

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT
on

MACHINE LEARNING **(20CS6PCMAL)**

Submitted by

PRITHVI J (1BM19CS122)

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 **PRITHVI J (1BM19CS122)**, 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

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

Index Sheet

Sl. No.	Experiment Title	Page No.
1.	<u>FIND-S</u> :- Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.	5
2.	<u>CANDIDATE ELIMINATION</u> :- 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.	11
3.	<u>ID3</u> :- 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.	20
4.a.	<u>NAÏVE BAYESIAN CLASSIFIER</u> :- 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.	<u>NAÏVE BAYESIAN CLASSIFIER (Without packages)</u> :- 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)	40
5.	<u>LINEAR REGRESSION</u> :- Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	55
6.	<u>BAYESIAN NETWORK</u> :-Write a program to construct a Bayesian network considering training data. Use this model to make predictions.	62
7.	<u>K-MEANS</u> :-Apply k-Means algorithm to cluster a set of data stored in a .CSV file.	76
8.	<u>EM</u> :-Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.	84
9.	<u>K NEAREST NEIGHBOUR</u> :-Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.	90
10.	<u>LOCALLY WEIGHTED REGRESSION</u> :-Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	101

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']
```

+*In[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 :

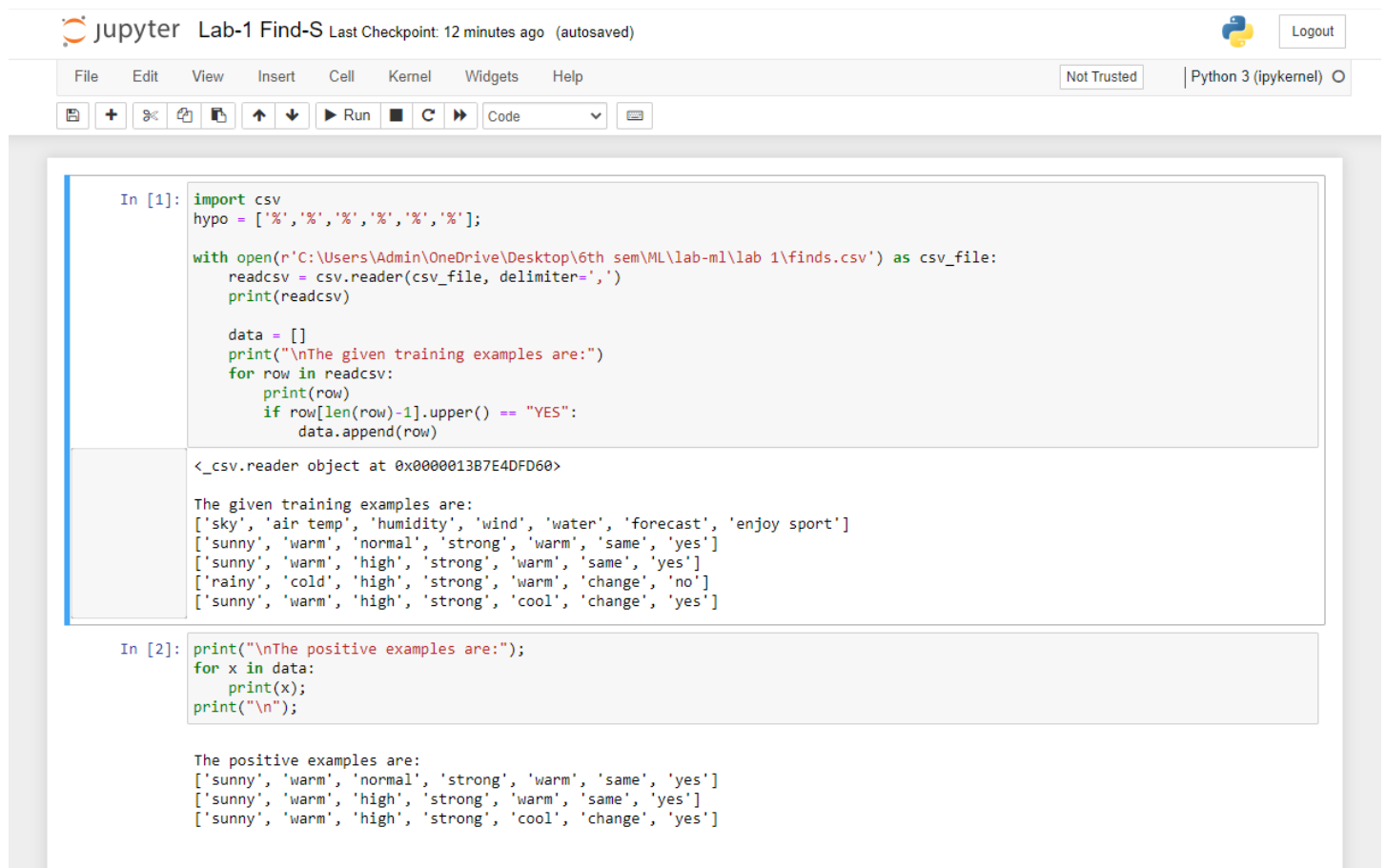
```
['sunny', 'warm', '?', 'strong', '?', '?']
```

```
----
```


+*In[]:*

[source, ipython3]

Output screenshots :-



The screenshot displays a JupyterLab environment with the following components:

- Header:** "jupyter Lab-1 Find-S Last Checkpoint: 12 minutes ago (autosaved)" and a "Logout" button.
- Menu Bar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help.
- Toolbar:** Includes icons for file operations, a "Run" button, and a "Code" dropdown menu.
- Code Cell 1 (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)
```

Output:

```
<_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']
```
- Code Cell 2 (In [2]):**

```
print("\nThe positive examples are:");
for x in data:
    print(x);
print("\n");
```

Output:

```
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']
```



[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 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)
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', '?', '?', '?', '?']]

