VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING

Submitted by

S R POOJA (1BM19CS135)

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

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled "Machine Learning Lab" carried out by **S R Pooja (1BM19CS135),** who is a 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 with respect to **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Dr. Asha G R	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

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

CODE:

```
import csv

a = []

with open('/kaggle/input/dataset/data.csv','r') as csvfile:

    for row in csv.reader(csvfile):

        a.append(row)

        print(a)

    print("\n The total number of training instances are : ",len(a))

num_attribute = len(a[0])-1

print("\n The initial hypothesis is : ")

hypothesis = ['0']*num_attribute

print(hypothesis)

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

    if a[i][num_attribute] == 'yes':

        for j in range(0, num_attribute):

            if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:

                hypothesis[j] = a[i][j]

            else:

                hypothesis[j] = '?'

    print("\n The hypothesis for the training instance {} is :\n" .format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training instances is :")

print(hypothesis)
```

OUTPUT:

```
The total number of training instances are :  5

The initial hypothesis is :
['0', '0', '0', '0', '0', '0']

The hypothesis for the training instance 1 is :
['0', '0', '0', '0', '0', '0']

The hypothesis for the training instance 2 is :
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

The hypothesis for the training instance 3 is :
['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 4 is :
['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 5 is :
['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instances is :
['sunny', 'warm', '?', 'strong', '?', '?']
```

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.

CODE:

```
import numpy as np
import pandas as pd

data = pd.read_csv('/kaggle/input/dataset/data.csv')
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("Initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)

    for i, h in enumerate(concepts):
        print("For Loop Starts")
        if target[i] == "yes":
            print("If instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("If instance is Negative ")
            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("Steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
        print("\n")
        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("Final Specific_h:", s_final, sep="\n")
```

```python
print("Final General_h:", g_final, sep="\n")
```

OUTPUT :

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
Initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
Steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


For Loop Starts
If instance is Positive
Steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


For Loop Starts
If instance is Negative
Steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]


For Loop Starts
If instance is Positive
Steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General h:
```

```
For Loop Starts
If instance is Positive
Steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


For Loop Starts
If instance is Negative
Steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]


For Loop Starts
If instance is Positive
Steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


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

+ Code   + Markdown

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.

CODE:

```
import math
import csv
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=""

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

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:
```

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

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

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

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)


'''Main program'''
dataset,features=load_csv("/kaggle/input/train/ids_train.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is :\n")
print_tree(node1,0)
```

OUTPUT:

```
The decision tree for the dataset using ID3 algorithm is :

Outlook
    └ Rain
      Wind
          └ Weak
            Yes
          └ Strong
            No
    └ Sunny
      Humidity
          └ Normal
            Yes
          └ High
            No
    └ Overcast
      Yes
```

4. Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

CODE:

```python
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("/kaggle/input/diabetes/diabetes.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
print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.40)

print ('\n The total number of Training Data :',ytrain.shape)
print ('\n The total number of Test Data :',ytest.shape)

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

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
print('\n The value of Precision', metrics.precision_score(ytest,predicted))
print('\n The value of Recall', metrics.recall_score(ytest,predicted))
print("Predicted Value for individual Test Data:", predictTestData)
```

OUTPUT:

```
[145 rows x 9 columns]>

 The total number of Training Data : (87, 1)

 The total number of Test Data : (58, 1)

 Confusion matrix
[[31  7]
 [10 10]]

 Accuracy of the classifier is 0.7068965517241379

 The value of Precision 0.5882352941176471

 The value of Recall 0.5
Predicted Value for individual Test Data: [1]
```

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

CODE:

```python
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


# Importing the dataset
dataset = pd.read_csv('/kaggle/input/years-of-experience-and-salary/Years Experience and Salary.csv')

X = dataset.iloc[:, :-1].values #get a copy of dataset exclude last column

y = dataset.iloc[:, 1].values #get array of dataset in column 1st


# Splitting the dataset into the Training set and Test set
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)


# 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)

Salary VS Experience (Test set)