VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

COURSE TITLE

Submitted by

MD IBADUDDIN SAFFAN (1BM19CS085)

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 "LAB COURSE **MACHINE LEARNING**" carried out by **MD IBADUDDIN SAFFAN (1BM19CS085),** who is bonafide 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 a **MACHINE LEARNING - 20CS6PCMAL** work prescribed for the said degree.

Name of the Lab-Incharge
Designation
Department of CSE
BMSCE, Bengaluru

PROF Saritha A N
Department of CSE
BMSCE, Bengaluru

`

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Find S	
2	Candidate Elimination Algorithm	
3	ID3	
4	Linear Regression	
5	Naive Bayes Classifier	

Course Outcome

CO1	Ability to apply the different learning algorithms.
CO2	Ability to analyze the learning techniques for given dataset.
CO3	Ability to design a model using machine learning to solve a problem.
CO4	Ability to conduct practical experiments to solve problems using appropriate machine learning techniques.

1. Find S

Code:

Implementation 1:

```
import pandas as pd
import numpy as np
df=pd.read_csv('./doc.csv')
d=np.array(df)

h=['!']*(m-1)

for i in range(len(d)):
    for j in range(len(d[0])):
        if(d[i][j]!=hypo[j]):
            hypo[j]="?"
print(h)
```

DATASET:

	Time	Weather	Temperature	Company	Humidity	Wind	Goes
0	Morning	Sunny	Warm	Yes	Mild	Strong	Yes
1	Evening	Rainy	Cold	No	Mild	Normal	No
2	Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
3	Evening	Sunny	Cold	Yes	High	Strong	Yes

OUTPUT:

```
['?' 'Sunny' '?' 'Yes' '?' '?']
```

Implementation 2

```
import pandas as pd
import numpy as np
n=int(input("Enter number of rows:"))
columns=['Time','Weather','Temperature','humidity','Enjoying?']
d=[]
print("Enter the data:\n")
for i in range(n):
    print("Enter Hypothesis:",i+1,"\n")
    temp=[]
    for x in columns:
        t=input("Enter value for: "+x+": ")
        temp.append(t)
    d.append(temp)
for x in d:
    print(x)
hypo=[]
for i in range(len(d[0])):
```

```
    hypo.append("?")
for i in range(len(d)):
    if d[i][len(d[0])-1]=='yes':
        hypo=d[i]
for i in range(len(d)):
    if d[i][len(d[0])-1]=='yes':
        for j in range(len(d[0])):
            if(d[i][j]!=hypo[j]):
                hypo[j]="?"
print(hypo)
```

DATASET:

	Time	Weather	Temperature	Company	Humidity	Wind	Goes
0	Morning	Sunny	Warm	Yes	Mild	Strong	Yes
1	Evening	Rainy	Cold	No	Mild	Normal	No
2	Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
3	Evening	Sunny	Cold	Yes	High	Strong	Yes

OUTPUT:

```
['?' 'Sunny' '?' 'Yes' '?' '?']
```

2. Candidate Elimination Algorithm

CODE

```
import numpy as np
import pandas as pd

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

def learn(concepts, target):
    specific_h = concepts[0].copy()
    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":
            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] = '?'

        print("Specific Boundary = ", specific_h)
        print("Generic Boundary = ", 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(" The Final Specific_h : ", s_final, sep="\n")
print("The Final General_h : ", g_final, sep="\n")
```