+*In[]:*+

[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']
```



```
print(concepts)

['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)
```


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

Run 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 :",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 arbirary names in that column it difficult to deal wi

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 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
```

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
```



File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3 (ipykernel)

Code

```
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 group
        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
```



File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3 (ipykernel)

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][-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 []:

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()

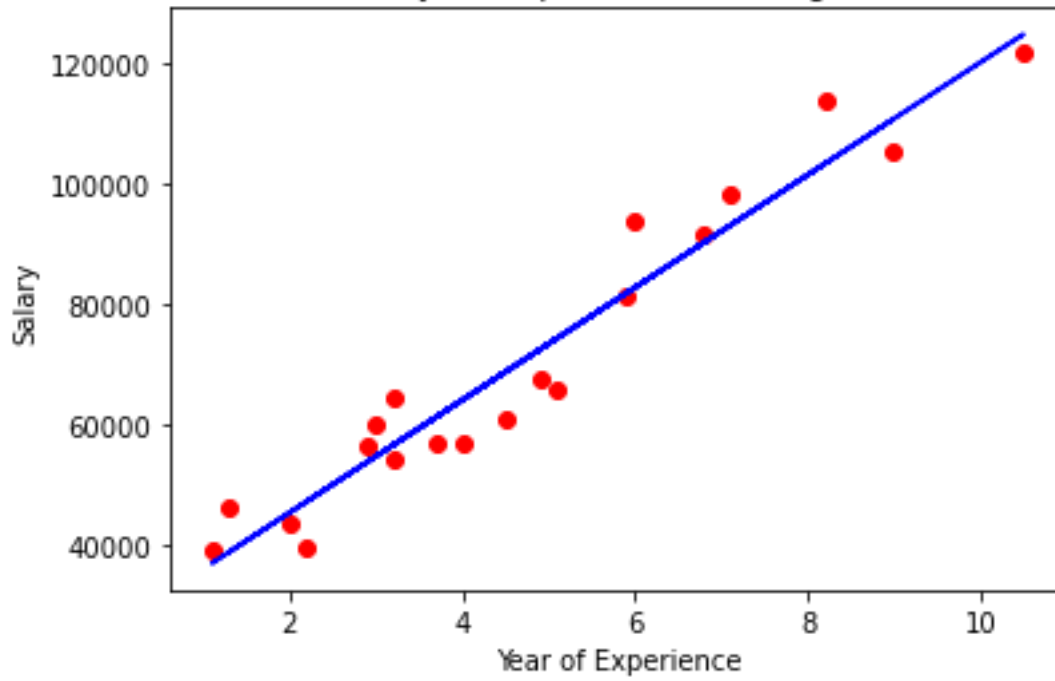
+*Out[17]:*+

![png](output_6_0.png)

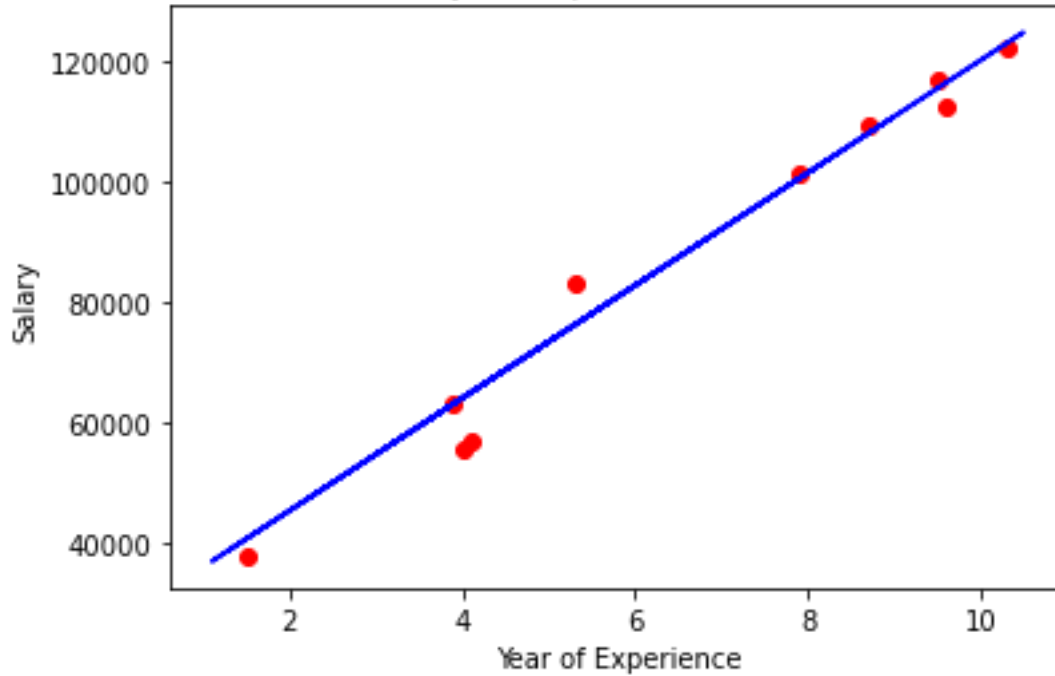
+*In []:*+

[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)  Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

```
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)  Logout

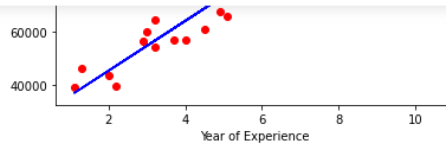
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

```
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()
```



Year of Experience	Salary
1	40000
2	45000
3	55000
4	60000
5	65000
6	80000
7	90000
8	100000
9	105000
10	115000



```
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	:	✕	✓	f_x 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		+		

Lab Program -6 :-

Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

Source code and output :-

```
.*In[1]:.*
```

```
[source, ipython3]
```

```
----
```

```
!pip install pgmpy
```

```
----
```

```
.*Out[1]:.*
```

```
----
```

Defaulting to user installation because normal site-packages is not writeable

Collecting pgmpy

Downloading pgmpy-0.1.18-py3-none-any.whl (1.9 MB)

Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.7.3)

Requirement already satisfied: pyparsing in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (3.0.4)

Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.4.2)

Collecting torch

Downloading torch-1.11.0-cp39-cp39-win_amd64.whl (157.9 MB)

Requirement already satisfied: scikit-learn in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.0.2)

Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.21.5)

Requirement already satisfied: tqdm in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (4.64.0)

Requirement already satisfied: networkx in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (2.7.1)

Requirement already satisfied: joblib in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.1.0)

Requirement already satisfied: statsmodels in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (0.13.2)

Requirement already satisfied: python-dateutil>=2.8.1 in c:\programdata\anaconda3\lib\site-packages (from pandas->pgmpy) (2.8.2)

Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pandas->pgmpy) (2021.3)

Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.8.1->pandas->pgmpy) (1.16.0)

Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn->pgmpy) (2.2.0)

Requirement already satisfied: patsy>=0.5.2 in c:\programdata\anaconda3\lib\site-packages (from statsmodels->pgmpy) (0.5.2)

Requirement already satisfied: packaging>=21.3 in c:\programdata\anaconda3\lib\site-packages (from statsmodels->pgmpy) (21.3)

Requirement already satisfied: typing-extensions in c:\programdata\anaconda3\lib\site-packages (from torch->pgmpy) (4.1.1)

Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (from tqdm->pgmpy) (0.4.4)

Installing collected packages: torch, pgmpy

Successfully installed pgmpy-0.1.18 torch-1.11.0

WARNING: The scripts convert-caffe2-to-onnx.exe, convert-onnx-to-caffe2.exe and torchrun.exe are installed in 'C:\Users\Admin\AppData\Roaming\Python\Python39\Scripts' which is not on PATH.

Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.

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

