ML Report Lab Test 1

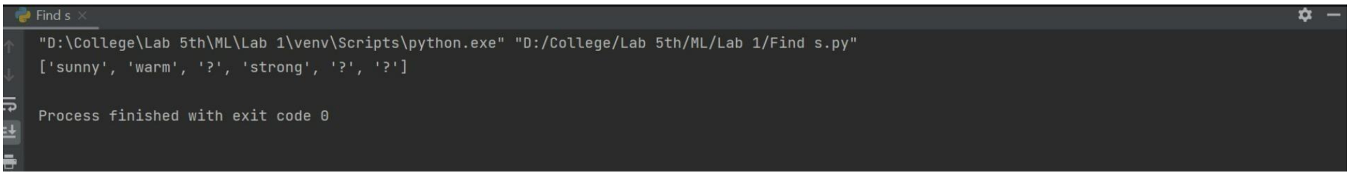
RITVIKA SINGH RATHORE

1BM19CS131

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

```
import csv
a=[]
with open('data.csv') as dataset:
    for x in csv.reader(dataset):
        a.append(x)
a.remove(a[0])
msh = ['0']*(len(a[0])-1)
for x in a:
    if x[len(x)-1]=='yes' or x[len(x)-1]=='Yes':
        for i in range(0,len(msh)):
            if msh[i]=='0' or msh[i]==x[i]:
                msh[i]=x[i]
            else:
                msh[i]='?'
print(msh)
```

Output:



```
Find s
"D:\College\Lab 5th\ML\Lab 1\venv\Scripts\python.exe" "D:/College/Lab 5th/ML/Lab 1/Find s.py"
['sunny', 'warm', '?', 'strong', '?', '?']

Process finished with exit code 0
```

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

# Reading the data from CSV file

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

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

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

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

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



def train(concepts, target):


    # Initializing general and specific hypothesis

    specific_h = concepts[0].copy()

    print("\nInitialization of specific hypothesis and general hypothesis")

    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, val in enumerate(concepts):

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

        #positive example

        if target[i] == "yes":

            print("Instance is Positive ")

            for x in range(len(specific_h)):

                if val[x]!= specific_h[x]:

                    specific_h[x] ='?'
```

```
                general_h[x][x] ='?'
        #negative example
        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if val[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 = train(concepts, target)
# displaying Specific_hypothesis
print("Final Specific_h: ", s_final, sep="\n")
# displaying Generalized_Hypothesis
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 hypothesis and general hypothesis

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. 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
import math
data=pd.read_csv('/content/data set-1.csv');

attributes=[feat for feat in data]
attributes.remove('answer')
# print(features)
class Node:
  def __init__(self):
    self.children=[];
    self.isLeaf=False;
    self.value="";
    self.pred="";

def main():
  res=ID3(data,attributes)
  printTree(res)

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)

# This function creates the decision tree recursively
def ID3(data_set,attributes):
  root=Node()
  max_gain=0.0;
  max_feat="";
  # Comparitively find out which attribute gives us the maximum information
  for attribute in attributes:
    gain=info_gain(data_set,attribute)
    if gain>max_gain:
      max_gain=gain
      max_feat=attribute
  # once we find the max gain, that will be the attribute which we use.
  root.value=max_feat

  # All types of a particular attribute. Ex: In outlook, we have sunny,rain,overcast
  types=np.unique(data_set[max_feat])

  for t in types:
    # Get all instances which match a particular type
    subdata=data_set[data_set[max_feat]==t]

    # In case we find instances where we have only one type of data result (yes/no). Entropy will be zero (Obviously!!)
    if entropy(subdata)==0.0:
      newNode=Node()
      newNode.isLeaf=True
      newNode.value=t
      newNode.pred=np.unique(subdata["answer"])
      root.children.append(newNode)
```

```
    else:
      # If even one instance has different type of data result, we still cannot come to conclusion,
      # hence go to the next attribute and create the node and apply the same algorithm on the next attribute.
      dummyNode=Node()
      dummyNode.value=t
      new_attr=attributes.copy()
      # We can remove the current attribute, only when we have come to a conclusion
      # that we cannot decide with this attribute, we have gone to the next attribute. Hence we don't want to come back.
      # + we may get stuck in cycle.
      new_attr.remove(max_feat)

      # Apply the algorithm on the next attribute with same current attributes which have been deleted.
      child=ID3(subdata,new_attr)
      dummyNode.children.append(child)
      root.children.append(dummyNode)
  return root



def info_gain(data_set,feature):
  types=np.unique(data_set[feature])
  # We are trying to get the entropy for the entire data_set we have taken into consideration.
  gain=entropy(data_set)

  for u in types:
    subdata=data_set[data_set[feature]==u]
    sub_e=entropy(subdata)
    gain-=(float(len(subdata))/float(len(data_set))*sub_e)

  return gain

def entropy(data):
  pos=0
  neg=0
```

```
  # For the formula of entropy we need to see for how many of the +ve samples (yes) we have and how many -ve samples(no).
  for _, row in data.iterrows():
      if row['answer'] == "yes":
          pos += 1
      else:
          neg += 1
  if pos==0.0 or neg==0.0:
    return 0.0
  p=pos/(pos+neg)
  n=neg/(pos+neg)
  return -(p*math.log(p,2)+n*math.log(n,2))
main()
```

<u>Output:</u>

```
outlook
        overcast -> ['yes']

        rain
                wind
                        strong -> ['no']

                        weak -> ['yes']

        sunny
                humidity
                        high -> ['no']

                        normal -> ['yes']
```

4. 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 set

```
import math

import csv

import random

# This make sures that the dataset is in an ordered format. If we have some arbirary names in that column it difficult to deal with that.


