

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT**  
**on**

## **MACHINE LEARNING** **(20CS6PCMAL)**

*Submitted by*

S SHREE LAKSHMI (1BM19CS136)

*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**” was carried out by **S SHREE LAKSHMI(1BM19CS136)**, who is a bona fide 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 the course **MACHINE LEARNING (20CS6PCMAL)** work prescribed for the said degree.

Name of the Lab-In charge  
Designation  
Department of CSE  
BMSCE, Bengaluru

**DR. ASHA G R**  
Assistant Professor  
Department of CSE  
BMSCE, Bengaluru

## Index Sheet

Sl. No.	Experiment Title	Page No.
1.	<b><u>FIND-S</u></b> :- Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.	2
2.	<b><u>CANDIDATE ELIMINATION</u></b> :- 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.	9
3.	<b><u>ID3</u></b> :- 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.	18
4.a.	<b><u>NAÏVE BAYESIAN CLASSIFIER</u></b> :- 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	31
4.b.	<b><u>NAÏVE BAYESIAN CLASSIFIER (Without packages)</u></b> :- 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.(Without packages)	39
5.	<b><u>LINEAR REGRESSION</u></b> :- Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	54

### Course Outcome :-

***At the end of the course the student will be able to***

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.

## Lab Program -1 :-

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

### Source code and output :-

```

+*In[1]:*+

```

```

[source, ipython3]

```

```

----

```

```

import csv

```

```

hypo = ['%', '%', '%', '%', '%', '%'];

```

```

with open(r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\lab 1\finds.csv') as csv_file:

```

```

    readcsv = csv.reader(csv_file, delimiter=',')

```

```

    print(readcsv)

```

```

data = []

```

```

print("\nThe given training examples are:")

```

```

for row in readcsv:

```

```

    print(row)

```

```

    if row[len(row)-1].upper() == "YES":

```

```

        data.append(row)

```

```

----

```

```

+*Out[1]:*+

```

```

----

```

```

<_csv.reader object at 0x0000013B7E4DFD60>

```

The given training examples are:

```
['sky', 'air temp', 'humidity', 'wind', 'water', 'forecast', 'enjoy sport']
```

```
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
```

```
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
```

```
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
```

```
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']
```

----

```
+*In[2]:*+
```

```
[source, ipython3]
```

----

```
print("\nThe positive examples are:");
```

```
for x in data:
```

```
    print(x);
```

```
print("\n");
```

----

```
+*Out[2]:*+
```

----

The positive examples are:

```
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
```

```
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
```

```
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']
```

----

+\*ln[3]:\*+

[source, ipython3]

----

TotalExamples = len(data);

i=0;

j=0;

k=0;

print("The steps of the Find-s algorithm are :\n",hypo);

list = [];

p=0;

d=len(data[p])-1;

for j in range(d):

list.append(data[i][j]);

hypo=list;

i=1;

for i in range>TotalExamples):

for k in range(d):

if hypo[k]!=data[i][k]:

hypo[k]='?';

k=k+1;

else:

hypo[k];

print(hypo);

i=i+1;

----

```
+*Out[3]:*+
```

```
----
```

The steps of the Find-s algorithm are :

```
['%', '%', '%', '%', '%', '%']
```

```
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
```

```
['sunny', 'warm', '?', 'strong', 'warm', 'same']
```

```
['sunny', 'warm', '?', 'strong', '?', '?']
```

```
----
```

```
+*In[4]:*+
```

```
[source, ipython3]
```

```
----
```

```
print("\nThe maximally specific Find-s hypothesis for the given training examples is :");
```

```
list=[];
```

```
for i in range(d):
```

```
    list.append(hypo[i]);
```

```
print(list);
```

```
----
```

```
+*Out[4]:*+
```

```
----
```

The maximally specific Find-s hypothesis for the given training examples is :

```
['sunny', 'warm', '?', 'strong', '?', '?']
```

```
----
```