```
[source, ipython3]
```

```
import numpy as np
```

```
import pandas as pd
```

```
import csv
```

```
import pgmpy
```



```

from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

----

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

#read Cleveland Heart Disease data
heartDisease = pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 6\heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

----

+*In[7]:*+
[source, ipython3]
----

#display the data
print('Sample instances from the dataset are given below')
print(heartDisease.head())

----

+*Out[7]:*+
----

Sample instances from the dataset are given below

   Unnamed: 0  age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang \
0      NaN  63.0  1.0  1.0   145.0  233.0  1.0    2.0   150.0   0.0

```

1	NaN	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0
2	NaN	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0
3	NaN	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0
4	NaN	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0

... slope ca thal heartdisease Unnamed: 15 Unnamed: 16 Unnamed: 17 \

0 ...	3.0	0	6	0.0	NaN	NaN	NaN
1 ...	2.0	3	3	2.0	NaN	NaN	NaN
2 ...	2.0	2	7	1.0	NaN	NaN	NaN
3 ...	3.0	0	3	0.0	NaN	NaN	NaN
4 ...	1.0	0	3	0.0	NaN	NaN	NaN

Unnamed: 18 Unnamed: 19 Unnamed: 20

0	NaN	NaN	NaN
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
4	NaN	NaN	NaN

[5 rows x 21 columns]

+*In[8]:*+

[source, ipython3]

#display the Attributes names and datatypes

print('\n Attributes and datatypes')

print(heartDisease.dtypes)

+*Out[8]:*+

Attributes and datatypes

Unnamed: 0 float64

age float64

sex float64

cp float64

trestbps float64

chol float64

fbs float64

restecg float64

thalach float64

exang float64

oldpeak float64

slope float64

ca object

thal object

heartdisease float64

Unnamed: 15 float64

Unnamed: 16 float64

Unnamed: 17 float64

Unnamed: 18 float64

Unnamed: 19 float64

Unnamed: 20 float64

dtype: object

+*In[9]:*+

[source, ipython3]

#Creat Model-Bayesian Network

model =

BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])

+*Out[9]:*+

C:\Users\Admin\AppData\Roaming\Python\Python39\site-packages\pgmpy\models\BayesianModel.py:8:
FutureWarning: BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class,
BayesianModel will be removed in future.

warnings.warn(

+*In[10]:*+

[source, ipython3]

#Learning CPDs using Maximum Likelihood Estimators

print('\n Learning CPD using Maximum likelihood estimators')

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

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

```
----
```

```
Learning CPD using Maximum likelihood estimators
```

```
----
```

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

```
[source, ipython3]
```

```
----
```

```
#Inferencing with Bayesian Network
```

```
print('\n Inferencing with Bayesian Network:')
```

```
HeartDiseasetest_infer = VariableElimination(model)
```

```
----
```

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

```
----
```

```
Inferencing with Bayesian Network:
```

```
----
```

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

```
[source, ipython3]
```

```
----
```

```
#computing the Probability of HeartDisease given restecg
```

```
print('\n 1.Probability of HeartDisease given evidence= restecg :1')
```

```
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
```

```
print(q1)
```

```
----
```

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

```
----
```

1.Probability of HeartDisease given evidence= restecg :1

0%| | 0/4 [00:00<?, ?it/s] 0%| | 0/4 [00:00<?, ?it/s]

```
+-----+-----+
```

```
| heartdisease | phi(heartdisease) |
```

```
+=====+
```

```
| heartdisease(0.0) |      0.2000 |
```

```
+-----+-----+
```

```
| heartdisease(1.0) |      0.2000 |
```

```
+-----+-----+
```

```
| heartdisease(2.0) |      0.2000 |
```

```
+-----+-----+
```

```
| heartdisease(3.0) |      0.2000 |
```

```
+-----+-----+
```

```
| heartdisease(4.0) |      0.2000 |
```

```
+-----+-----+
```

```
----
```

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

```
[source, ipython3]
```

```
----
```

```
#computing the Probability of HeartDisease given cp
print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

```
----
```

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

```
----
```


```
2.Probability of HeartDisease given evidence= cp:2
0%|      | 0/3 [00:00<?, ?it/s] 0%|      | 0/3 [00:00<?, ?it/s]
```

```
+-----+-----+
| heartdisease | phi(heartdisease) |
+=====+=====+
| heartdisease(0.0) |      0.2000 |
+-----+-----+
| heartdisease(1.0) |      0.2000 |
+-----+-----+
| heartdisease(2.0) |      0.2000 |
+-----+-----+
| heartdisease(3.0) |      0.2000 |
+-----+-----+
| heartdisease(4.0) |      0.2000 |
+-----+-----+
```

```
----
```


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

[source, ipython3]

 jupyter









Lab-6 Bayesian Network (heart dataset) 1BM19CS159

Last Checkpoint: 6 minutes ago (autosaved)