def encode_class(dataset):

  classes=[]

  for i in range(len(dataset)):

    if dataset[i][-1] not in classes:

      classes.append(dataset[i][-1])


  # Looping across the classes which we have derived above.This will make sure that we have definitive classes (numeric) and not arbitrary

  for i in range(len(classes)):

    # Looping across all rows of dataset

    for j in range(len(dataset)):

      if dataset[j][-1] == classes[i]:

        dataset[j][-1]=i

  return dataset

# Splitting the data between training set and testing set. Normally its a general understanding the training:testing=7:3


def  train_test_split(dataset,ratio):

  test_num=int(ratio*len(dataset))

  train=list(dataset)

  test=[]

  for i in range(test_num):

    rand=random.randrange(len(train))

    test.append(train.pop(rand))
```

```
  return train,test
```

Now depending on resultant value (last column values), we need to group the rows. It will be usefult for calculating mean and std_dev

```
def groupUnderClass(train):
  dict={}
  for row in train:
    if row[-1] not in dict:
      dict[row[-1]]=[]
    dict[row[-1]].append(row)
  return dict
```

Standard formulae (just by-heart)

```
def mean(val):
  return sum(val)/float(len(val)) #Obvious


def stdDev(val):
  avg=mean(val)
  variance=sum([pow(x-avg,2) for x in val])/float(len(val)-1) # Especially this one
  return math.sqrt(variance)
```

We will calculte the mean and std dev with respect to each attribute. Important while calculating gaussian probablity

```
def meanStdDev(instances):
  info=[(mean(x),stdDev(x)) for x in zip(*instances)] # Here we are taking complete column's values of all instances.
  del info[-1]
  return info
```

As explained earlier why e need to group. We will be calculating the mean and std dev with respect each class.

```
def MeanAndStdDevForClass(train):
  info={}
  dictionary=groupUnderClass(train)
  # print(dictionary)
```

```
  for key,value in dictionary.items():
    # dictionary[key]=meanStdDev(value)
    info[key]=meanStdDev(value) #Here value stands for a complete group.
  return info
# Its a formula by heart (no choice)


def calculateGaussianProbablity(x,mean,std_dev):
  expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(std_dev, 2))))
  return (1 / (math.sqrt(2 * math.pi) * std_dev)) * expo
# After calculating mean and std dev w.r.t training data now its time to check if the logic will work on testing data



def calculateClassProbablities(info,ele):
  probablities={}
  for key,summaries in info.items(): # Info contains the groupName (key) and list of (mean,std_dev) for each attribute of that group
    probablities[key]=1
    for i in range(len(summaries)): #Loop across all attributes
      mean,std_dev=summaries[i]
      x=ele[i] # Testing data's one instance's attribute value.
      probablities[key] *= calculateGaussianProbablity(x, mean, std_dev)
  return probablities
def predict(info,ele):
  probablities=calculateClassProbablities(info,ele) # returns a dictionary of probablities for each group
  bestLabel,bestProb=None,-1
  # Consider group name whichever gives you the highest probablities for this instance of testing data
  for key,prob in probablities.items():
    if bestLabel==None or prob>bestProb:
      bestProb=prob
      bestLabel=key
  return bestLabel
# Loop across testing data and store the predicted result from our model in the list.
```

```
def getPredictions(info,test):
  predictions=[]
  for ele in test:
    result=predict(info,ele) # This will give you the group to which it will belong.
    predictions.append(result)
  return predictions
def check_accuracy(predictions,test):
  count=0
  for i in range(len(test)):
    if predictions[i]==test[i][-1]:
      count+=1
  return count/float(len(test))*100
filename="/content/bayes.csv"
dataset=csv.reader(open(filename))
dataset=list(dataset)
dataset=encode_class(dataset)
for i in range(len(dataset)):
  dataset[i]=[float(x) for x in dataset[i]]

ratio=0.3
print(len(dataset))
train,test=train_test_split(dataset,ratio)
info=MeanAndStdDevForClass(train)

predictions=getPredictions(info,test)
accuracy=check_accuracy(predictions,test)
accuracy
```

Output:

```
         291
Out[ ]:  60.91954022988506
```

5. Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

```
import csv
import math
def calculate(X,Y):
  sum_x=sum(X)
  sum_y=sum(Y)
  n=len(Y)
  sum_xy=0
  for i in range(len(X)):
    sum_xy+=X[i]*Y[i]
  sum_x2=sum([x**2 for x in X])
  denomin=float((n*sum_x2)-(sum_x**2))
  # y=y_intercept+slope*x
  y_intercept=float((sum_y*sum_x2)-(sum_x*sum_xy))/denomin
  slope=float((n*sum_xy)-(sum_x*sum_y))/denomin
  return slope,y_intercept
filename='/content/insurance.csv'
file=open(filename)
dataset=csv.reader(file)
dataset=list(dataset)
X=[]
Y=[]
for x in dataset:
X.append(x[3])
Y.append(x[len(x)-1])
print(dataset[0])
x_tag=str(X[0])
y_tag=str(Y[0])
X=X[1:200]
Y=Y[1:200]
X=[float(x) for x in X]
```

```
Y=[float(y) for y in Y]
# print(Y)
slope,y_intercept=calculate(X,Y)
print(slope,y_intercept)
```

```
['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']
299.40712303629675 12768.55599860939
```

```
import matplotlib.pyplot as plt
plt.scatter(X,Y,marker='o')
plt.xlabel(x_tag)
plt.ylabel(y_tag)
plt.title('Simple Linear Regression')
y_pred=[slope*x+y_intercept for x in X]
plt.plot(X,y_pred,color='red')
plt.show()
```