VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning

Submitted by

SaiPraveen Marni(1BM19CS138)

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**" carried out by **SaiPraveen Marni (1BM19CS138),** 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 in respect of a "**Machine Learning**"- **(20CS6PCMAL)** work prescribed for the said degree.

Dr. Asha G.R
Assistant Professor
Department of CSE
BMSCE, Bengaluru

    Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

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

```python
import pandas as pd
import numpy as np


data=pd.read_csv("data.csv")
print(data)


d=np.array(data)[:,:-1]
print("The attributes are : \n",d)
target=np.array(data)[:,-1]
print("\nThe target is : ",target)



def train(c,t):
  for i,val in enumerate(t):
    if val == "yes":
      specific_hypothesis=c[i].copy()
      break

  for i,val in enumerate(c):
    if t[i] == "yes":
      for x in range(len(specific_hypothesis)):
        if val[x] != specific_hypothesis[x]:
          specific_hypothesis[x] = '?'
        else:
          pass

  return specific_hypothesis


print("The final hypothesis is : ",train(d,target))
```

Output:

```
In [15]: import pandas as pd
         import numpy as np
```

```
In [16]: data=pd.read_csv("data.csv")
         print(data)
```

```
     sky air temp humidity   wind water forecast enjoy sport
0  sunny    warm   normal  strong  warm     same          yes
1  sunny    warm     high  strong  warm     same          yes
2  rainy    cold     high  strong  warm   change           no
3  sunny    warm     high  strong  cool   change          yes
```

```
In [17]: d=np.array(data)[:,:-1]
         print("The attributes are : \n",d)
         target=np.array(data)[:,-1]
         print("\nThe target is : ",target)
```

```
The attributes 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']]

The target is :  ['yes' 'yes' 'no' 'yes']
```

```
In [18]: def train(c,t):
             for i,val in enumerate(t):
                 if val == "yes":
                     specific_hypothesis=c[i].copy()
                     break

             for i,val in enumerate(c):
                 if t[i] == "yes":
                     for x in range(len(specific_hypothesis)):
                         if val[x] != specific_hypothesis[x]:
                             specific_hypothesis[x] = '?'
                         else:
                             pass

             return specific_hypothesis
```

```
In [19]: print("The final hypothesis is : ",train(d,target))
```

```
The final hypothesis is :  ['sunny' 'warm' '?' 'strong' '?' '?']
```

Prg 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

```python
import numpy as np
import pandas as pd

data = pd.read_csv('data.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: "0, 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":
            print("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("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("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 = learn(concepts, target)
print("Final Specific_h: ", s_final, sep="\n")
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_h and genearal_h

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', '?', '?', '?', '?']]
```

Prg3: 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.

```python
import pandas as pd
import math
import numpy as np

data = pd.read_csv("id3.csv")
features = [feat for feat in data]
features.remove("Answer")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["Answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain = entropy(examples)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    return gain

def ID3(examples, attrs):
    root = Node()
```

```python
        max_gain = 0
        max_feat = ""
        for feature in attrs:
            gain = info_gain(examples, feature)
            if gain > max_gain:
                max_gain = gain
                max_feat = feature
        root.value = max_feat
        uniq = np.unique(examples[max_feat])
        for u in uniq:
            subdata = examples[examples[max_feat] == u]
            if entropy(subdata) == 0.0:
                newNode = Node()
                newNode.isLeaf = True
                newNode.value = u
                newNode.pred = np.unique(subdata["Answer"])
                root.children.append(newNode)
            else:
                dummyNode = Node()
                dummyNode.value = u
                new_attrs = attrs.copy()
                new_attrs.remove(max_feat)
                child = ID3(subdata, new_attrs)
                dummyNode.children.append(child)
                root.children.append(dummyNode)
        return root

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)

root = ID3(data, features)
printTree(root)
```

Output:

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

        rain
                Wind
                        strong -> ['no']

                        weak -> ['yes']

        sunny
                Humidity
                        high -> ['no']

                        normal -> ['yes']
```

Prog 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 sets.

```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("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:

```
Confusion matrix
[[132  29]
 [ 45  48]]

Accuracy of the classifier is 0.7086614173228346

The value of Precision 0.6233766233766234

The value of Recall 0.5161290322580645
Predicted Value for individual Test Data: [1]
```

Prog 5: Implement the linear Regression Algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import r2_score
import plotly.express as px

dataset = pd.read_csv('salary_data.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

X_train, X_test, y_train, y_test = tts(X, y, test_size=1/3, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)
r2_score(y_test,y_pred)
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()
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)