 Logout

FileEditViewInsertCellKernelWidgetsHelp

TrustedPython 3 (ipykernel)



```
In [1]: !pip install pgmpy

Defaulting to user installation because normal site-packages is not writeable
Collecting pgmpy
  Downloading pgmpy-0.1.18-py3-none-any.whl (1.9 MB)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.7.3)
Requirement already satisfied: pyparsing in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (3.0.4)
Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.4.2)
Collecting torch
  Downloading torch-1.11.0-cp39-cp39-win_amd64.whl (157.9 MB)
Requirement already satisfied: scikit-learn in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.0.2)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.21.5)
Requirement already satisfied: tqdm in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (4.64.0)
Requirement already satisfied: networkx in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (2.7.1)
Requirement already satisfied: joblib in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.1.0)
Requirement already satisfied: statsmodels in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (0.13.2)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\programdata\anaconda3\lib\site-packages (from pandas->pgmpy) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pandas->pgmpy) (2021.3)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.8.1->pandas->pgmpy) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn->pgmpy) (2.2.0)
Requirement already satisfied: patsy>=0.5.2 in c:\programdata\anaconda3\lib\site-packages (from statsmodels->pgmpy) (0.5.2)
Requirement already satisfied: packaging>=21.3 in c:\programdata\anaconda3\lib\site-packages (from statsmodels->pgmpy) (21.3)
Requirement already satisfied: typing-extensions in c:\programdata\anaconda3\lib\site-packages (from torch->pgmpy) (4.1.1)
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (from tqdm->pgmpy) (0.4.4)
Installing collected packages: torch, pgmpy
Successfully installed pgmpy-0.1.18 torch-1.11.0

WARNING: The scripts convert-caffe2-to-onnx.exe, convert-onnx-to-caffe2.exe and torchrun.exe are installed in 'C:\Users\Admin\AppData\Roaming\Python\Python39\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
```

```
In [1]: import numpy as np
import pandas as pd
import csv
import pgmpy
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
```




```
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
```

```
In [6]: #read Cleveland Heart Disease data
heartDisease = pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 6\heart.csv')
heartDisease = heartDisease.replace('?', np.nan)
```

```
In [7]: #display the data
print('Sample instances from the dataset are given below')
print(heartDisease.head())
```

Sample instances from the dataset are given below

	Unnamed: 0	age	sex	cp	trestbps	chol	fb	restecg	thalach	exang	\
0	NaN	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	
1	NaN	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	
2	NaN	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	
3	NaN	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	
4	NaN	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	

	...	slope	ca	thal	heartdisease	Unnamed: 15	Unnamed: 16	Unnamed: 17	\
0	...	3.0	0	6	0.0	NaN	NaN	NaN	
1	...	2.0	3	3	2.0	NaN	NaN	NaN	
2	...	2.0	2	7	1.0	NaN	NaN	NaN	
3	...	3.0	0	3	0.0	NaN	NaN	NaN	
4	...	1.0	0	3	0.0	NaN	NaN	NaN	

	Unnamed: 18	Unnamed: 19	Unnamed: 20
0	NaN	NaN	NaN
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
4	NaN	NaN	NaN

[5 rows x 21 columns]

```
In [8]: #display the Attributes names and datatypes
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
Attributes and datatypes
Unnamed: 0      float64
age            float64
sex            float64
cp            float64
trestbps       float64
chol           float64
fbs            float64
restecg        float64
thalach        float64
exang          float64
oldpeak        float64
slope          float64
ca             object
thal           object
heartdisease    float64
Unnamed: 15     float64
Unnamed: 16     float64
Unnamed: 17     float64
Unnamed: 18     float64
Unnamed: 19     float64
Unnamed: 20     float64
dtype: object
```

```
In [9]: #Creat Model-Bayesian Network
model = BayesianModel([('age', 'heartdisease'), ('sex', 'heartdisease'), ('exang', 'heartdisease'), ('cp', 'heartdisease'), ('heartdisease', 'restecg')])

C:\Users\Admin\AppData\Roaming\Python\Python39\site-packages\pgmpy\models\BayesianModel.py:8: FutureWarning: BayesianModel has
been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.
warnings.warn()
```

```
In [10]: #Learning CPDs using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)
```

Learning CPD using Maximum likelihood estimators

```
In [11]: #Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
```

Inferencing with Bayesian Network:

```
In [12]: #computing the Probability of HeartDisease given restecg
print('\n 1.Probability of HeartDisease given evidence= restecg :1')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
```

1.Probability of HeartDisease given evidence= restecg :1

Finding Elimination Order: : 100% 4/4 [00:00<00:00, 142.85it/s]

Eliminating: exang: 100% 4/4 [00:00<00:00, 129.02it/s]

heartdisease	phi(heartdisease)
heartdisease(0.0)	0.2000
heartdisease(1.0)	0.2000
heartdisease(2.0)	0.2000
heartdisease(3.0)	0.2000
heartdisease(4.0)	0.2000

```
In [13]: #computing the Probability of HeartDisease given cp
```

```
In [14]: #computing the Probability of HeartDisease given cp
print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

2.Probability of HeartDisease given evidence= cp:2

Finding Elimination Order: : 0% 0/3 [00:00<?, ?it/s]

Eliminating: exang: 100% 3/3 [00:00<00:00, 157.74it/s]

```
+-----+-----+
| heartdisease | phi(heartdisease) |
+-----+-----+
| heartdisease(0.0) | 0.2000 |
+-----+-----+
| heartdisease(1.0) | 0.2000 |
+-----+-----+
| heartdisease(2.0) | 0.2000 |
+-----+-----+
| heartdisease(3.0) | 0.2000 |
+-----+-----+
| heartdisease(4.0) | 0.2000 |
+-----+-----+
```

In []:

A	B	C	D	E	F	G	H	I	J	K	L	M
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
	63	1	1	145	233	1	2	150	0	2.3	3	
	67	1	4	160	286	0	2	108	1	1.5	2	
	67	1	4	120	229	0	2	129	1	2.6	2	
	37	1	3	130	250	0	0	187	0	3.5	3	
	41	0	2	130	204	0	2	172	0	1.4	1	
	56	1	2	120	236	0	0	178	0	0.8	1	
	62	0	4	140	268	0	2	160	0	3.6	3	
	57	0	4	120	354	0	0	163	1	0.6	1	
	63	1	4	130	254	0	2	147	0	1.4	2	
	53	1	4	140	203	1	2	155	1	3.1	3	
	57	1	4	140	192	0	0	148	0	0.4	2	
	56	0	2	140	294	0	2	153	0	1.3	2	
	56	1	3	130	256	1	2	142	1	0.6	2	
	44	1	2	120	263	0	0	173	0	0	1	
	52	1	3	172	199	1	0	162	0	0.5	1	
	57	1	3	150	168	0	0	174	0	1.6	1	
	48	1	2	110	229	0	0	168	0	1	3	
	54	1	4	140	239	0	0	160	0	1.2	1	
	48	0	3	130	275	0	0	139	0	0.2	1	
	49	1	2	130	266	0	0	171	0	0.6	1	
	64	1	1	110	211	0	2	144	1	1.8	2	
	58	0	1	150	283	1	2	162	0	1	1	
	58	1	2	120	284	0	2	160	0	1.8	2	

Lab Program -7 :-

Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

Source code and output :-

