebyclass(dataset):

separated = separatebyclass(dataset);

#print(separated)

summaries = {}

for classvalue, instances in separated.items():

#for key,value in dic.items()

#summaries is a dic of tuples(mean,std) for each class value

summaries[classvalue] = summarize(instances) #summarize is used to cal to mean and std

return summaries

def calculateprobability(x, mean, stdev):

exponent = math.exp(-(math.pow(x-mean,2)/(2\*math.pow(stdev,2))))

return (1 / (math.sqrt(2\*math.pi) \* stdev)) \* exponent

def calculateclassprobabilities(summaries, inputvector):

probabilities = {} # probabilities contains the all prob of all class of test data

for classvalue, classsummaries in summaries.items():#class and attribute information as mean and sd

probabilities[classvalue] = 1

for i in range(len(classsummaries)):

mean, stdev = classsummaries[i] #take mean and sd of every attribute for class 0 and 1 seperaely

x = inputvector[i] #testvector's first attribute

probabilities[classvalue] \*= calculateprobability(x, mean, stdev);#use normal dist

return probabilities

def predict(summaries, inputvector): #training and test data is passed

probabilities = calculateclassprobabilities(summaries, inputvector)

bestLabel, bestProb = None, -1

for classvalue, probability in probabilities.items():#assigns that class which has he highest prob

if bestLabel is None or probability > bestProb:

bestProb = probability

bestLabel = classvalue

return bestLabel

def getpredictions(summaries, testset):

predictions = []

for i in range(len(testset)):

result = predict(summaries, testset[i])

predictions.append(result)

return predictions

def getaccuracy(testset, predictions):

correct = 0

for i in range(len(testset)):

if testset[i][-1] == predictions[i]:

correct += 1

return (correct/float(len(testset))) \* 100.0

def main():

filename = 'naivedata.csv'

splitratio = 0.67

dataset = loadcsv(filename);

trainingset, testset = splitdataset(dataset, splitratio)

print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset), len(testset)))

# prepare model

summaries = summarizebyclass(trainingset);

#print(summaries)

# test model

predictions = getpredictions(summaries, testset) #find the predictions of test data with the training data

accuracy = getaccuracy(testset, predictions)

print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()

================================================================================

**OUTPUT :**

Split 306 rows into train=205 and test=101 rows

Accuracy: 72.27722772277228%

**7.Assuming a set of documents that need to be classified, use the naive Bayesian**

**Classifier model to perform this task. Calculate the accuracy, precision, and recall for**

**your data set.**

import pandas as pd

msg = pd.read\_csv('document.csv', names=['message', 'label'])

print("Total Instances of Dataset: ", msg.shape[0])

msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})

X = msg.message

y = msg.labelnum

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

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

from sklearn.feature\_extraction.text import CountVectorizer

count\_v = CountVectorizer()

Xtrain\_dm = count\_v.fit\_transform(Xtrain)

Xtest\_dm = count\_v.transform(Xtest)

df = pd.DataFrame(Xtrain\_dm.toarray(),columns=count\_v.get\_feature\_names\_out())

print(df[0:5])

from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB()

clf.fit(Xtrain\_dm, ytrain)

pred = clf.predict(Xtest\_dm)

for doc, p in zip(Xtrain, pred):

p = 'pos' if p == 1 else 'neg'

print("%s -> %s" % (doc, p))

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

print('Accuracy Metrics: \n')

print('Accuracy: ', accuracy\_score(ytest, pred))

print('Recall: ', recall\_score(ytest, pred))

print('Precision: ', precision\_score(ytest, pred))

**OUTPUT**:

Total Instances of Dataset: 18

about am amazing an and awesome ... to today very view went what

0 0 0 0 0 0 0 ... 0 0 0 0 0 0

1 0 0 0 0 0 0 ... 0 0 0 0 0 0

2 0 0 0 0 0 0 ... 0 0 0 0 0 0

3 0 0 1 1 0 0 ... 0 0 0 0 0 0

4 0 1 0 0 0 0 ... 0 0 0 0 0 0

[5 rows x 45 columns]

I do not like this restaurant -> pos

He is my sworn enemy -> pos

My boss is horrible -> neg

This is an amazing place -> neg

I am tired of this stuff -> neg

Accuracy Metrics:

Accuracy: 0.6

Recall: 0.5

Precision: 1.0

**8. Construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.**

**Source 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('heart.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'),('sex','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:

╒════════════════╤════

│ heartdisease │ phi(heartdisease) │

╞══════════════════════

│ heartdisease\_0 │ 0.5593 │

├─────────────────────┤

│ heartdisease\_1 │ 0.4407

**9 Demonstrate the working of EM algorithm to cluster a set of data stored in a .CSV file.**

**Source 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("iris.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'])