OUTPUT:

```
Specific Boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '
?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Specific Boundary =  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary =  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'
, '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Specific Boundary =  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary =  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'
, '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Specific Boundary =  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary =  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '
?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']
, ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Specific Boundary =  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary =  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '
?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']
, ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


 The Final Specific_h :
['sunny' 'warm' '?' 'strong' '?' '?']
The Final General_h :
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

3. ID3

IMPLEMENTATION 1:

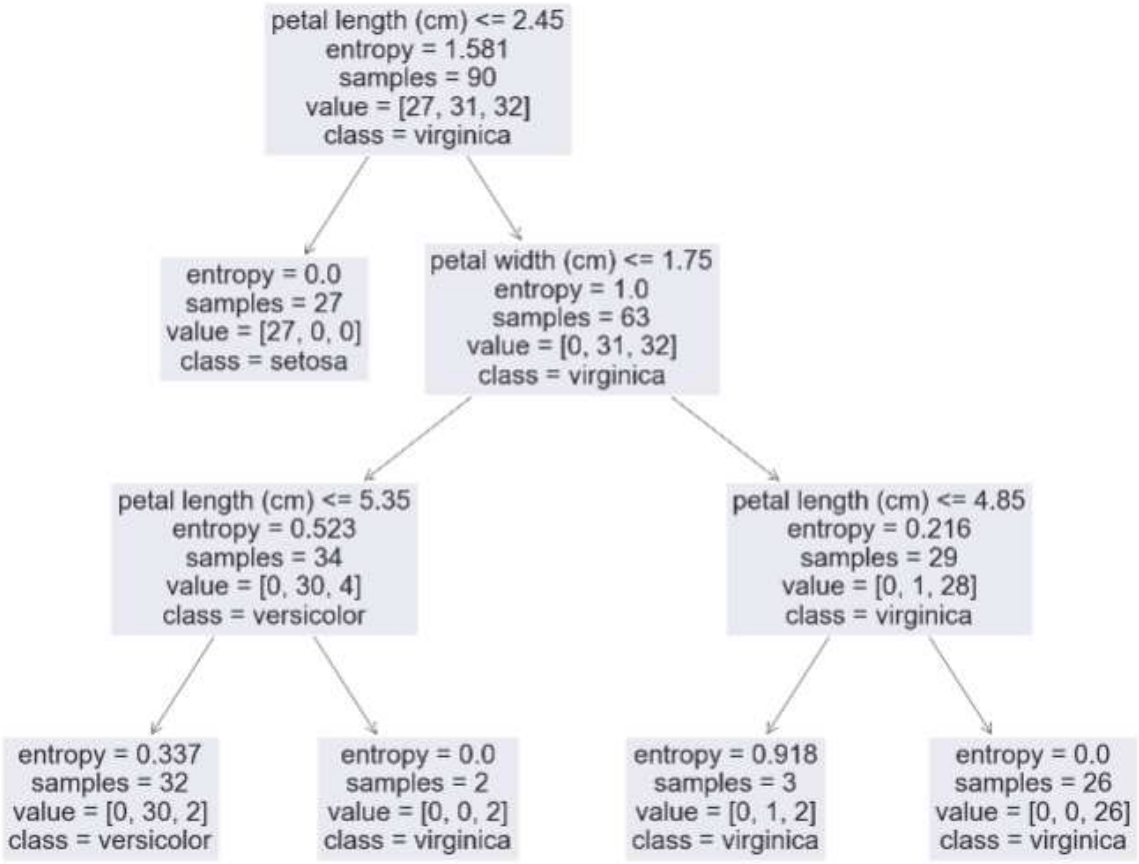
CODE:

```
import pandas as pd

import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

data = load_iris()

df = pd.DataFrame(data.data, columns = data.feature_names)

df.head()

df['Species'] = data.target

#replace this with the actual names

target = np.unique(data.target)

target_names = np.unique(data.target_names)

targets = dict(zip(target, target_names))

df['Species'] = df['Species'].replace(targets)

x = df.drop(columns="Species")

y = df["Species"]

feature_names = x.columns
labels = y.unique()

from sklearn.model_selection import train_test_split

X_train, test_x, y_train, test_lab = train_test_split(x,y,test_size =
0.4,random_state = 42)


from sklearn.tree import DecisionTreeClassifier
 clf = DecisionTreeClassifier(max_depth =3, random_state =
42,criterion='entropy')

clf.fit(X_train, y_train)


DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
test_pred = clf.predict(test_x)

clf.score(test_x,test_lab)
```

```
from sklearn import tree
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf,
                   feature_names=data.feature_names,
                   class_names=data.target_names)
```

OUTPUT:

```
Accuracy: 0.9833333333333333
```



IMPLEMENTATION 2:

```
import math
import csv
```

In [2]:

```
def load_csv(filename):
    lines=csv.reader(open(filename,"r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers

class Node:
    def __init__ (self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""
```

In [3]:

```
def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1
    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic
```

In [4]:

```
def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
```

```
        sums+=-1*cnt*math.log(cnt,2)
    return sums
```

In [5]:

```
def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy
```

In [6]:

```
def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]
    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node
```

In [7]:

```
def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)
```

In [8]:

```
def classify(node,x_test,features):
    if node.answer!="":
```

```
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)
```

In [9]:

```
'''Main program'''
dataset,features=load_csv("data.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("data.csv")
for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end=" ")
    classify(node1,xtest,features)
```

OUTPUT:

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   rain
     Wind
       strong
         no
       weak
         yes
   sunny
     Humidity
       high
         no
       normal
         yes
   overcast
     yes
The test instance: ['sunny', 'hot', 'high', 'weak', 'no']
The label for test instance: no
The test instance: ['sunny', 'hot', 'high', 'strong', 'no']
The label for test instance: no
The test instance: ['overcast', 'hot', 'high', 'weak', 'yes']
The label for test instance: yes
The test instance: ['rain', 'mild', 'high', 'weak', 'yes']
The label for test instance: yes
The test instance: ['rain', 'cool', 'normal', 'weak', 'yes']
The label for test instance: yes
The test instance: ['rain', 'cool', 'normal', 'strong', 'no']
The label for test instance: no
```