```
#!/usr/bin/env python
```

```
# coding: utf-8
```

```
# In[18]:
```

```
import pandas as pd
```

```
import matplotlib
```

```
import numpy as np
```

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

```
from sklearn.cluster import KMeans
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
from matplotlib import pyplot as plt
```

```
get_ipython().run_line_magic('matplotlib', 'inline')
```

```
# In[19]:
```

```
df = pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 7\income.csv')
```

```
df.head(10)
```

```
# In[20]:
```

```
scaler = MinMaxScaler()
scaler.fit(df[['Age']])
df[['Age']] = scaler.transform(df[['Age']])

scaler.fit(df[['Income($)']])
df[['Income($)']] = scaler.transform(df[['Income($)']])
df.head(10)
```

```
# In[21]:
```

```
plt.scatter(df['Age'], df['Income($)'])
```

```
# In[22]:
```

```
k_range = range(1, 11)
sse = []
for k in k_range:
    kmc = KMeans(n_clusters=k)
    kmc.fit(df[['Age', 'Income($)']])
    sse.append(kmc.inertia_)
sse
```

```
# In[23]:
```

```
plt.xlabel = 'Number of Clusters'  
plt.ylabel = 'Sum of Squared Errors'  
plt.plot(k_range, sse)
```

Therefore, the elbow point is 3

```
# In[24]:
```

```
km = KMeans(n_clusters=3)  
km
```

```
# In[25]:
```

```
y_predict = km.fit_predict(df[['Age', 'Income($)']])  
y_predict
```

```
# In[26]:
```

```
df['cluster'] = y_predict  
df.head()
```

```
# In[27]:
```

```
df0 = df[df.cluster == 0]
```

```
df0
```

```
# In[28]:
```

```
df1 = df[df.cluster == 1]
```

```
df1
```

```
# In[29]:
```

```
df2 = df[df.cluster == 2]
```

```
df2
```

```
# In[30]:
```

```
km.cluster_centers_
```

```
# In[34]:
```

```

p1 = plt.scatter(df0['Age'], df0['Income($)', marker='+', color='red')
p2 = plt.scatter(df1['Age'], df1['Income($)', marker='*', color='blue')
p3 = plt.scatter(df2['Age'], df2['Income($)', marker='^', color='green')
c = plt.scatter(km.cluster_centers[:,0], km.cluster_centers[:,1], color='black')
plt.legend((p1, p2, p3, c),
           ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))

```

	A	B	C	D
1	Name	Age	Income(\$)	
2	Rob	27	70000	
3	Michael	29	90000	
4	Mohan	29	61000	
5	Ismail	28	60000	
6	Kory	42	150000	
7	Gautam	39	155000	
8	David	41	160000	
9	Andrea	38	162000	
10	Brad	36	156000	
11	Angelina	35	130000	
12	Donald	37	137000	
13	Tom	26	45000	
14	Arnold	27	48000	
15	Jared	28	51000	
16	Stark	29	49500	
17	Ranbir	32	53000	
18	Dipika	40	65000	
19	Priyanka	41	63000	
20	Nick	43	64000	
21	Alia	39	80000	
22	Sid	41	82000	
23	Abdul	39	58000	

In[]:


```
In [18]: import pandas as pd
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
```

```
In [19]: df = pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 7\income.csv')
df.head(10)
```

Out[19]:

	Name	Age	Income(\$)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000
5	Gautam	39	155000
6	David	41	160000
7	Andrea	38	162000
8	Brad	36	156000
9	Angelina	35	130000

```
In [20]: scaler = MinMaxScaler()
scaler.fit(df[['Age']])
df[['Age']] = scaler.transform(df[['Age']])

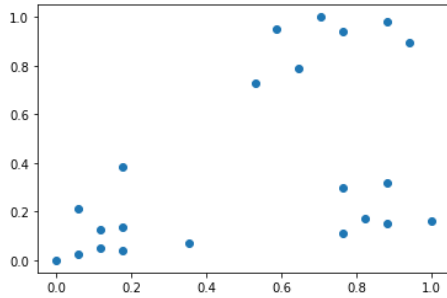
scaler.fit(df[['Income($)']])
df[['Income($)']] = scaler.transform(df[['Income($)']])
df.head(10)
```

Out[20]:

	Name	Age	Income(\$)
0	Rob	0.058824	0.213675
1	Michael	0.176471	0.384615

```
In [21]: plt.scatter(df['Age'], df['Income($)'])
```

```
Out[21]: <matplotlib.collections.PathCollection at 0x298b0f99760>
```



```
In [22]: k_range = range(1, 11)
sse = []
for k in k_range:
    kmc = KMeans(n_clusters=k)
    kmc.fit(df[['Age', 'Income($)']])
    sse.append(kmc.inertia_)
sse
```

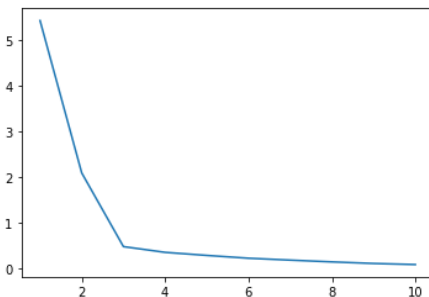
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMPI_NUM_THREADS=1.
warnings.warn(

```
Out[22]: [5.434011511988178,
2.0911363886990775,
0.4750783498553096,
0.34910470944195654,
0.2818479744366238,
0.22020960864009398,
0.17840674931327938,
0.1397684499538816,
0.10497488680620909,
0.08139933135681814]
```

```
In [23]: plt.xlabel = 'Number of Clusters'
```

```
In [23]: plt.xlabel = 'Number of Clusters'
plt.ylabel = 'Sum of Squared Errors'
plt.plot(k_range, sse)
```

```
Out[23]: <matplotlib.lines.Line2D at 0x298b1fc7eb0>
```



Therefore, the elbow point is 3