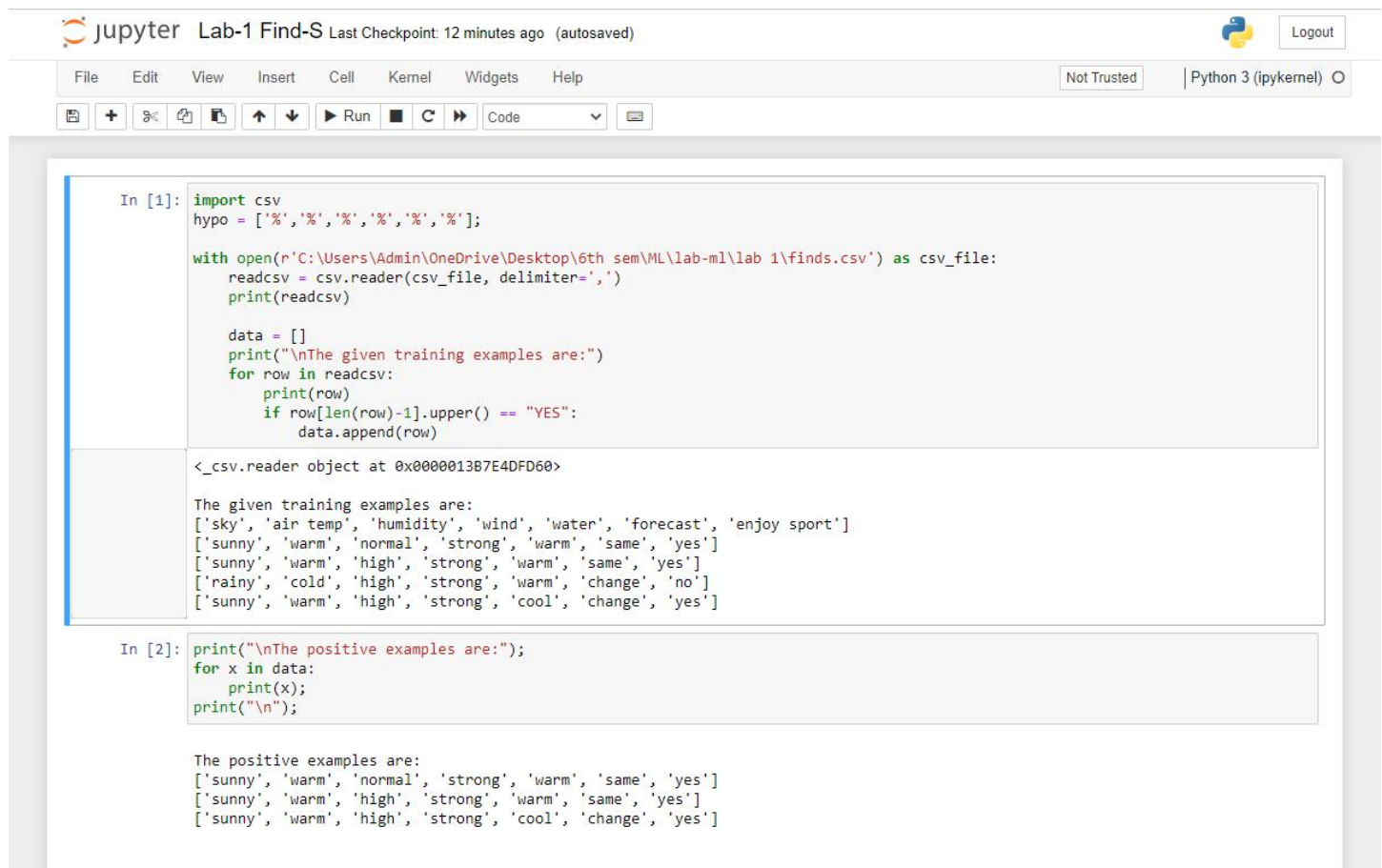
+\*In[ ]:\*

[source, ipython3]

----

----

### Output screenshots :-



The screenshot displays a JupyterLab environment with the title 'Lab-1 Find-S'. The top bar shows the Jupyter logo, the title, and a 'Last Checkpoint: 12 minutes ago (autosaved)' status. On the right, there is a 'Logout' button and a Python 3 (ipykernel) indicator. The main interface features a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. Two code cells are visible. The first cell, labeled 'In [1]:', contains Python code that imports the 'csv' module, defines a hypothesis string, opens a CSV file at 'r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\lab 1\finds.csv'', and reads its contents. It then prints the data and filters for rows where the last element is 'YES', appending them to a list. The output of this cell shows the CSV reader object and a list of training examples. The second cell, labeled 'In [2]:', prints the positive examples from the list. Its output shows the filtered list of examples.

```
In [1]: import csv
hypo = ['%', '%', '%', '%', '%', '%'];

with open(r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\lab 1\finds.csv') as csv_file:
    readcsv = csv.reader(csv_file, delimiter=',')
    print(readcsv)

    data = []
    print("\nThe given training examples are:")
    for row in readcsv:
        print(row)
        if row[len(row)-1].upper() == "YES":
            data.append(row)

<_csv.reader object at 0x0000013B7E4DFD60>

The given training examples are:
['sky', 'air temp', 'humidity', 'wind', 'water', 'forecast', 'enjoy sport']
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

In [2]: print("\nThe positive examples are:");
for x in data:
    print(x);
print("\n");

The positive examples are:
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']
```





```
In [3]: TotalExamples = len(data);
i=0;
j=0;
k=0;
print("The steps of the Find-s algorithm are :\n",hypo);
list = [];
p=0;
d=len(data[p])-1;
for j in range(d):
    list.append(data[i][j]);
hypo=list;
i=1;
for i in range(TotalExamples):
    for k in range(d):
        if hypo[k]!=data[i][k]:
            hypo[k]='?';
            k=k+1;
        else:
            hypo[k];
    print(hypo);
i=i+1;
```

The steps of the Find-s algorithm are :

```
['%', '%', '%', '%', '%', '%']
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
['sunny', 'warm', '?', 'strong', 'warm', 'same']
['sunny', 'warm', '?', 'strong', '?', '?']
```

```
In [4]: print("\nThe maximally specific Find-s hypothesis for the given training examples is :");
list=[];
for i in range(d):
    list.append(hypo[i]);
print(list);
```

The maximally specific Find-s hypothesis for the given training examples is :

```
['sunny', 'warm', '?', 'strong', '?', '?']
```

In [ ]:

	A	B	C	D	E	F	G	H
1	sky	air temp	humidity	wind	water	forecast	enjoy sport	
2	sunny	warm	normal	strong	warm	same	yes	
3	sunny	warm	high	strong	warm	same	yes	
4	rainy	cold	high	strong	warm	change	no	
5	sunny	warm	high	strong	cool	change	yes	
6								
7								
8								

## Lab Program -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.

### Source code and output :-

```
+*In[7]:*+
```

```
[source, ipython3]
```

```
----
```

```
import numpy as np
```

```
import pandas as pd
```

```
----
```

```
+*In[10]:*+
```

```
[source, ipython3]
```

```
----
```

```
# Loading Data from a CSV File
```

```
data = pd.DataFrame(data=pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\6th  
sem\ML\lab-ml\lab 2\trainingdata.csv'))
```

```
print(data)
```

```
----
```

```
+*Out[10]:*+
```

----

sky airtemp humidity wind water forecast 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

----

+\*In[11]:\*+

[source, ipython3]

----

# Separating concept features from Target

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

print(concepts)

----

+\*Out[11]:\*+

----

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']

['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']

----

```
+*In[12]:*+
```

```
[source, ipython3]
```

```
----
```

```
# Isolating target into a separate DataFrame
```

```
# copying last column to target array
```

```
target = np.array(data.iloc[:,-1])
```

```
print(target)
```

```
----
```

```
+*Out[12]:*+
```

```
----
```

```
['Yes' 'Yes' 'No' 'Yes']
```

```
----
```

```
+*In[13]:*+
```

```
[source, ipython3]
```

```
----
```

```
def learn(concepts, target):
```

```
'''
```

learn() function implements the learning method of the Candidate elimination algorithm.

Arguments:

concepts - a data frame with all the features

target - a data frame with corresponding output values

'''

# Initialise S0 with the first instance from concepts

# .copy() makes sure a new list is created instead of just pointing to the same memory location

specific\_h = concepts[0].copy()

print("\nInitialization of specific\_h and general\_h")

print(specific\_h)

#h=["#" for i in range(0,5)]

#print(h)

general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

print(general\_h)

# The learning iterations

for i, h in enumerate(concepts):

# Checking if the hypothesis has a positive target

if target[i] == "Yes":

for x in range(len(specific\_h)):

# Change values in S & G only if values change

if h[x] != specific\_h[x]:

specific\_h[x] = '?'

general\_h[x][x] = '?'

# Checking if the hypothesis has a positive target

```

if target[i] == "No":
    for x in range(len(specific_h)):
        # For negative hyposthesis change values only in G
        if h[x] != specific_h[x]:
            general_h[x][x] = specific_h[x]
        else:
            general_h[x][x] = '?'

