**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.**

import csv

data = []

with open('enjoysport.csv') as csvfile:

for row in csv.reader(csvfile):

if row[-1] == "yes":

data.append(row[:-1])

s = ['0']\*len(data[0])

for d in data:

i = 0

for feature in d:

if s[i] == '0' or s[i] == feature:

s[i] = feature

else:

s[i] = '?'

i += 1

print(s)

**2. 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.**

import csv

data = []

with open('trainingexamples.csv') as csvFile:

for line in csv.reader(csvFile):

data.append(tuple(line))

def Domain():

for i in range(len(data[0])):

l=[]

for ele in data:

if ele[i] not in l:

l.append(ele[i])

d.append(l)

return d

D = Domain()

def consistant(h1, h2): # compare the 2 participating hypothesis for compatibility

for x, y in zip(h1, h2):

if not (x == "?" or (x != "ɸ" and (x == y or y == "ɸ"))):

return False

return True

def candidate\_elimination():

G = {('?',)\*(len(data[0]) - 1),}

S = ['ɸ']\*(len(data[0]) - 1)

no = 0

print("\nG:",G)

print("\nS:",S)

for item in data:

no += 1

inp , res = item[:-1] , item[-1]

if res in "Yy": # For +ve examples

i = 0

G = {g for g in G if consistant(g,inp)}

for s,x in zip(S,inp):

if not s==x:

S[i] = '?' if s != 'ɸ' else x

i += 1

else: # For -ve examples

S = S

Gprev = G.copy()

for g in Gprev:

for i in range(len(g)):

if g[i] == "?":

for val in D[i]:

if inp[i] != val and val == S[i]:

g\_new = g[:i] + (val,) + g[i+1:]

G.add(g\_new)

else:

G.add(g)

if ('?','?','?','?','?','?') in G :

G.remove(('?','?','?','?','?','?'))

print("\nG:",G)

print("\nS:",S)

candidate\_elimination()

**3. 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.**

def infoGain(P, N):

import math

return -P / (P + N) \* math.log2(P / ( P + N)) - N / (P + N) \* math.log2(N / (P + N))

def insertNode(tree, addTo, Node):

for k, v in tree.items():

if isinstance(v, dict):

tree[k] = insertNode(v, addTo, Node)

if addTo in tree:

if isinstance(tree[addTo], dict):

tree[addTo][Node] = 'None'

else:

tree[addTo] = {Node:'None'}

return tree

def insertConcept(tree, addTo, Node):

for k, v in tree.items():

if isinstance(v, dict):

tree[k] = insertConcept(v, addTo, Node)

if addTo in tree:

tree[addTo] = Node

return tree

def getNextNode(data, AttributeList, concept, conceptVals, tree, addTo):

Total = data.shape[0]

if Total == 0:

return tree

countC = {}

for cVal in conceptVals:

dataCC = data[data[concept] == cVal]

countC[cVal] = dataCC.shape[0]

if countC[conceptVals[0]] == 0:

tree = insertConcept(tree, addTo, conceptVals[1])

return tree

if countC[conceptVals[1]] == 0:

tree = insertConcept(tree, addTo, conceptVals[0])

return tree

ClassEntropy = infoGain(countC[conceptVals[1]],countC[conceptVals[0]])

Attr = {}

for a in AttributeList:

Attr[a] = list(set(data[a]))

AttrCount = {}

EntropyAttr = {}

for att in Attr:

for vals in Attr [att]:

for c in conceptVals:

iData = data[data[att] == vals]

dataAtt = iData[iData[concept] == c]

AttrCount[c] = dataAtt.shape[0]

TotalInfo = AttrCount[conceptVals[1]] + AttrCount[conceptVals[0]]

if AttrCount[conceptVals[1]] == 0 or AttrCount[conceptVals[0]] == 0:

InfoGain=0

else:

InfoGain = infoGain(AttrCount[conceptVals[1]], AttrCount[conceptVals[0]])

if att not in EntropyAttr:

EntropyAttr[att] = ( TotalInfo / Total ) \* InfoGain

else:

EntropyAttr[att] = EntropyAttr[att] + ( TotalInfo / Total ) \* InfoGain

Gain = {}

for g in EntropyAttr:

Gain[g] = ClassEntropy - EntropyAttr[g]