```
In [24]: km = KMeans(n_clusters=3)
km
```

```
Out[24]: KMeans(n_clusters=3)
```

```
In [25]: y_predict = km.fit_predict(df[['Age', 'Income($)']])
y_predict
```

```
Out[25]: array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2])
```

```
In [26]: df['cluster'] = y_predict
df.head()
```

```
In [28]: df1 = df[df.cluster == 1]
df1
```

```
Out[28]:
```

	Name	Age	Income(\$)	cluster
4	Kory	0.941176	0.897436	1
5	Gautam	0.764706	0.940171	1
6	David	0.882353	0.982906	1
7	Andrea	0.705882	1.000000	1
8	Brad	0.588235	0.948718	1
9	Angelina	0.529412	0.726496	1
10	Donald	0.647059	0.786325	1

```
In [29]: df2 = df[df.cluster == 2]
df2
```

```
Out[29]:
```

	Name	Age	Income(\$)	cluster
16	Dipika	0.823529	0.170940	2
17	Priyanka	0.882353	0.153846	2
18	Nick	1.000000	0.162393	2
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2
21	Abdul	0.764706	0.111111	2

```
In [30]: km.cluster_centers_
```

```
Out[30]: array([[0.1372549 , 0.11633428],
 [0.72268908, 0.8974359 ],
 [0.85294118, 0.2022792 ]])
```

```
In [34]: p1 = plt.scatter(df0['Age'], df0['Income($)'], marker='+', color='red')
p2 = plt.scatter(df1['Age'], df1['Income($)'], marker='*', color='blue')
p3 = plt.scatter(df2['Age'], df2['Income($)'], marker='^', color='green')
c = plt.scatter(km.cluster_centers_[0], km.cluster_centers_[1], color='black')
```

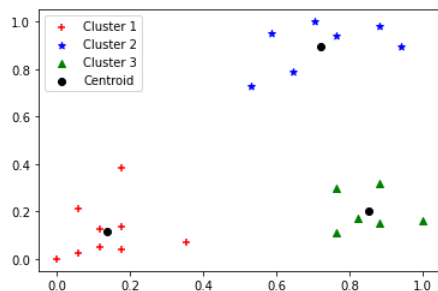
```
21 Abdul 0.764706 0.111111 2
```

```
In [30]: km.cluster_centers_
```

```
Out[30]: array([[0.1372549 , 0.11633428],
 [0.72268908, 0.8974359 ],
 [0.85294118, 0.2022792 ]])
```

```
In [34]: p1 = plt.scatter(df0['Age'], df0['Income($)'], marker='+', color='red')
p2 = plt.scatter(df1['Age'], df1['Income($)'], marker='*', color='blue')
p3 = plt.scatter(df2['Age'], df2['Income($)'], marker='^', color='green')
c = plt.scatter(km.cluster_centers_[0], km.cluster_centers_[1], color='black')
plt.legend((p1, p2, p3, c),
           ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))
```

```
Out[34]: <matplotlib.legend.Legend at 0x298b47d5970>
```



```
In [ ]:
```

Lab Program -8 :-

Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

Source code and output :-

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

```
[source, ipython3]
```

```
----
```

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

```
from sklearn import datasets
```

```
from sklearn.cluster import KMeans
```

```
import sklearn.metrics as sm
```

```
import pandas as pd
```

```
import numpy as np
```

```
iris = datasets.load_iris()
```

```
X = pd.DataFrame(iris.data)
```

```
X.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']
```

```
y = pd.DataFrame(iris.target)
```

```
y.columns = ['Targets']
```

```
model = KMeans(n_clusters=3)
```

```
model.fit(X)
```

```
plt.figure(figsize=(14,7))
```

```
colormap = np.array(['red', 'lime', 'black'])
```

```
# Plot the Original Classifications
```

```
plt.subplot(1, 2, 1)
```

```
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
```

```
plt.title('Real Classification')
```

```
plt.xlabel('Petal Length')
```

```
plt.ylabel('Petal Width')
```

```
# Plot the Models Classifications
```

```
plt.subplot(1, 2, 2)
```

```
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
```

```
plt.title('K Mean Classification')
```

```
plt.xlabel('Petal Length')
```

```
plt.ylabel('Petal Width')
```

```
print('The accuracy score of K-Mean: ', sm.accuracy_score(y, model.labels_))
```

```
print('The Confusion matrix of K-Mean: ', sm.confusion_matrix(y, model.labels_))
```

```
from sklearn import preprocessing
```

```
scaler = preprocessing.StandardScaler()
```

```
scaler.fit(X)
```

```
xsa = scaler.transform(X)
```

```
xs = pd.DataFrame(xsa, columns = X.columns)
```

```
#xs.sample(5)
```

```
from sklearn.mixture import GaussianMixture
```

```
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

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

The accuracy score of K-Mean: 0.24

The Confusion matrix of K-Mean: [[0 50 0]

[48 0 2]

[14 0 36]]

The accuracy score of EM: 0.3533333333333333

The Confusion matrix of EM: [[5 0 45]

[2 48 0]

[0 50 0]]

![png](output_0_1.png)

+*In[]:*+

[source, ipython3]



File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3 (ipykernel) C

```
In [1]: import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X)

plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ', sm.accuracy_score(y, model.labels_))
print('The Confusion matrix of K-Mean: ', sm.confusion_matrix(y, model.labels_))

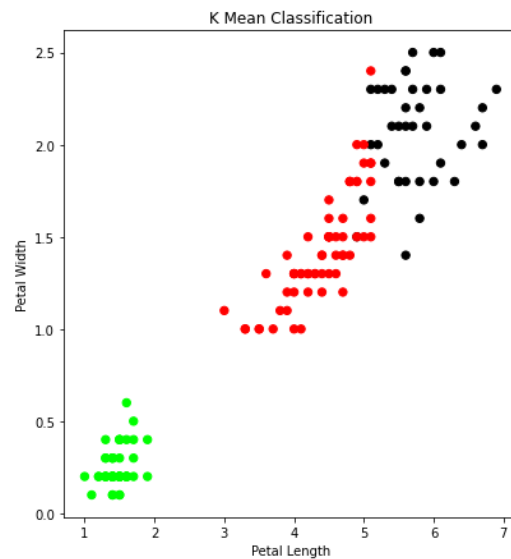
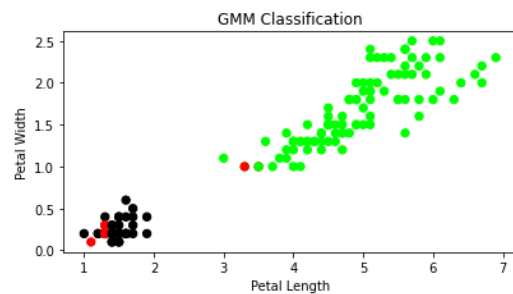
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
```



```
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