```

```

print("\nSteps of Candidate Elimination Algorithm",i+1)
print(specific_h)
print(general_h)

```

```

# find indices where we have empty rows, meaning those that are unchanged
indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
for i in indices:
    # remove those rows from general_h
    general_h.remove(['?', '?', '?', '?', '?', '?'])
# Return final values
return specific_h, general_h

```

----

+\*In[14]:\*+

[source, ipython3]

----

s\_final, g\_final = learn(concepts, target)

```
print("\nFinal Specific_h:", s_final, sep="\n")
```

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

```
----
```

```
+*Out[14]:*+
```

```
----
```

Initialization of specific\_h and general\_h

```
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
```

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

Steps of Candidate Elimination Algorithm 1

```
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
```

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

Steps of Candidate Elimination Algorithm 2

```
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
```

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

Steps of Candidate Elimination Algorithm 3

```
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
```

```
[[ 'Sunny', '?', '?', '?', '?', '?' ], [ '?', 'Warm', '?', '?', '?', '?' ], [ '?', '?', '?', '?', '?', '?' ], [ '?', '?', '?', '?', '?',  
'?' ], [ '?', '?', '?', '?', '?', '?' ], [ '?', '?', '?', '?', '?', 'Same' ]]
```

## Steps of Candidate Elimination Algorithm 4

['Sunny' 'Warm' '?' 'Strong' '?' '?']

[[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]]

Final Specific\_h:

['Sunny' 'Warm' '?' 'Strong' '?' '?']

Final General\_h:

```
[[ 'Sunny', '?', '?', '?', '?', '?' ], [ '?', 'Warm', '?', '?', '?', '?' ]]
```

\_\_\_\_\_

$$+^*\ln[ ]:^*+$$

```
[source, ipython3]
```

— — — —

— — — —

**Output screenshots :-**



```
In [7]: import numpy as np
import pandas as pd
```

```
In [10]: # Loading Data from a CSV File
data = pd.DataFrame(data=pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\lab 2\trainingdata.csv'))
print(data)
```

	sky	airtemp	humidity	wind	water	forecast	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

```
In [11]: # Separating concept features from Target
concepts = np.array(data.iloc[:,0:-1])
print(concepts)

[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

```
In [12]: # Isolating target into a separate DataFrame
# copying last column to target array
target = np.array(data.iloc[:, -1])
print(target)

['Yes' 'Yes' 'No' 'Yes']
```

```
[ 'Yes' 'Yes' 'No' 'Yes']
```

```
In [13]: def learn(concepts, target):

    ...
    learn() function implements the learning method of the Candidate elimination algorithm.
    Arguments:
        concepts - a data frame with all the features
        target - a data frame with corresponding output values
    ...

    # Initialise S0 with the first instance from concepts
    # .copy() makes sure a new List is created instead of just pointing to the same memory location
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and general_h")
    print(specific_h)
    #h=["#" for i in range(0,5)]
    #print(h)

    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)
    # The Learning iterations
    for i, h in enumerate(concepts):

        # Checking if the hypothesis has a positive target
        if target[i] == "Yes":
            for x in range(len(specific_h)):

                # Change values in S & G only if values change
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

        # Checking if the hypothesis has a positive target
        if target[i] == "No":
            for x in range(len(specific_h)):
                # For negative hypothesis change values only in G
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

    print("\nSteps of Candidate Elimination Algorithm",i+1)
```

```

Initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 1
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Steps of Candidate Elimination Algorithm 4
['Sunny' 'Warm' '?' 'Strong' '?' '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']

Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

```

[illegible]

### Lab Program -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.

#### Source code and output :-

```
+#In[1]:*+
```

```
[source, ipython3]
```

```
----
```

```
import numpy as np
```

```
import math
```

```
import csv
```

```
----
```

```
+#In[2]:*+
```

```
[source, ipython3]
```

```
----
```

```
def read_data(filename):
```

```
    with open(filename, 'r') as csvfile:
```

```
        datareader = csv.reader(csvfile, delimiter=',')
```

```
        headers = next(datareader)
```

```
        metadata = []
```

```
        traindata = []
```

```
        for name in headers:
```

```
            metadata.append(name)
```

```
for row in datareader:
    traindata.append(row)

return (metadata, traindata)
```

----

```
+*In[5]:*+
[source, ipython3]
```

----

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

    def __str__(self):
        return self.attribute
```

----

```
+*In[6]:*+
[source, ipython3]
```

----

```
def subtables(data, col, delete):
    dict = {}
```

```
items = np.unique(data[:, col])
count = np.zeros((items.shape[0], 1), dtype=np.int32)
```

```
for x in range(items.shape[0]):
    for y in range(data.shape[0]):
        if data[y, col] == items[x]:
            count[x] += 1
```

```
for x in range(items.shape[0]):
    dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")
    pos = 0
    for y in range(data.shape[0]):
        if data[y, col] == items[x]:
            dict[items[x]][pos] = data[y]
            pos += 1
    if delete:
        dict[items[x]] = np.delete(dict[items[x]], col, 1)
```

```
return items, dict
```

```
----
```

```
+*ln[7]:*+
```

```
[source, ipython3]
```

```
----
```

```
def entropy(S):
```

```
items = np.unique(S)
```

```
if items.size == 1:
```

```
    return 0
```

```
counts = np.zeros((items.shape[0], 1))
```

```
sums = 0
```

```
for x in range(items.shape[0]):
```

```
    counts[x] = sum(S == items[x]) / (S.size * 1.0)
```

```
for count in counts:
```

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

```
return sums
```

```
----
```

```
.*ln[8].*+
```

```
[source, ipython3]
```

```
----
```

```
def gain_ratio(data, col):
```

```
    items, dict = subtables(data, col, delete=False)
```

```
    total_size = data.shape[0]
```

```
    entropies = np.zeros((items.shape[0], 1))
```

```
    intrinsic = np.zeros((items.shape[0], 1))
```

```

for x in range(items.shape[0]):
    ratio = dict[items[x]].shape[0]/(total_size * 1.0)
    entropies[x] = ratio * entropy(dict[items[x]][:, -1])
    intrinsic[x] = ratio * math.log(ratio, 2)

```

```

total_entropy = entropy(data[:, -1])

```

```

iv = -1 * sum(intrinsic)

```

```

for x in range(entropies.shape[0]):

```

```

    total_entropy -= entropies[x]

```

```

return total_entropy / iv

```

```

----

```

```

+*ln[9]:*+

```

```

[source, ipython3]

```

```

----

```

```

def create_node(data, metadata):

```

```

    if (np.unique(data[:, -1])).shape[0] == 1:

```

```

        node = Node("")

```

```

        node.answer = np.unique(data[:, -1])[0]

```

```

        return node

```

```

gains = np.zeros((data.shape[1] - 1, 1))