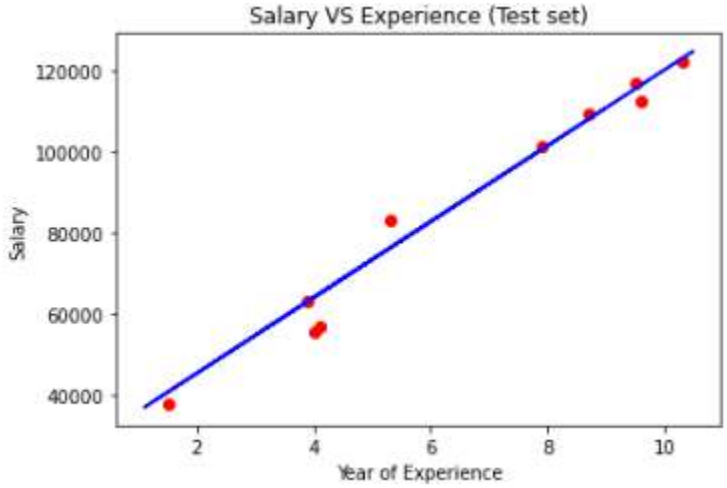
4. Linear Regression

CODE:

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


dataset = pd.read_csv('salary_data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3,
random_state=0)



# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

LinearRegression()
# Predicting the Test set results
y_pred = regressor.predict(X_test)



# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()



# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

OUTPUT:


Salary VS Experience (Training set)
120000
100000
80000
60000
40000
Salary
2
4
6
8
10
Year of Experience


Salary VS Experience (Test set)
120000
100000
80000
60000
40000
Salary
2
4
6
8
10
Year of Experience

5. Naive Bayes Classifier

CODE:

IMPLEMENTATION 1:

```
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
#read Cleveland Heart Disease data
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)
#display the data
print('Sample instances from the dataset are given below')
print(heartDisease.head())
#display the Attributes names and datatyes
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
#Creat Model- Bayesian Network
model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),(
'exang','heartdisease'),('cp','heartdisease'),('heartdisease',
'restecg'),('heartdisease','chol')])
#Learning CPDs using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
# Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
```

```
#computing the Probability of HeartDisease given restecg

print('\n 1.Probability of HeartDisease given evidence= restecg :1')

q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})

print(q1)

#computing the Probability of HeartDisease given cp

print('\n 2.Probability of HeartDisease given evidence= cp:2 ')

q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})

print(q2)
```

Implementation 2:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

df = pd.read_csv("pima_indian.csv")
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness',
'insulin', 'bmi', 'diab_pred', 'age']
predicted_class_names = ['diabetes']
X = df[feature_col_names].values
y = df[predicted_class_names].values
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)
```

In [19]:

```
df.head()
```

```
clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
```

In [30]:

```
metrics.confusion_matrix(ytest,predicted)
```

Out[30]:

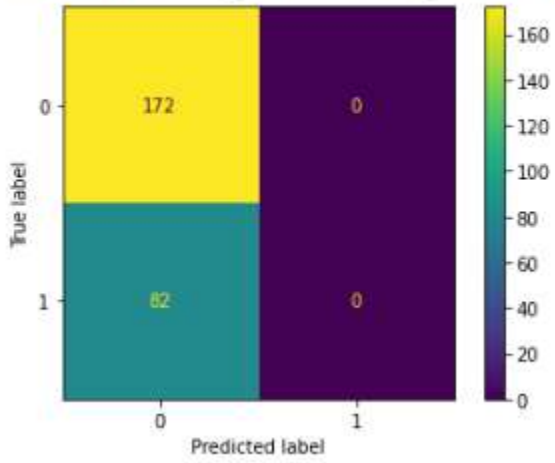
```
array([[139,  26],
       [ 33,  56]], dtype=int64)
```

In [28]:

```
print('\nConfusion matrix')

print(metrics.plot_confusion_matrix(clf,ytest,predicted))
print(metrics.classification_report(ytest,predicted))


print("Predicted Value for individual Test Data:", predictTestData)
```

OUTPUT

```
Confusion matrix
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay
```



	precision	recall	f1-score	support
0	0.81	0.84	0.82	165
1	0.68	0.63	0.65	89
accuracy			0.77	254
macro avg	0.75	0.74	0.74	254
weighted avg	0.76	0.77	0.77	254