VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING
(20CS6PCMAL)

Submitted by

SAKSHI P KHANDOBA (1BM19CS139)

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 **SAKSHI P KHANDOBA (1BM19CS139),** 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 **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Saritha A. N
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

`

Index Sheet

Sl. No.	Experiment Title	Page No.
1.	Find-S	
2.	Candidate Elimination	
3.	Decision tree based on ID3	
4.	Naive Bayesian Classifier	
5.	Linear Regression	

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

Code:

```
import pandas as pd
import numpy as np

data = pd.read_csv("ENJOYSPORT.csv")
print(data,"\n")

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

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

global specific_hypothesis

def findS(c,t):

    for i, val in enumerate(t):
        if val == 1:
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == 1:
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

print("\n The final hypothesis is:",findS(d,target))
```

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.DataFrame(data=pd.read_csv('ENJOYSPORT.csv'))
print(data)
print()

concepts = np.array(data.iloc[:,0:-1])
print(concepts)

target = np.array(data.iloc[:,-1])
print("\nTarget: ",target)

def learn(concepts, target):
    print("\nInitialization of specific_hypothesis and general_hypothesis:")
    specific_h = concepts[0].copy()
    print("\nSpecific Hypothesis: ",specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneral Hypothesis: ",general_h)
    print("\nSteps of Candidate Elimination Algorithm: \n")
    for i, h in enumerate(concepts):
        if target[i] == 1:
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] = specific_h[x]
                    print("Specific: ",specific_h)
                    print("General: ",general_h)
                    print()
        if target[i] == 0:
            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: ",specific_h)
                    print("General: ",general_h)
                    print()
    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 Hypothesis:", s_final, sep="\n")
print("Final General Hypothesis:", 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', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'
e']]



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



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

```
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', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?
e']]
```

```
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:
        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/id3-dataset/id3 dataset.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 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.

Code:

```
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-data/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:

```
<bound method NDFrame.head of      num_preg  glucose_conc  diastolic_bp  thickness  insulin   bmi  \
0           6           148            72         35        0  33.6
1           1            85            66         29        0  26.6
2           8           183            64          0        0  23.3
3           1            89            66         23       94  28.1
4           0           137            40         35      168  43.1
..        ...           ...           ...        ...      ...   ...
140         3           128            78          0        0  21.1
141         5           106            82         30        0  39.5
142         2           108            52         26       63  32.5
143        10           108            66          0        0  32.4
144         4           154            62         31      284  32.8

     diab_pred  age  diabetes
0        0.627   50         1
1        0.351   31         0
2        0.672   32         1
3        0.167   21         0
4        2.288   33         1
..         ...  ...       ...
140      0.268   55         0
141      0.286   38         0
142      0.318   22         0
143      0.272   42         1
144      0.237   23         0

[145 rows x 9 columns]>

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

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

 Confusion matrix
[[34  5]
 [ 8 11]]

 Accuracy of the classifier is 0.7758620689655172

 The value of Precision 0.6875

 The value of Recall 0.5789473684210527
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:

Linear Regression with dataset:

```
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()
```

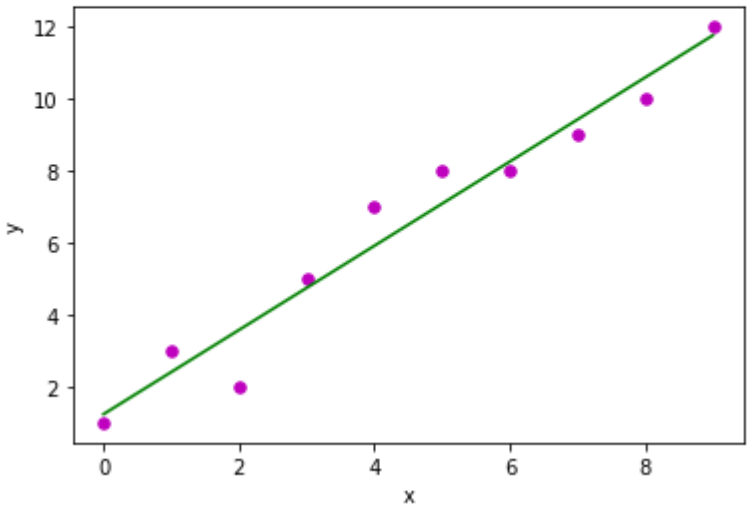
Linear Regression:

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

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')
```

```
        # function to show plot
        plt.show()

    def main():
        # observations / data
        x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
        y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

        # estimating coefficients
        b = estimate_coef(x, y)
        print("Estimated coefficients:\nb_0 = {} \
            \nb_1 = {}".format(b[0], b[1]))

        # plotting regression line
        plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

Output:

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```




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