```

```
for col in range(data.shape[1] - 1):
    gains[col] = gain_ratio(data, col)
```

```
split = np.argmax(gains)
```

```
node = Node(metadata[split])
metadata = np.delete(metadata, split, 0)
```

```
items, dict = subtables(data, split, delete=True)
```

```
for x in range(items.shape[0]):
    child = create_node(dict[items[x]], metadata)
    node.children.append((items[x], child))
```

```
return node
```

```
----
```

```
+*ln[10]:*+
```

```
[source, ipython3]
```

```
----
```

```
def empty(size):
```

```
    s = ""
```

```
    for x in range(size):
```

```
        s += " "
```



```
return s
```

```
def print_tree(node, level):  
    if node.answer != "":  
        print(empty(level), node.answer)  
        return  
    print(empty(level), node.attribute)  
    for value, n in node.children:  
        print(empty(level + 1), value)  
        print_tree(n, level + 2)
```

```
----
```

```
+*In[11]:*+
```

```
[source, ipython3]
```

```
----
```

```
metadata, traindata = read_data(r"C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-  
ml\Lab 3\id3 training dataset.csv")  
data = np.array(traindata)  
node = create_node(data, metadata)  
print_tree(node, 0)
```

```
----
```

```
+*Out[11]:*+
```

```
----
```

Outlook

overcast

b'yes'

rain

Wind

b'strong'

b'no'

b'weak'

b'yes'

sunny

Humidity

b'high'

b'no'

b'normal'

b'yes'

----

+\*ln[ ]:\*+

[source, ipython3]

----

----

**Output screenshots :-**



```
In [1]: import numpy as np
import math
import csv
```

```
In [2]: def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)
```

```
In [5]: class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

    def __str__(self):
        return self.attribute
```

```
In [6]: def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="<S32")
        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
    if delete:
```

```
dict[items[x]] = np.delete(dict[items[x]], col, 1)

return items, dict
```

```
In [7]: def entropy(S):
        items = np.unique(S)

        if items.size == 1:
            return 0

        counts = np.zeros((items.shape[0], 1))
        sums = 0

        for x in range(items.shape[0]):
            counts[x] = sum(S == items[x]) / (S.size * 1.0)

        for count in counts:
            sums += -1 * count * math.log(count, 2)
        return sums
```

```
In [8]: def gain_ratio(data, col):
        items, dict = subtables(data, col, delete=False)

        total_size = data.shape[0]
        entropies = np.zeros((items.shape[0], 1))
        intrinsic = np.zeros((items.shape[0], 1))

        for x in range(items.shape[0]):
            ratio = dict[items[x]].shape[0]/(total_size * 1.0)
            entropies[x] = ratio * entropy(dict[items[x]][:-1])
            intrinsic[x] = ratio * math.log(ratio, 2)

        total_entropy = entropy(data[:, -1])
        iv = -1 * sum(intrinsic)

        for x in range(entropies.shape[0]):
            total_entropy -= entropies[x]

        return total_entropy / iv
```

```

In [9]: def create_node(data, metadata):
        if (np.unique(data[:, -1])).shape[0] == 1:
            node = Node("")
            node.answer = np.unique(data[:, -1])[0]
            return node

        gains = np.zeros((data.shape[1] - 1, 1))

        for col in range(data.shape[1] - 1):
            gains[col] = gain_ratio(data, col)

        split = np.argmax(gains)

        node = Node(metadata[split])
        metadata = np.delete(metadata, split, 0)

        items, dict = subtables(data, split, delete=True)

        for x in range(items.shape[0]):
            child = create_node(dict[items[x]], metadata)
            node.children.append((items[x], child))

        return node

```

Jupyter Lab-3 ID3 1BM19CS159 Last Checkpoint: 6 minutes ago (autosaved)



Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3 (ipykernel) O

Code

```

In [10]: def empty(size):
        s = ""
        for x in range(size):
            s += " "
        return s

        def print_tree(node, level):
            if node.answer != "":
                print(empty(level), node.answer)
                return
            print(empty(level), node.attribute)
            for value, n in node.children:
                print(empty(level + 1), value)
                print_tree(n, level + 2)

In [11]: metadata, traindata = read_data(r"C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 3\id3 training dataset.csv")
        data = np.array(traindata)
        node = create_node(data, metadata)
        print_tree(node, 0)

```

```

Outlook
  overcast
    b'yes'
  rain
    Wind
      b'strong'
      b'no'
      b'weak'
      b'yes'
    sunny
      Humidity
        b'high'
        b'no'
        b'normal'
        b'yes'

```

In [ ]:

A1							
Outlook							
	A	B	C	D	E	F	G
1	Outlook	Temperat	Humidity	Wind	Answer		
2	sunny	hot	high	weak	no		
3	sunny	hot	high	strong	no		
4	overcast	hot	high	weak	yes		
5	rain	mild	high	weak	yes		
6	rain	cool	normal	weak	yes		
7	rain	cool	normal	strong	no		
8	overcast	cool	normal	strong	yes		
9	sunny	mild	high	weak	no		
10	sunny	cool	normal	weak	yes		
11	rain	mild	normal	weak	yes		
12	sunny	mild	normal	strong	yes		
13	overcast	mild	high	strong	yes		
14	overcast	hot	normal	weak	yes		
15	rain	mild	high	strong	no		
16							
17							
18							

### Lab Program -4.a.-

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

#### Source code and output :-

```
+*In[1]:*+
```

```
[source, ipython3]
```

```
----
```

```
# import necessary libarities
```

```
import pandas as pd
```

```
from sklearn import tree
```

```
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.naive_bayes import GaussianNB
```

```
# load data from CSV
```

```
data = pd.read_csv(r"C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 4\Naive  
Bayesian classifier training dataset.csv")
```

```
print("The first 5 values of data is :\n",data.head())
```

```
----
```

```
+*Out[1]:*+
```

```
----
```

```
The first 5 values of data is :
```

```
    Outlook Temperature Humidity Windy PlayTennis
```

0	Sunny	Hot	High	False	No
1	Sunny	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Rainy	Mild	High	False	Yes
4	Rainy	Cool	Normal	False	Yes

----

```
+*In[2]:*+
```

```
[source, ipython3]
```

----

```
# obtain Train data and Train output
```

```
X = data.iloc[:, :-1]
```

```
print("\nThe First 5 values of train data is\n", X.head())
```

