**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**



# LAB REPORT

**on**

**Machine Learning**

***Submitted by***

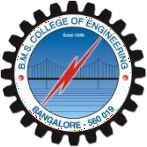
# SHASHWAT KHANNA (1BM19CS148)

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

# COMPUTER SCIENCE AND ENGINEERING



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

# BENGALURU-560019

**May-2022 to July-2022**

# B. M. S. College of Engineering,

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “MACHINE LEARNING” carried out by SHASHWAT KHANNA (1BM19CS148), who is Bonafede student of B. M.

S. College of Engineering. It is in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of Machine Learning- (20CS6PCOMD) work prescribed for the said degree.

SARITHA A N

PROFESSOR

BMSCE

BANGALURU

Lab Expt-1: Find S Algorithm

import csv;

a=[];

with open('/content/sample\_data/data-2.csv') as csvfile:

for x in csv.reader(csvfile):

a.append(x)

# print(a)'

msh=['0'] \* (len(a[0])-1)

a.remove(a[0])

for x in a:

if x[len(x)-1]=='yes':

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

if msh[i]=='0' or x[i]==msh[i]:

msh[i]=x[i]

else:

msh[i]='?'

print(msh)

Output:



Lab Expt-2: Candidate Elimination Algorithm

import csv

with open('/content/data-2.csv') as file:

  csv\_file=csv.reader(file)

  data=list(csv\_file)

  s=['0' for i in data[0]]

  s=s[:-1]

  g=[['?' for i in range(len(s))] for j in range(len(s))]

# print(s)

for row in data:

  if row[-1]=='yes':

    for j in range(len(s)):

      if s[j]!='0' and row[j]!=s[j]:

        s[j]='?'

        g[j][j]='?'

      elif s[j]=='0':

        s[j]=row[j]

    # print(s)

  elif row[-1]=='no':

    for j in range(len(s)):

      if s[j]!='0' and row[j]!=s[j]:

        g[j][j]=s[j]

      else:

        g[j][j]='?'

print(s)

print(g)

Output:



Lab Experiment-3: ID3 Algorithm

**import** pandas **as** pd

**import** numpy **as** np

**import** math

data**=**pd**.**read\_csv('/content/dataset-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

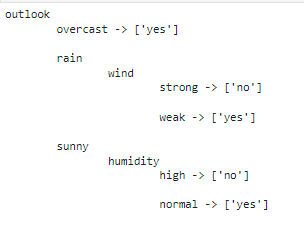
p**=**pos**/**(pos**+**neg)

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

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

main()

OUTPUT:



Lab Experiment-4: Naïve Bayes Algorithm

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)

y\_test=[]

for row in test:

  y\_test.append(row[-1])

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

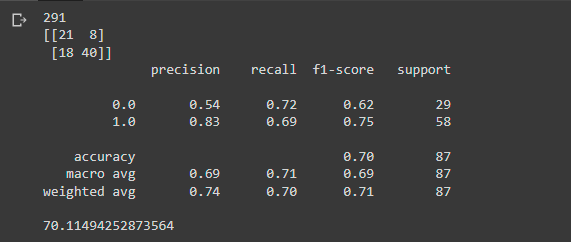
from sklearn.metrics import confusion\_matrix

print(confusion\_matrix(y\_test,predictions))

print(classification\_report(y\_test,predictions))

accuracy\_score(y\_test,predictions)\*100

Output:



Lab Expt-5: Linear Regression

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[2])

  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)

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()

Output

