ML PROGRAMS

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

```
import numpy as np
import pandas as pd

data = pd.DataFrame(data = pd.read_csv("finds.csv"))
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])

def learn(concepts,target):
    specific_h = concepts[0].copy()
    for i,h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = "?"
    return (specific_h)

specific_h = learn(concepts,target)
print(specific_h)
```

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 numpy as np
import pandas as pd

data = pd.DataFrame(data = pd.read_csv("finds.csv"))
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])

def learn(concepts,target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))]
                                                for i in range(len(specific_h))]


    for i,h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = "?"
                    general_h[x][x] = "?"

       if target[i] == "No":
            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] = "?"


    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 S: ",s_final)
print("Final G: ",g_final)
```

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.

```
import pandas as pd
import numpy as np
dataset= pd.read_csv('P3_Tennis.csv')
dataset


def entropy(target_col):
elements,counts = np.unique(target_col,return_counts = True)
entropy = np.sum([(-
counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in
range(len(elements))])
return entropy

def InfoGain(data,split_attribute_name,target_name="PlayTennis"):
    total_entropy = entropy(data[target_name])
    vals,counts= np.unique(data[split_attribute_name],return_counts=True)
    Weighted_Entropy = 
np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_na
me]==vals[i]).dropna()[target_name]) for i in range(len(vals))])
    InfoGain = total_entropy - Weighted_Entropy
    return InfoGain

def 
ID3(data,originaldata,features,target_attribute_name="PlayTennis",parent_node
_class = None):
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data)==0:
        return 
np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originalda
ta[target_attribute_name],return_counts=True)[1])]
    elif len(features) ==0:
        return parent_node_class
    else:
        parent_node_class = 
np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attrib
ute_name],return_counts=True)[1])]
        item_values = [InfoGain(data,feature,target_attribute_name) for 
feature in features] #Return the information gain values for the features in 
the dataset
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]
        tree = {best_feature:{}}
        features = [i for i in features if i != best_feature]
        for value in np.unique(data[best_feature]):
            value = value
            sub_data = data.where(data[best_feature] == value).dropna()
            subtree = 
ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)
            tree[best_feature][value] = subtree
        return(tree)

tree = ID3(dataset,dataset,dataset.columns[:-1])
print(dataset.head())
print(' \nDisplay Tree\n',tree)
```

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,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=7000 #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))
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))
#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)
    hiddengrad = derivatives_sigmoid(hlayer_act)
    #how much hidden layer wts contributed to error

    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr
```

```
# dotproduct of nextlayererror and currentlayerop
    # bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    #bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *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)):
        #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 = {}
```

```
#creates a dictionary of classes 1 and 0 where the values are the instacnes
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):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in
zip(*dataset)]
    del summaries[-1]
    return summaries

def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.items():
        #summaries is a dic of tuples(mean,std) for each class value
        summaries[classValue] = summarize(instances)
    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 = {}
    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]
            x = inputVector[i]
      probabilities[classValue] *= calculateProbability(x, mean, stdev)
   return probabilities
```

```
def predict(summaries, inputVector):
    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 = '5_pima-indians-diabetes.data.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);

    # test model
    predictions = getPredictions(summaries, testSet)

    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

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.

```
import pandas as pd
msg=pd.read_csv('6pg.csv',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
print(X)
print(y)
#splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print(xtest.shape)
print(xtrain.shape)
print(ytest.shape)
print(ytrain.shape)
#output of count vectoriser is a sparse matrix
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print(count_vect.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df)#tabular representation
print(xtrain_dtm) #sparse matrix representation
# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
#printing accuracy metrics
from sklearn import metrics
print('Accuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('Recall and Precison ')
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))
```

7. 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.

```
import bayespy as bp
import numpy as np
import csv
from colorama import init
from colorama import Fore, Back, Style
init()

# Define Parameter Enum values
#Age
ageEnum = {'SuperSeniorCitizen':0, 'SeniorCitizen':1, 'MiddleAged':2,
'Youth':3, 'Teen':4}
# Gender
genderEnum = {'Male':0, 'Female':1}
# FamilyHistory
familyHistoryEnum = {'Yes':0, 'No':1}
# Diet(Calorie Intake)
dietEnum = {'High':0, 'Medium':1, 'Low':2}
# LifeStyle
lifeStyleEnum = {'Athlete':0, 'Active':1, 'Moderate':2, 'Sedetary':3}
# Cholesterol
cholesterolEnum = {'High':0, 'BorderLine':1, 'Normal':2}
# HeartDisease
heartDiseaseEnum = {'Yes':0, 'No':1}
#heart_disease_data.csv
with open('7_heart_disease_data.csv') as csvfile:
    lines = csv.reader(csvfile)
    dataset = list(lines)
    data = []
for x in dataset:

      data.append([ageEnum[x[0]],genderEnum[x[1]],familyHistoryEnum[x[2]],die
      tEnum[x[3]],lifeStyleEnum[x[4]],cholesterolEnum[x[5]],heartDiseaseEnum[
      x[6]]])
# Training data for machine learning todo: should import from csv
data = np.array(data)
print (data)
N = len(data)
print(f"N={N}")