----

```
+*Out[2]:*+
```

----

The First 5 values of train data is

	Outlook	Temperature	Humidity	Windy
--	---------	-------------	----------	-------

0	Sunny	Hot	High	False
1	Sunny	Hot	High	True
2	Overcast	Hot	High	False
3	Rainy	Mild	High	False



```
4    Rainy    Cool    Normal    False
```

```
----
```

```
+*In[3]:*+
```

```
[source, ipython3]
```

```
----
```

```
y = data.iloc[:, -1]
```

```
print("\nThe first 5 values of Train output is\n", y.head())
```

```
----
```

```
+*Out[3]:*+
```

```
----
```

```
The first 5 values of Train output is
```

```
0    No
```

```
1    No
```

```
2    Yes
```

```
3    Yes
```

```
4    Yes
```

```
Name: PlayTennis, dtype: object
```

```
----
```

```
+*In[4]:*+
```

```
[source, ipython3]
```

```
----
```

```
# Convert then in numbers
```

```
le_outlook = LabelEncoder()
```

```
X.Outlook = le_outlook.fit_transform(X.Outlook)
```

```
le_Temperature = LabelEncoder()
```

```
X.Temperature = le_Temperature.fit_transform(X.Temperature)
```

```
le_Humidity = LabelEncoder()
```

```
X.Humidity = le_Humidity.fit_transform(X.Humidity)
```

```
le_Windy = LabelEncoder()
```

```
X.Windy = le_Windy.fit_transform(X.Windy)
```

```
print("\nNow the Train data is :\n",X.head())
```

```
----
```

```
+*Out[4]:*+
```

```
----
```

```
Now the Train data is :
```

```
Outlook Temperature Humidity Windy
```

```
0    2         1     0    0
```

```
1    2         1     0    1
```

```
2    0    1    0    0
3    1    2    0    0
4    1    0    1    0
```

```
----
```

```
+*In[5]:*+
```

```
[source, ipython3]
```

```
----
```

```
le_PlayTennis = LabelEncoder()
```

```
y = le_PlayTennis.fit_transform(y)
```

```
print("\nNow the Train output is\n",y)
```

```
----
```

```
+*Out[5]:*+
```

```
----
```

```
Now the Train output is
```

```
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
----
```

```
+*In[6]:*+
```

```
[source, ipython3]
```

```
----
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)
```

```
classifier = GaussianNB()
classifier.fit(X_train,y_train)
```

```
from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

----

```
+*Out[6]:*+
```

----

```
Accuracy is: 0.3333333333333333
```

----

```
+*In[ ]:*+
```

```
[source, ipython3]
```

----

----

**Output screenshots :-**



```
In [1]: # import necessary libraries
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# Load data from CSV
data = pd.read_csv(r"C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 4\Naive Bayesian classifier training dataset.csv")
print("The first 5 values of data is :\n",data.head())
```

The first 5 values of data is :

	Outlook	Temperature	Humidity	Windy	PlayTennis
0	Sunny	Hot	High	False	No
1	Sunny	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Rainy	Mild	High	False	Yes
4	Rainy	Cool	Normal	False	Yes

```
In [2]: # obtain Train data and Train output
X = data.iloc[:, :-1]
print("\nThe First 5 values of train data is\n",X.head())
```

The First 5 values of train data is

	Outlook	Temperature	Humidity	Windy
0	Sunny	Hot	High	False
1	Sunny	Hot	High	True
2	Overcast	Hot	High	False
3	Rainy	Mild	High	False
4	Rainy	Cool	Normal	False

```
In [3]: y = data.iloc[:, -1]
print("\nThe first 5 values of Train output is\n",y.head())
```

The first 5 values of Train output is

0	No
1	No
2	Yes
3	Yes
4	Yes

Name: PlayTennis, dtype: object



```
In [4]: # Convert then in numbers
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)

print("\nNow the Train data is :\n",X.head())
```

```
Now the Train data is :
   Outlook  Temperature  Humidity  Windy
0         2             1         0      0
1         2             1         0      1
2         0             1         0      0
3         1             2         0      0
4         1             0         1      0
```

```
In [5]: le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
```

```
Now the Train output is
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
In [6]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)

classifier = GaussianNB()
classifier.fit(X_train,y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

```
Accuracy is: 0.3333333333333333
```

In [ ]:

A1						Outlook
	A	B	C	D	E	F
1	Outlook	Temperat	Humidity	Windy	PlayTennis	
2	Sunny	Hot	High	FALSE	No	
3	Sunny	Hot	High	TRUE	No	
4	Overcast	Hot	High	FALSE	Yes	
5	Rainy	Mild	High	FALSE	Yes	
6	Rainy	Cool	Normal	FALSE	Yes	
7	Rainy	Cool	Normal	TRUE	No	
8	Overcast	Cool	Normal	TRUE	Yes	
9	Sunny	Mild	High	FALSE	No	
10	Sunny	Cool	Normal	FALSE	Yes	
11	Rainy	Mild	Normal	FALSE	Yes	
12	Sunny	Mild	Normal	TRUE	Yes	
13	Overcast	Mild	High	TRUE	Yes	
14	Overcast	Hot	Normal	FALSE	Yes	
15	Rainy	Mild	High	TRUE	No	
16						
17						
18						
19						

### Lab Program -4.b.-

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 (without packages).

#### Source code and output :-

```
+*In[1]:*+
```

```
[source, ipython3]
```

```
----
```

```
import math
```

```
import csv
```

```
import random
```

```
----
```

```
+*In[2]:*+
```

```
[source, ipython3]
```

```
----
```

# This make sures that the dataset is in an ordered format. If we have some arbitrary names in that column it difficult to deal with that.

```
def encode_class(dataset):
```

```
    classes=[]
```

```
    for i in range(len(dataset)):
```

```
        if dataset[i][-1] not in classes:
```

```
            classes.append(dataset[i][-1])
```



```
# Looping across the classes which we have derived above.This will make sure that we have definitive classes (numeric) and not arbitrary
```

```
for i in range(len(classes)):
```

```
    # Looping across all rows of dataset
```

```
    for j in range(len(dataset)):
```

```
        if dataset[j][-1] == classes[i]:
```

```
            dataset[j][-1]=i
```

```
return dataset
```

```
----
```

```
+*In[3]:*+
```

```
[source, ipython3]
```

```
----
```

```
# Splitting the data between training set and testing set. Normally its a general understanding the training:testing=7:3
```

```
def train_test_split(dataset,ratio):
```

```
    test_num=int(ratio*len(dataset))
```

```
    train=list(dataset)
```

```
    test=[]
```

```
    for i in range(test_num):
```

```
        rand=random.randrange(len(train))
```

```
        test.append(train.pop(rand))
```

```
    return train,test
```

```
----
```

```
+*In[4]:*+
```

```
[source, ipython3]
```

```
----
```