Node = max(Gain, key = Gain.get)

tree = insertNode(tree, addTo, Node)

for nD in Attr[Node]:

tree = insertNode(tree, Node, nD)

newData = data[data[Node] == nD].drop(Node, axis = 1)

AttributeList=list(newData)[:-1]

tree = getNextNode(newData, AttributeList, concept, conceptVals, tree, nD)

return tree

def main():

import pandas as pd

data = pd.read\_csv('id3.csv')

AttributeList = list(data)[:-1]

concept = str(list(data)[-1])

conceptVals = list(set(data[concept]))

tree = getNextNode(data, AttributeList, concept, conceptVals, {'root':'None'}, 'root')

print(tree)

compute(tree)

main()

**4. Build an Artificial Neural Network by implementing the Backpropagation**

**algorithm and test the same using appropriate data sets.**

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) # max of array

y = y/100

def sigmoid (x):

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

def derivatives\_sigmoid(x):

return x \* (1 - x)

epoch=5000

lr=0.1

wh = np.random.uniform(size=(2,3))

bh = np.random.uniform(size=(1,3))

wout = np.random.uniform(size=(3,1))

bout = np.random.uniform(size=(1,1))

for i in range(epoch):

# forward prop

hinp=np.dot(X,wh) + bh

hlayer\_act = sigmoid(hinp)

outinp=np.dot(hlayer\_act,wout) + bout

output = sigmoid(outinp)

hiddengrad = derivatives\_sigmoid(hlayer\_act)

outgrad = derivatives\_sigmoid(output)

EO = y-output

d\_output = EO\* outgrad

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

d\_hiddenlayer = EH \* hiddengrad

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)

**5. 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)):

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

return dataset

def splitDataset(dataset, splitRatio):

trainSize = int(len(dataset) \* splitRatio)

trainSet = []

trainSet,testSet = dataset[:trainSize],dataset[trainSize:]

return [trainSet, testSet]

def mean(numbers):

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

def stdev(numbers):

avg = mean(numbers)

v = 0

for x in numbers:

v += (x-avg)\*\*2

return math.sqrt(v/(len(numbers)-1))

def summarizeByClass(dataset):

separated = {}

for i in range(len(dataset)):

vector = dataset[i]

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

separated[vector[-1]] = []

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

summaries = {}

for classValue, instances in separated.items():

summaries[classValue] = [(mean(attribute), stdev(attribute)) for attribute in zip(\*instances)][:-1]

return summaries

def calculateProbability(x, mean, stdev):

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

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

def predict(summaries, inputVector):

probabilities = {}

for classValue, classSummaries in summaries.items():

probabilities[classValue] = 1

for i in range(len(classSummaries)):

mean, stdev = classSummaries[i]

x = inputVector[i]

probabilities[classValue] \*= calculateProbability(x, mean, stdev)

bestLabel, bestProb = None, -1

for classValue, probability in probabilities.items():

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/(len(testSet))) \* 100.0

filename = 'pima-indians-diabetes.csv'

splitRatio = 0.67

dataset = loadcsv(filename)

trainingSet, testSet = splitDataset(dataset, splitRatio)

summaries = summarizeByClass(trainingSet)

predictions = getPredictions(summaries, testSet)

print("\nPredictions:\n",predictions)

accuracy = getAccuracy(testSet, predictions)

print('Accuracy ',accuracy)

**6. Assuming a set of documents that need to be classified, use the naïve 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.**

from sklearn.datasets import fetch\_20newsgroups

from sklearn.metrics import confusion\_matrix, classification\_report, accuracy\_score

import numpy as np

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

twenty\_train = fetch\_20newsgroups(subset='train',shuffle=True)

categories = twenty\_train.target\_names[:4]

twenty\_train = fetch\_20newsgroups(subset='train',categories=categories,shuffle=True)

twenty\_test = fetch\_20newsgroups(subset='test',categories=categories,shuffle=True)

tfidf\_transformer = CountVectorizer()

X\_train\_tfidf = tfidf\_transformer.fit\_transform(twenty\_train.data)

mod = MultinomialNB()

mod.fit(X\_train\_tfidf, twenty\_train.target)