# Input data column assignment
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
```

```
print(f"p_age={p_age}")
age = bp.nodes.Categorical(p_age, plates=(N,))
print(f"age={age}")
age.observe(data[:,0])
print(f"OBSERVE AGE{age.observe(data[:,0])}")

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
print(f"p_gender={p_gender}")
gender = bp.nodes.Categorical(p_gender, plates=(N,))
print(f"gender={gender}")
gender.observe(data[:,1])

p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))
print(f"p_familyhistory={p_familyhistory}")
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
print(f"familyhistory={familyhistory}")
familyhistory.observe(data[:,2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
print(f"p_diet={p_diet}")
diet = bp.nodes.Categorical(p_diet, plates=(N,))
print(f"diet={diet}")
diet.observe(data[:,3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
print(f"p_lifestyle={p_lifestyle}")
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
print(f"lifestyle={lifestyle}")
lifestyle.observe(data[:,4])

p_cholesterol = bp.nodes.Dirichlet(1.0*np.ones(3))
print(f"p_cholesterol={p_cholesterol}")
cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))
print(f"cholesterol={cholesterol}")
cholesterol.observe(data[:,5])
#print(data)

# Prepare nodes and establish edges
# np.ones(2) ->  HeartDisease has 2 options Yes/No
# plates(5, 2, 2, 3, 4, 3)  ->  corresponds to options present for domain
values
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
print(f"p_heartdisease={p_heartdisease}")
heartdisease = bp.nodes.MultiMixture([age, gender, familyhistory, diet,
lifestyle, cholesterol], bp.nodes.Categorical, p_heartdisease)
#print(f"heartdisease={heartdisease}")
```

```
heartdisease.observe(data[:,6])
p_heartdisease.update()
print(data)

# Interactive Test
m = 0
while m == 0:
    print("\n")
      res = bp.nodes.MultiMixture([int(input('Enter Age: y' + str(ageEnum))),
      int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter
      FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum:
      ' + str(dietEnum))), int(input('Enter LifeStyle: ' +
      str(lifeStyleEnum))), int(input('Enter Cholesterol: ' +
      str(cholesterolEnum)))], bp.nodes.Categorical,
      p_heartdisease).get_moments()[0][heartDiseaseEnum['No']]
    print("Probability(HeartDisease) = " +  str(res))
    #print(Style.RESET_ALL)
    m = int(input("Enter for Continue:0, Exit :1  "))
```

8.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.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

 # import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

 # Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to

 # # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
```

```
 # Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
 # Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

 # General EM for GMM
from sklearn import preprocessing
 # transform your data such that its distribution will have a
 # mean value 0 and standard deviation of 1.
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)


plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the
true labels more closely than the Kmeans.')
```

---

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.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_iris
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Class'] = data.target_names[data.target]
df.head()
x = df.iloc[:, :-1].values
y = df.Class.values
print(x[:5])
print(y[:5])
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(x_train, y_train)
predictions = knn_classifier.predict(x_test)
print(predictions)
from sklearn.metrics import accuracy_score, confusion_matrix
print("Training accuracy Score is : ", accuracy_score(y_train,
knn_classifier.predict(x_train)))
print("Testing accuracy Score is : ", accuracy_score(y_test,
knn_classifier.predict(x_test)))
print("Training Confusion Matrix is : \n", confusion_matrix(y_train,
knn_classifier.predict(x_train)))
print("Testing Confusion Matrix is : \n", confusion_matrix(y_test,
knn_classifier.predict(x_test)))
```

---

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 matplotlib.pyplot as plt
import pandas as pd
import numpy as np



def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W
def localWeightRegression(xmat,ymat,k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();
# load data points
data = pd.read_csv('10data_tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)
mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols
ypred = localWeightRegression(X,mtip,8) # increase k to get smooth curves
graphPlot(X,ypred)
```