# Now depending on resultant value (last column values), we need to group the rows. It will be usefult for calculating mean and std\_dev

```
def groupUnderClass(train):
```

```
    dict={}
```

```
    for row in train:
```

```
        if row[-1] not in dict:
```

```
            dict[row[-1]]=[]
```

```
            dict[row[-1]].append(row)
```

```
    return dict
```

```
----
```

```
+*In[5]:*+
```

```
[source, ipython3]
```

```
----
```

# Standard formulae (just by-heart)

```
def mean(val):
```

```
    return sum(val)/float(len(val)) #Obvious
```

```
def stdDev(val):
```

```
avg=mean(val)
variance=sum([pow(x-avg,2) for x in val])/float(len(val)-1) # Especially this one
return math.sqrt(variance)
```

----

```
+*ln[6]:*+
```

```
[source, ipython3]
```

----

# We will calculate the mean and std dev with respect to each attribute. Important while calculating gaussian probability

```
def meanStdDev(instances):
```

```
    info=[(mean(x),stdDev(x)) for x in zip(*instances)] # Here we are taking complete column's values of all instances.
```

```
    del info[-1]
```

```
    return info
```

----

```
+*ln[7]:*+
```

```
[source, ipython3]
```

----

# As explained earlier why we need to group. We will be calculating the mean and std dev with respect each class.

```
def MeanAndStdDevForClass(train):
    info={}
    dictionary=groupUnderClass(train)
    # print(dictionary)
    for key,value in dictionary.items():
        # dictionary[key]=meanStdDev(value)
        info[key]=meanStdDev(value) #Here value stands for a complete group.
    return info
```

----

+\*ln[8]:\*+

[source, ipython3]

----

# Its a formula by heart (no choice)

```
def calculateGaussianProbablity(x,mean,std_dev):
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(std_dev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * std_dev)) * expo
```

----

+\*ln[9]:\*+

[source, ipython3]

----

# After calculating mean and std dev w.r.t training data now its time to check if the logic will work on testing data

```
def calculateClassProbablities(info,ele):
```

```
    probablities={}
```

```
    for key,summaries in info.items(): # Info contains the groupName (key) and list of
    (mean,std_dev) for each attribute of that group
```

```
        probablities[key]=1
```

```
        for i in range(len(summaries)): #Loop across all attributes
```

```
            mean,std_dev=summaries[i]
```

```
            x=ele[i] # Testing data's one instance's attribute value.
```

```
            probablities[key] *= calculateGaussianProbability(x, mean, std_dev)
```

```
    return probablities
```

```
----
```

```
+#ln[10]:*+
```

```
[source, ipython3]
```

```
----
```

```
def predict(info,ele):
```

```
    probablities=calculateClassProbablities(info,ele) # returns a dictionary of probablities for each
    group
```

```
    bestLabel,bestProb=None,-1
```

```
    # Consider group name whichever gives you the highest probablities for this instance of
    testing data
```

```
for key,prob in probablities.items():  
    if bestLabel==None or prob>bestProb:  
        bestProb=prob  
        bestLabel=key  
return bestLabel
```

----

```
+*In[11]:*+
```

```
[source, ipython3]
```

----

# Loop across testing data and store the predicted result from our model in the list.

```
def getPredictions(info,test):
```

```
    predictions=[]
```

```
    for ele in test:
```

```
        result=predict(info,ele) # This will give you the group to which it will belong.
```

```
        predictions.append(result)
```

```
    return predictions
```

----

```
+*In[12]:*+
```

```
[source, ipython3]
```

----

```
def check_accuracy(predictions,test):  
    count=0  
    for i in range(len(test)):  
        if predictions[i]==test[i][-1]:  
            count+=1  
    return count/float(len(test))*100
```

----

+\*ln[13]:\*+

[source, ipython3]

----

```
filename=r"C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 4\pima-indians-  
diabetes.csv"  
dataset=csv.reader(open(filename))  
dataset=list(dataset)  
dataset=encode_class(dataset)  
for i in range(len(dataset)):  
    dataset[i]=[float(x) for x in dataset[i]]  
  
ratio=0.3  
print(len(dataset))  
train,test=train_test_split(dataset,ratio)  
info=MeanAndStdDevForClass(train)  
  
predictions=getPredictions(info,test)
```

```
accuracy=check_accuracy(predictions,test)
```

```
accuracy
```

```
----
```

```
+*Out[13]:*+
```

```
----
```

```
768
```

```
75.21739130434783----
```

```
+*In[ ]:*+
```

```
[source, ipython3]
```

```
----
```

```
----
```

**Output screenshots :-**





```
In [1]: import math
import csv
import random
```

```
In [2]: # This make sures that the dataset is in an ordered format. If we have some arbitrary names in that column it difficult to deal with

def encode_class(dataset):
    classes=[]
    for i in range(len(dataset)):
        if dataset[i][-1] not in classes:
            classes.append(dataset[i][-1])

    # Looping across the classes which we have derived above.This will make sure that we have definitive classes (numeric) and not
    for i in range(len(classes)):
        # Looping across all rows of dataset
        for j in range(len(dataset)):
            if dataset[j][-1] == classes[i]:
                dataset[j][-1]=i
    return dataset
```

```
In [3]: # Splitting the data between training set and testing set. Normally its a general understanding the training:testing=7:3

def train_test_split(dataset,ratio):
    test_num=int(ratio*len(dataset))
    train=list(dataset)
    test=[]
    for i in range(test_num):
        rand=random.randrange(len(train))
        test.append(train.pop(rand))
    return train,test
```

```
In [4]: # Now depending on resultant value (last column values), we need to group the rows. It will be usefult for calculating mean and s

def groupUnderClass(train):
    dict={}
    for row in train:
        if row[-1] not in dict:
            dict[row[-1]]=[]
        dict[row[-1]].append(row)
    return dict
```



In [5]: *# Standard formulae (just by-heart)*

```
def mean(val):  
    return sum(val)/float(len(val)) #Obvious  
  
def stdDev(val):  
    avg=mean(val)  
    variance=sum([pow(x-avg,2) for x in val])/float(len(val)-1) # Especially this one  
    return math.sqrt(variance)
```

In [6]: *# We will calculate the mean and std dev with respect to each attribute. Important while calculating gaussian probability*

```
def meanStdDev(instances):  
    info=[(mean(x),stdDev(x)) for x in zip(*instances)] # Here we are taking complete column's values of all instances.  
    del info[-1]  
    return info
```

In [7]: *# As explained earlier why e need to group. We will be calculating the mean and std dev with respect each class.*

```
def MeanAndStdDevForClass(train):  
    info={}  
    dictionary=groupUnderClass(train)  
    # print(dictionary)  
    for key,value in dictionary.items():  
        # dictionary[key]=meanStdDev(value)  
        info[key]=meanStdDev(value) #Here value stands for a complete group.  
    return info
```

