VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning
(20CS6PCMAL)

Submitted by

S SKANDA (1BM19CS137)

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 **S SKANDA (1BM19CS137),** 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.

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

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

`

Index Sheet

1 Find S Algorithm

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

```
import pandas as pd
import numpy as np

#to read the data in the csv file
data =
pd.read_csv('D:/engineering/sem6/ML/LAB/1_Find_S/enjoysport.csv')
print(data)

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("The attributes are: ")
print(d)

#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("The target is: ")
print(target)

#training function to implement find-s algorithm
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

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

Output

```
PS D:\engineering\sem6\ML\LAB> python -u "d:\engineering\sem6\ML\LAB\1_Find_S\FindS.py"
     sky airtemp humidity    wind water forcast enjoysport
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
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']
The final hypothesis is:
['sunny' 'warm' '?' 'strong' '?' '?']
```

2 Candidate Elimination Algorithm

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

data=pd.read_csv('D:/engineering/sem6/ML/LAB/2_Candidate_Elimination/e
njoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print('Concepts:')
print(concepts)
target = np.array(data.iloc[:,-1])
print('Target:')
print(target)

def learn(concepts, target):
    print("Initialization of specific_h and general_h")

    specific_h = concepts[0].copy()
    print('\t specific_h:', specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print('\t general_h:', general_h)

    for i, h in enumerate(concepts):
        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('\n Steps of Candidate Elimination Algorithm : ', i+1)
```

```
        print('specific_h')
        print(specific_h)
        print('general_h:')
        print(general_h)

    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("\n Final specific_h:" )
print(s_final)
print("\n Final general_h:")
print(g_final)
```

Output

```
Target:
['yes' 'yes' 'no' 'yes']
Initialization of specific_h and general_h
('\t specific_h:', array(['sunny', 'warm', 'normal', 'strong', 'warm', 'same'], dtype=object))
('\t general_h:', [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '
?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']])
('\n Steps of Candidate Elimination Algorithm : ', 1)
specific_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
general_h:
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
('\n Steps of Candidate Elimination Algorithm : ', 2)
specific_h
['sunny' 'warm' '?' 'strong' 'warm' 'same']
general_h:
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
('\n Steps of Candidate Elimination Algorithm : ', 3)
specific_h
['sunny' 'warm' '?' 'strong' 'warm' 'same']
general_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
('\n Steps of Candidate Elimination Algorithm : ', 4)
specific_h
['sunny' 'warm' '?' 'strong' '?' '?']
general_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Final specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']

 Final general_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

3 ID3 Algorithm (Decision Tree)

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 math
import numpy as np

data = pd.read_csv("D:/engineering/sem6/ML/LAB/3_ID3/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])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    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
        #print ("\n",gain)
    return gain

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

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        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, depth=0):
```

```
    for i in range(depth):
        print("\t")
    print(root.value)
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

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

Output

```
The decision tree for the dataset using ID3 algorithm is:
 Outlook
        overcast
                > yes
        rain
                Wind
                        weak
                                > yes
                        strong
                                > no
        sunny
                Humidity
                        high
                                > no
                        normal
                                > yes
```

4 Naïve Bayesian Classifier

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

(Using Libraries)

```
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('D:/engineering/sem6/ML/LAB/4_Naive_Bayes/pima_indian.csv'
)
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp',
'thickness', 'insulin', 'bmi','age', 'diab_pred']
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

```
26          7          147           76          0        0  39.4
27          1           97           66         15      140  23.2
28         13          145           82         19      110  22.2
('\n the total number of Training Data :', (460L, 1L))
('\n the total number of Test Data :', (308L, 1L))

 Confusion matrix
[[164  42]
 [ 45  57]]
('\n Accuracy of the classifier is', 0.7175324675324676)
('\n The value of Precision', 0.5757575757575758)
('\n The value of Recall', 0.5588235294117647)
('Predicted Value for individual Test Data:', array([1], dtype=int64))
```

(Without using libraries)

```
import csv
import random
import math

def loadcsv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    for i in range(len(dataset)):
     #converting strings into numbers for processing
        dataset[i] = [float(x) for x in dataset[i]]

    return dataset

def splitdataset(dataset, splitratio):
     #67% training size
    trainsize = int(len(dataset) * splitratio)
    trainset = []
    copy = list(dataset)
    while len(trainset) < trainsize:
        index = random.randrange(len(copy))
        trainset.append(copy.pop(index))
    return [trainset, copy]

def separatebyclass(dataset):
    separated = {} #dictionary of classes 1 and 0
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
```

```
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated

def mean(numbers):

    return sum(list(numbers))/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in
numbers])/float(len(numbers)-1)
    return math.sqrt(variance)

def summarize(dataset): #creates a dictionary of classes
    summaries =    [(mean(attribute), stdev(attribute))  for
attribute in zip(*dataset)]
    del summaries[-1] #excluding labels +ve or -ve
    return summaries

def summarizebyclass(dataset):
    separated = separatebyclass(dataset); #print(separated)
    summaries = {}
    for classvalue, instances in separated.items():
        summaries[classvalue] = summarize(instances) #summarize is
used to cal to mean and std
    return summaries

def calculateprobability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateclassprobabilities(summaries, inputvector):
# probabilities contains the all prob of all class of test data
    probabilities = {}
    for classvalue, classsummaries in summaries.items(): #class and
attribute information as mean and sd
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i] #take mean and sd of every
attribute for class 0 and 1 seperaely
            x = inputvector[i] #testvector's first attribute
```

```
            probabilities[classvalue] *= calculateprobability(x, mean,
stdev);#use normal dist
    return probabilities

def predict(summaries, inputvector): #training and test data is passed
    probabilities = calculateclassprobabilities(summaries,
inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items(): #assigns
that class which has the highest prob
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel


def getpredictions(summaries, testset):
    predictions = []
    for i in range(len(testset)):
        result = predict(summaries, testset[i])
        predictions.append(result)
    return predictions

def getaccuracy(testset, predictions):
    correct = 0
    for i in range(len(testset)):
        if testset[i][-1] == predictions[i]:
            correct += 1
    return (correct/float(len(testset))) * 100.0

def main():
    filename ='D:/engineering/sem6/ML/LAB/4_Naive_Bayes/naivedata.csv'
    splitratio = 0.67
    dataset = loadcsv(filename)
    trainingset, testset = splitdataset(dataset, splitratio)
    print('Split {0} rows into train={1} and
test={2}rows'.format(len(dataset), len(trainingset), len(testset))) #
prepare model
    summaries = summarizebyclass(trainingset); #print(summaries)
# test model
    predictions = getpredictions(summaries, testset) #find the
predictions of test data with the training data
```

```
    accuracy = getaccuracy(testset, predictions)
    print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

Output

```
PS D:\engineering\sem6\ML\LAB> python -u "d:\engineering\sem6\ML\LAB\4_Naive_Bayes\Naive_Bayes.py"
Split 768 rows into train=514 and test=254rows
Accuracy of the classifier is : 75.5905511811%
```

5 Linear Regression

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

```
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)

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


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