# REAL PLOT

plt.subplot(1,3,1)

plt.title('Real')

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

# K-PLOT

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 PLOT

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))

**OUTPUT**:

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]]

1. **Demonstrate the working of SVM classifier for a suitable data set**

# importing numpy library

import numpy as np

class SVM\_classifier():

# initiating the hyperparameters

def \_\_init\_\_(self, learning\_rate, no\_of\_iterations, lambda\_parameter):

self.learning\_rate = learning\_rate

self.no\_of\_iterations = no\_of\_iterations

self.lambda\_parameter = lambda\_parameter

# fitting the dataset to SVM Classifier

def fit(self, X, Y):

# m  --> number of Data points --> number of rows

# n  --> number of input features --> number of columns

self.m, self.n = X.shape

# initiating the weight value and bias value

self.w = np.zeros(self.n)

self.b = 0

self.X = X

self.Y = Y

# implementing Gradient Descent algorithm for Optimization

for i in range(self.no\_of\_iterations):

self.update\_weights()

# function for updating the weight and bias value

def update\_weights(self):

# label encoding

y\_label = np.where(self.Y <= 0, -1, 1)

# gradients ( dw, db)

for index, x\_i in enumerate(self.X):

condition = y\_label[index] \* (np.dot(x\_i, self.w) - self.b) >= 1

if (condition == True):

dw = 2 \* self.lambda\_parameter \* self.w

db = 0

else:

dw = 2 \* self.lambda\_parameter \* self.w - np.dot(x\_i, y\_label[index])

db = y\_label[index]

self.w = self.w - self.learning\_rate \* dw

self.b = self.b - self.learning\_rate \* db

# predict the label for a given input value

def predict(self, X):

output = np.dot(X, self.w) - self.b

predicted\_labels = np.sign(output)

y\_hat = np.where(predicted\_labels <= -1, 0, 1)

return y\_hat

import pandas as pd

from sklearn.preprocessing import StandardScaler

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

from sklearn.metrics import accuracy\_score

# loading the data from csv file to pandas dataframe

diabetes\_data = pd.read\_csv('diabetes.csv')

# print the first 5 rows of the dataframe

diabetes\_data.head()

# number of rows and columns in the dataset

diabetes\_data.shape

# getting the statistical measures of the dataset

diabetes\_data.describe()

diabetes\_data['Outcome'].value\_counts()

# separating the features and target

features = diabetes\_data.drop(columns='Outcome', axis=1)

target = diabetes\_data['Outcome']

print(features)

print(target)

scaler = StandardScaler()

scaler.fit(features)

standardized\_data = scaler.transform(features)

print(standardized\_data)

features = standardized\_data

target = diabetes\_data['Outcome']

print(features)

print(target)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(features, target, test\_size=0.2, random\_state = 2)

print(features.shape, X\_train.shape, X\_test.shape)

classifier = SVM\_classifier(learning\_rate=0.001, no\_of\_iterations=1000, lambda\_parameter=0.01)

# training the SVM classifier with training data

classifier.fit(X\_train, Y\_train)

# accuracy on training data

X\_train\_prediction = classifier.predict(X\_train)

training\_data\_accuracy = accuracy\_score(Y\_train, X\_train\_prediction)

print('Accuracy score on training data = ', training\_data\_accuracy)

# accuracy on training data

X\_test\_prediction = classifier.predict(X\_test)

test\_data\_accuracy = accuracy\_score(Y\_test, X\_test\_prediction)

print('Accuracy score on test data = ', test\_data\_accuracy)

input\_data = (5,166,72,19,175,25.8,0.587,51)

# change the input data to numpy array

input\_data\_as\_numpy\_array = np.asarray(input\_data)

# reshape the array

input\_data\_reshaped = input\_data\_as\_numpy\_array.reshape(1,-1)

# standardizing the input data

std\_data = scaler.transform(input\_data\_reshaped)

print(std\_data)

prediction = classifier.predict(std\_data)

print(prediction)

if (prediction[0] == 0):

print('The person is not diabetic')

else:

print('The Person is diabetic')

**OUTPUT:**

Name: Outcome, Length: 768, dtype: int64

(768, 8) (614, 8) (154, 8)

Accuracy score on training data = 0.7768729641693811

Accuracy score on test data = 0.7532467532467533

[[ 0.3429808 1.41167241 0.14964075 -0.09637905 0.82661621 -0.78595734

0.34768723 1.51108316]]

[1]

The Person is diabetic