X\_test\_tfidf = tfidf\_transformer.transform(twenty\_test.data)

predicted = mod.predict(X\_test\_tfidf)

print("Accuracy:", accuracy\_score(twenty\_test.target, predicted)\*100)

print(classification\_report(twenty\_test.target,predicted,target\_names=twenty\_test.target\_names))

print("confusion matrix is \n",confusion\_matrix(twenty\_test.target, predicted))

**7. Write a program to construct aBayesian 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.**

import numpy as np

from urllib.request import urlopen

import pandas as pd

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.inference import VariableElimination

URL = 'http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.hungarian.data'

names = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n']

heartDisease = pd.read\_csv(urlopen(URL), names = names)

del heartDisease['k']

del heartDisease['l']

del heartDisease['m']

del heartDisease['j']

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

model = BayesianModel([('a','b'),('c','d'),('e','f'),('g','h'),('i','n')])

model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

model.get\_independencies()

HeartDisease\_infer = VariableElimination(model)

q = HeartDisease\_infer.query(variables=['n'], evidence={'a': 28})

print(q)

q = HeartDisease\_infer.query(variables=['n'], evidence={'a': 50})

print(q)

**8. Apply the 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.**

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import sklearn.metrics as sm

import pandas as pd

import numpy as np

from sklearn import preprocessing

from sklearn.mixture import GaussianMixture

l1 = [0,1,2]

def rename(s):

l2 = []

for i in s:

if i not in l2:

l2.append(i)

for i in range(len(s)):

pos = l2.index(s[i])

s[i] = l1[pos]

return s

iris = datasets.load\_iris()

X = pd.DataFrame(iris.data,columns =['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width'] )

y = pd.DataFrame(iris.target,columns = ['Targets'])

def graph\_plot(l,title,s,target):

plt.subplot(l[0],l[1],l[2])

if s==1:

plt.scatter(X.Sepal\_Length,X.Sepal\_Width, c=colormap[target], s=40)

else:

plt.scatter(X.Petal\_Length,X.Petal\_Width, c=colormap[target], s=40)

plt.title(title)

plt.figure()

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

graph\_plot([1, 2, 1],'sepal',1,y.Targets)

graph\_plot([1, 2, 2],'petal',0,y.Targets)

plt.show()

def fit\_model(modelName):

model = modelName(3)

model.fit(X)

plt.figure()

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

graph\_plot([1, 2, 1],'Real Classification',0,y.Targets)

if modelName == KMeans:

m = 'Kmeans’

else:

m = 'Em'

y1 = model.predict(X)

graph\_plot([1, 2, 2],m,0,y1)

plt.show()

km = rename(y1)

print("\nPredicted: \n", km)

print("Accuracy ",sm.accuracy\_score(y, km))

print("Confusion Matrix ",sm.confusion\_matrix(y, km))

fit\_model(KMeans)

fit\_model(GaussianMixture)

**9. 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.**

from sklearn.datasets import load\_iris

from sklearn.neighbors import KNeighborsClassifier

import numpy as np

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

iris\_dataset = load\_iris()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris\_dataset["data"],iris\_dataset["target"])

kn = KNeighborsClassifier()

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

y\_pred = kn.predict(X\_test)

a = [0]\*2

true = 0

for i,j in zip(y\_pred, y\_test):

if i == j:

a[0] = [i, j, 'Correct']

true += 1

else:

a[1] = [i, j, 'Wrong']

print("Accuracy (%) : ", true/len(y\_pred)\*100)

print(a)

Op:

Accuracy (%) : 94.73684210526315

[[0, 0, 'Correct'], [1, 2, 'Wrong']]

**10. Implement the non-parametric Locally Weighted Regression algorithm in order to**

**fit data points. Select appropriate data set for your experiment and draw graphs.**

import numpy as np

import matplotlib.pyplot as plt

def local\_regression(x0, X, Y, tau):

x0 = [1, x0]

X = [[1, i] for i in X]

X = np.asarray(X)

xw = (X.T) \* np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau))

beta = np.linalg.pinv(xw @ X) @ xw @ Y @ x0

return beta

def draw(tau):

prediction = [local\_regression(x0, X, Y, tau) for x0 in domain]

plt.plot(X, Y, 'o', color='black')

plt.plot(domain, prediction, color='red')

plt.show()

X = np.linspace(-3, 3, num=1000)

domain = X

Y = np.log(np.abs(X \*\* 2 - 1) + .5)

draw(10)

draw(0.1)