In [8]: *# Its a formula by heart (no choice)*

```
def calculateGaussianProbability(x,mean,std_dev):  
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(std_dev, 2))))  
    return (1 / (math.sqrt(2 * math.pi) * std_dev)) * expo
```



```
return (1 / (math.sqrt(2 * math.pi) * std_dev)) * expo
```

In [9]: *# After calculating mean and std dev w.r.t training data now its time to check if the Logic will work on testing data*

```
def calculateClassProbabilities(info,ele):  
    probabilities={}  
    for key,summaries in info.items(): # Info contains the groupName (key) and List of (mean,std_dev) for each attribute of that gr  
        probabilities[key]=1  
        for i in range(len(summaries)): #Loop across all attributes  
            mean,std_dev=summaries[i]  
            x=ele[i] # Testing data's one instance's attribute value.  
            probabilities[key] *= calculateGaussianProbability(x, mean, std_dev)  
    return probabilities
```

In [10]:

```
def predict(info,ele):  
    probabilities=calculateClassProbabilities(info,ele) # returns a dictionary of probabilities for each group  
    bestLabel,bestProb=None,-1  
    # Consider group name whichever gives you the highest probabilities for this instance of testing data  
    for key,prob in probabilities.items():  
        if bestLabel==None or prob>bestProb:  
            bestProb=prob  
            bestLabel=key  
    return bestLabel
```

In [11]: *# Loop across testing data and store the predicted result from our model in the list.*

```
def getPredictions(info,test):  
    predictions=[]  
    for ele in test:  
        result=predict(info,ele) # This will give you the group to which it will belong.  
        predictions.append(result)  
    return predictions
```

In [12]:

```
def check_accuracy(predictions,test):  
    count=0  
    for i in range(len(test)):  
        if predictions[i]==test[i]:  
            count+=1  
    return count/float(len(test))*100
```



```
def getPredictions(info,test):
    predictions=[]
    for ele in test:
        result=predict(info,ele) # This will give you the group to which it will belong.
        predictions.append(result)
    return predictions
```

In [12]:

```
def check_accuracy(predictions,test):
    count=0
    for i in range(len(test)):
        if predictions[i]!=test[i][-1]:
            count+=1
    return count/float(len(test))*100
```

In [13]:

```
filename=r"C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 4\pima-indians-diabetes.csv"
dataset=csv.reader(open(filename))
dataset=list(dataset)
dataset=encode_class(dataset)
for i in range(len(dataset)):
    dataset[i]=[float(x) for x in dataset[i]]

ratio=0.3
print(len(dataset))
train,test=train_test_split(dataset,ratio)
info=MeanAndStdDevForClass(train)

predictions=getPredictions(info,test)
accuracy=check_accuracy(predictions,test)
accuracy
```

768

Out[13]: 75.21739130434783

In [ ]:

	A	B	C	D	E	F	G	H	I	J
1	Pregnancies	Glucose	BloodPressure	BloodThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
2	6	148	72	35	0	33.6	0.627	50	1	
3	1	85	66	29	0	26.6	0.351	31	0	
4	8	183	64	0	0	23.3	0.672	32	1	
5	1	89	66	23	94	28.1	0.167	21	0	
6	0	137	40	35	168	43.1	2.288	33	1	
7	5	116	74	0	0	25.6	0.201	30	0	
8	3	78	50	32	88	31	0.248	26	1	
9	10	115	0	0	0	35.3	0.134	29	0	
10	2	197	70	45	543	30.5	0.158	53	1	
11	8	125	96	0	0	0	0.232	54	1	
12	4	110	92	0	0	37.6	0.191	30	0	
13	10	168	74	0	0	38	0.537	34	1	
14	10	139	80	0	0	27.1	1.441	57	0	
15	1	189	60	23	846	30.1	0.398	59	1	
16	5	166	72	19	175	25.8	0.587	51	1	
17	7	100	0	0	0	30	0.484	32	1	
18	0	118	84	47	230	45.8	0.551	31	1	
19	7	107	74	0	0	29.6	0.254	31	1	
20	1	103	30	38	83	43.3	0.183	33	0	
21	1	115	70	30	96	34.6	0.529	32	1	
22	3	126	88	41	235	39.3	0.704	27	0	
23	8	99	84	0	0	35.4	0.388	50	0	
24	7	196	90	0	0	39.8	0.451	41	1	
25	9	119	80	35	0	29	0.263	29	1	
26	11	143	94	33	146	36.6	0.254	51	1	
27	10	125	70	26	115	31.1	0.205	41	1	
28	7	147	76	0	0	39.4	0.257	43	1	

### Lab Program -5:-

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

#### Source code and output :-

```
)*In[1]:*+
```

```
[source, ipython3]
```

```
----
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import pandas as pd
```

```
----
```

```
)*In[11]:*+
```

```
[source, ipython3]
```

```
----
```

```
dataset = pd.read_csv(r"C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 5\Lr-Salary Dataset.csv")
```

```
X = dataset.iloc[:, :-1].values
```

```
y = dataset.iloc[:, 1].values
```

```
----
```

```
)*In[13]:*+
```

```
[source, ipython3]
```

```
----
```

```
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)
```

```
----
```

```
+*In[14]:*+
```

```
[source, ipython3]
```

```
----
```

```
# Fitting Simple Linear Regression to the Training set
```

```
from sklearn.linear_model import LinearRegression
```

```
regressor = LinearRegression()
```

```
regressor.fit(X_train, y_train)
```

```
----
```

```
+*Out[14]:*+
```

```
----LinearRegression()----
```

```
+*In[15]:*+
```

```
[source, ipython3]
```

```
----
```

```
# Predicting the Test set results
```

```
y_pred = regressor.predict(X_test)
```

----

+\*In[19]:\*+

[source, ipython3]

----

# 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()

----

+\*Out[19]:\*+

----

![png](output\_5\_0.png)

----

+\*In[17]:\*+

[source, ipython3]



----

# 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()

----

+\*Out[17]:\*+

----

![png](output\_6\_0.png)

----

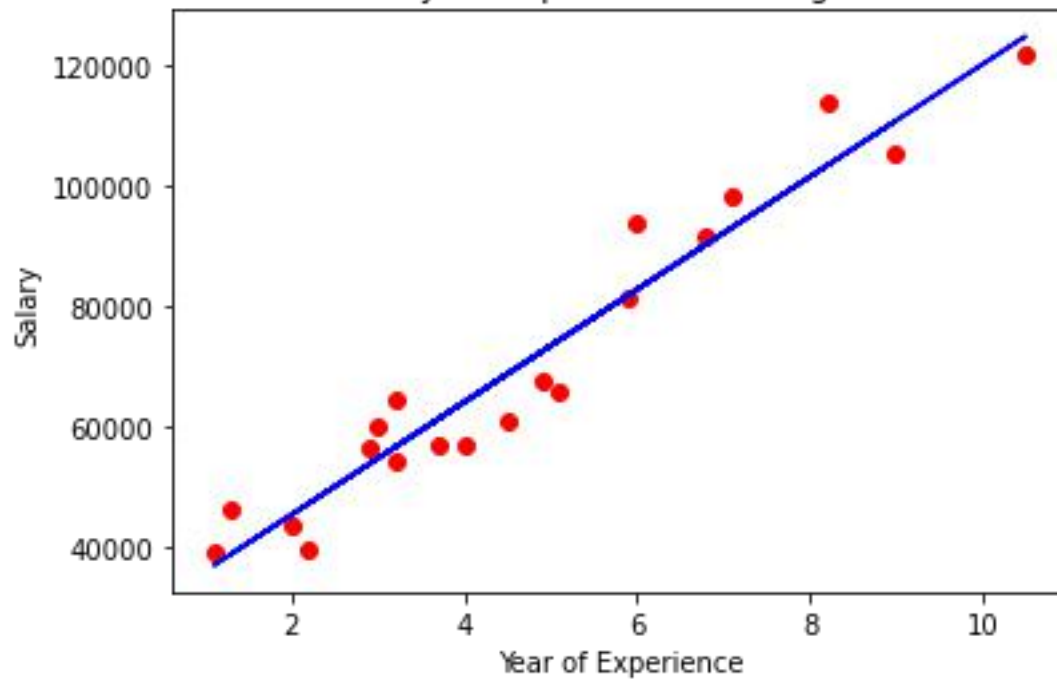
+\*In[ ]:\*+

[source, ipython3]

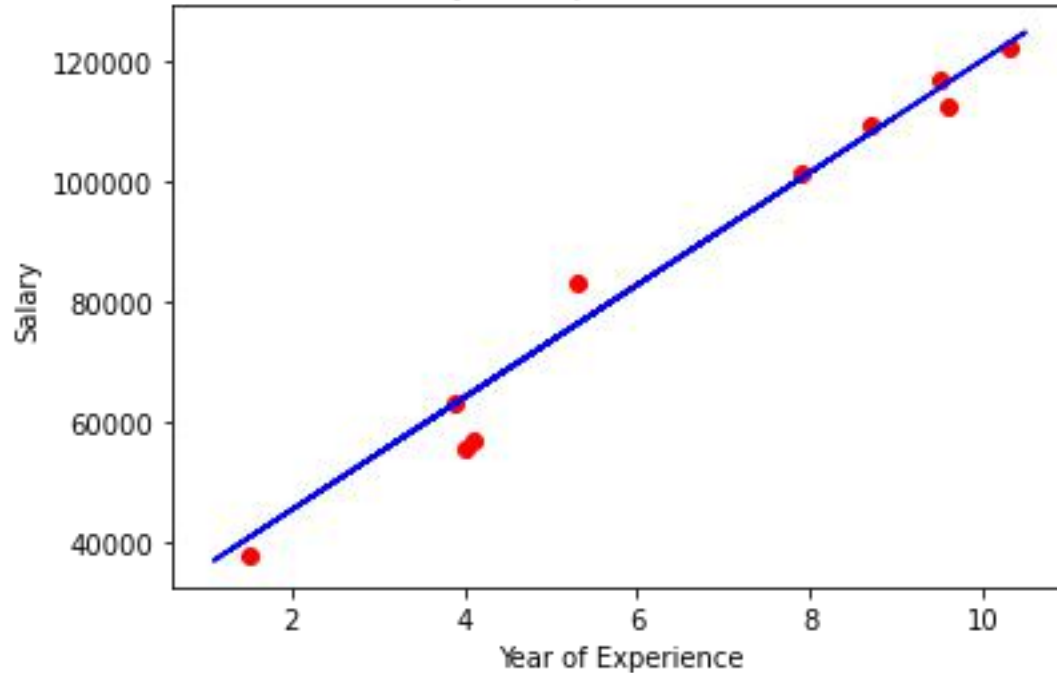
----

----

Salary VS Experience (Training set)



Salary VS Experience (Test set)



## Output screenshots :-

jupyter Lab-5 Linear regression 1BM19CS159 Last Checkpoint: 4 minutes ago (autosaved) Python 3 (ipykernel) [Logout](#)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

In [11]: dataset = pd.read_csv(r"C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 5\Lr-Salary Dataset.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

In [13]: 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)

In [14]: # Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

Out[14]: LinearRegression()

In [15]: # Predicting the Test set results
y_pred = regressor.predict(X_test)
```


jupyter Lab-5 Linear regression 1BM19CS159 Last Checkpoint: 4 minutes ago (autosaved) Python 3 (ipykernel) [Logout](#)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

```
y_pred = regressor.predict(X_test)

In [19]: # 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()
```

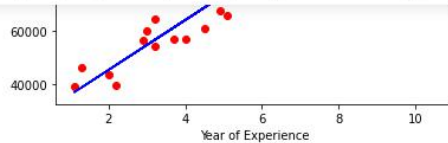




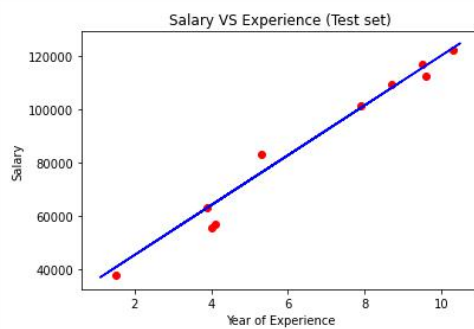
File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3 (ipykernel)



```
In [17]: # 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()
```



In [ ]:

format.

A1 X ✓ fx Year

	A	B	C
1	YearsExperience	Salary	
2	1.1	39343	
3	1.3	46205	
4	1.5	37731	
5	2	43525	
6	2.2	39891	
7	2.9	56642	
8	3	60150	
9	3.2	54445	
10	3.2	64445	
11	3.7	57189	
12	3.9	63218	
13	4	55794	
14	4	56957	
15	4.1	57081	
16	4.5	61111	
17	4.9	67938	
18	5.1	66029	
19	5.3	83088	
20	5.9	81363	
21	6	93940	
22	6.8	91738	
23	7.1	98273	
24	7.9	101302	
25	8.2	113812	

Lr-Salary Dataset +