```
The accuracy score of K-Mean: 0.24
The Confusion matrix of K-Mean: [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
The accuracy score of EM: 0.35333333333333333
The Confusion matrix of EM: [[ 5  0 45]
 [ 2 48  0]
 [ 0 50  0]]
```



In []:

Lab Program -9:-

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

Source code and output :-

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

```
[source, ipython3]
```

```
----
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
from sklearn import datasets
```

```
iris=datasets.load_iris()
```

```
x = iris.data
```

```
y = iris.target
```

```
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
```

```
print(x)
```

```
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
```

```
print(y)
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
```

```
#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
```

```
#To make predictions on our test data
y_pred=classifier.predict(x_test)
```

```
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
----
```

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

```
----
```

```
sepal-length sepal-width petal-length petal-width
```

```
[[5.1 3.5 1.4 0.2]
```

```
[4.9 3. 1.4 0.2]
```

```
[4.7 3.2 1.3 0.2]
```

```
[4.6 3.1 1.5 0.2]
```

```
[5. 3.6 1.4 0.2]
```

```
[5.4 3.9 1.7 0.4]
```

```
[4.6 3.4 1.4 0.3]
```

```
[5. 3.4 1.5 0.2]
```

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

[5.4 3.7 1.5 0.2]

[4.8 3.4 1.6 0.2]

[4.8 3. 1.4 0.1]

[4.3 3. 1.1 0.1]

[5.8 4. 1.2 0.2]

[5.7 4.4 1.5 0.4]

[5.4 3.9 1.3 0.4]

[5.1 3.5 1.4 0.3]

[5.7 3.8 1.7 0.3]

[5.1 3.8 1.5 0.3]

[5.4 3.4 1.7 0.2]

[5.1 3.7 1.5 0.4]

[4.6 3.6 1. 0.2]

[5.1 3.3 1.7 0.5]

[4.8 3.4 1.9 0.2]

[5. 3. 1.6 0.2]

[5. 3.4 1.6 0.4]

[5.2 3.5 1.5 0.2]

[5.2 3.4 1.4 0.2]

[4.7 3.2 1.6 0.2]

[4.8 3.1 1.6 0.2]

[5.4 3.4 1.5 0.4]

[5.2 4.1 1.5 0.1]

[5.5 4.2 1.4 0.2]

[4.9 3.1 1.5 0.2]

[5. 3.2 1.2 0.2]

[5.5 3.5 1.3 0.2]

[4.9 3.6 1.4 0.1]

[4.4 3. 1.3 0.2]

[5.1 3.4 1.5 0.2]

[5. 3.5 1.3 0.3]

[4.5 2.3 1.3 0.3]

[4.4 3.2 1.3 0.2]

[5. 3.5 1.6 0.6]

[5.1 3.8 1.9 0.4]

[4.8 3. 1.4 0.3]

[5.1 3.8 1.6 0.2]

[4.6 3.2 1.4 0.2]

[5.3 3.7 1.5 0.2]

[5. 3.3 1.4 0.2]

[7. 3.2 4.7 1.4]

[6.4 3.2 4.5 1.5]

[6.9 3.1 4.9 1.5]

[5.5 2.3 4. 1.3]

[6.5 2.8 4.6 1.5]

[5.7 2.8 4.5 1.3]

[6.3 3.3 4.7 1.6]

[4.9 2.4 3.3 1.]

[6.6 2.9 4.6 1.3]

[5.2 2.7 3.9 1.4]

[5. 2. 3.5 1.]

[5.9 3. 4.2 1.5]

[6. 2.2 4. 1.]

[6.1 2.9 4.7 1.4]

[5.6 2.9 3.6 1.3]

[6.7 3.1 4.4 1.4]

[5.6 3. 4.5 1.5]

[5.8 2.7 4.1 1.]

[6.2 2.2 4.5 1.5]

[5.6 2.5 3.9 1.1]

[5.9 3.2 4.8 1.8]

[6.1 2.8 4. 1.3]

[6.3 2.5 4.9 1.5]

[6.1 2.8 4.7 1.2]

[6.4 2.9 4.3 1.3]

[6.6 3. 4.4 1.4]

[6.8 2.8 4.8 1.4]

[6.7 3. 5. 1.7]

[6. 2.9 4.5 1.5]

[5.7 2.6 3.5 1.]

[5.5 2.4 3.8 1.1]

[5.5 2.4 3.7 1.]

[5.8 2.7 3.9 1.2]

[6. 2.7 5.1 1.6]

[5.4 3. 4.5 1.5]

[6. 3.4 4.5 1.6]

[6.7 3.1 4.7 1.5]

[6.3 2.3 4.4 1.3]

[5.6 3. 4.1 1.3]

[5.5 2.5 4. 1.3]

[5.5 2.6 4.4 1.2]

[6.1 3. 4.6 1.4]

[5.8 2.6 4. 1.2]

[5. 2.3 3.3 1.]

[5.6 2.7 4.2 1.3]

[5.7 3. 4.2 1.2]

[5.7 2.9 4.2 1.3]

[6.2 2.9 4.3 1.3]

[5.1 2.5 3. 1.1]

[5.7 2.8 4.1 1.3]

[6.3 3.3 6. 2.5]

[5.8 2.7 5.1 1.9]

[7.1 3. 5.9 2.1]

[6.3 2.9 5.6 1.8]

[6.5 3. 5.8 2.2]

[7.6 3. 6.6 2.1]

[4.9 2.5 4.5 1.7]

[7.3 2.9 6.3 1.8]

[6.7 2.5 5.8 1.8]

[7.2 3.6 6.1 2.5]

[6.5 3.2 5.1 2.]

[6.4 2.7 5.3 1.9]

[6.8 3. 5.5 2.1]

[5.7 2.5 5. 2.]

[5.8 2.8 5.1 2.4]

[6.4 3.2 5.3 2.3]

[6.5 3. 5.5 1.8]

[7.7 3.8 6.7 2.2]

[7.7 2.6 6.9 2.3]

[6. 2.2 5. 1.5]

[6.9 3.2 5.7 2.3]

[5.6 2.8 4.9 2.]

[7.7 2.8 6.7 2.]

[6.3 2.7 4.9 1.8]

[6.7 3.3 5.7 2.1]

[7.2 3.2 6. 1.8]

[6.2 2.8 4.8 1.8]

[6.1 3. 4.9 1.8]

[6.4 2.8 5.6 2.1]

[7.2 3. 5.8 1.6]

[7.4 2.8 6.1 1.9]

[7.9 3.8 6.4 2.]

[6.4 2.8 5.6 2.2]

[6.3 2.8 5.1 1.5]

[6.1 2.6 5.6 1.4]

[7.7 3. 6.1 2.3]

[6.3 3.4 5.6 2.4]

[6.4 3.1 5.5 1.8]

[6. 3. 4.8 1.8]

[6.9 3.1 5.4 2.1]

[6.7 3.1 5.6 2.4]

[6.9 3.1 5.1 2.3]

[5.8 2.7 5.1 1.9]

[6.8 3.2 5.9 2.3]

[6.7 3.3 5.7 2.5]

[6.7 3. 5.2 2.3]

[6.3 2.5 5. 1.9]

[6.5 3. 5.2 2.]

[6.2 3.4 5.4 2.3]

[5.9 3. 5.1 1.8]]

class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica

```
[0000000000000000000000000000000000000000000000000
```

0000000000000011111111111111111111111111

11111111111111111111111111111111112222222222

222

 $2\ 2]$

Confusion Matrix

[[12 0 0]]

$$[0 \ 14 \ 0]$$
$$\begin{bmatrix} 0 & 0 & 19 \end{bmatrix}$$

Accuracy Metrics

```
precision  recall  f1-score  support
```

0	1.00	1.00	1.00	12
---	------	------	------	----

1	1.00	1.00	1.00	14
---	------	------	------	----

2	1.00	1.00	1.00	19
---	------	------	------	----

accuracy			1.00	45
----------	--	--	------	----

macro avg	1.00	1.00	1.00	45
-----------	------	------	------	----

weighted avg	1.00	1.00	1.00	45
--------------	------	------	------	----

`+*ln[]:*+`

`[source, ipython3]`

```
In [1]: from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

iris=datasets.load_iris()

x = iris.data
y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0- Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

#To make predictions on our test data
y_pred=classifier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5. 3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3. 1.4 0.1]
 [4.3 3. 1.1 0.1]]
```


Lab Program -10 :-

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Source code and output :-

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

```
[source, ipython3]
```

```
----
```

```
import numpy as np
```

```
from bokeh.plotting import figure, show, output_notebook
```

```
from bokeh.layouts import gridplot
```

```
from bokeh.io import push_notebook
```

```
from matplotlib import pyplot as plt
```

```
def local_regression(x0, X, Y, tau):# add bias term
```

```
    x0 = np.r_[1, x0] # Add one to avoid the loss in information
```

```
    X = np.c_[np.ones(len(X)), X]
```

```
    # fit model: normal equations with kernel
```

```
    xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W
```

```
    beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product
```

```

# predict value
return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernal Bias Function

n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])
def plot_lwr(tau):
    # prediction through regression
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot

```

```
show(gridplot([
    [plot_lwr(10.), plot_lwr(1.)],
    [plot_lwr(0.1), plot_lwr(0.01)]]))
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
```

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

The Data Set (10 Samples) X :

```
[-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
-2.95795796 -2.95195195 -2.94594595]
```

The Fitting Curve Data Set (10 Samples) Y :

```
[2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
2.11015444 2.10584249 2.10152068]
```

Normalised (10 Samples) X :

```
[-2.88440998 -2.97461063 -2.97639127 -2.9042727 -3.1194782 -3.06506157
-2.8349021 -2.90676221 -2.92454458]
```

Xo Domain Space(10 Samples) :

```
[-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
-2.85953177 -2.83946488 -2.81939799]
```

```
Text(0.5, 0, 'Petal Length')
```

```
![png](output_0_2.png)
```

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

```
[source, ipython3]
```

```
----
```

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

```
import pandas as pd
```

```
import numpy as np
```

```
def kernel(point,xmat, k):
```

```
    m,n = np.shape(xmat)
```

```
    weights = np.mat(np.eye((m))) # eye - identity matrix
```

```
    for j in range(m):
```

```
        diff = point - X[j]
```

```
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
```

```
    return weights
```

```
def localWeight(point,xmat,yamat,k):
```

```
    wei = kernel(point,xmat,k)
```

```
    W = (X.T*(wei*X)).I*(X.T*(wei*yamat.T))
```

```
    return W
```

```
def localWeightRegression(xmat,yamat,k):
```

```
    m,n = np.shape(xmat)
```

```
    ypred = np.zeros(m)
```

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



```
ypred[i] = xmat[i]*localWeight(xmat[i],xmat,yamat,k)
```

```
return ypred
```

```
def graphPlot(X,ypred):
```

```
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
```

```
    xsort = X[sortindex][:,0]
```

```
    fig = plt.figure()
```

```
    ax = fig.add_subplot(1,1,1)
```

```
    ax.scatter(bill,tip, color='green')
```

```
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
```

```
    plt.xlabel('Total bill')
```

```
    plt.ylabel('Tip')
```

```
    plt.show();
```

```
# load data points
```

```
data = pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 10\tips.csv')
```

```
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
```

```
tip = np.array(data.tip)
```

```
mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
```

```
mtip = np.mat(tip)
```

```
m= np.shape(mbill)[1]
```

```
one = np.mat(np.ones(m))
```

```
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols
```

```
# increase k to get smooth curves
```

```
ypred = localWeightRegression(X,mtip,3)
```

```
graphPlot(X,ypred)
```

```
----
```

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

```
----
```

```
![png](output_1_0.png)
```

```
----
```

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

```
[source, ipython3]
```

```
----
```

	A	B	C	D	E	F	G	H	I
1	total_bill	tip	sex	smoker	day	time	size		
2	16.99	1.01	Female	No	Sun	Dinner	2		
3	10.34	1.66	Male	No	Sun	Dinner	3		
4	21.01	3.5	Male	No	Sun	Dinner	3		
5	23.68	3.31	Male	No	Sun	Dinner	2		
6	24.59	3.61	Female	No	Sun	Dinner	4		
7	25.29	4.71	Male	No	Sun	Dinner	4		
8	8.77	2	Male	No	Sun	Dinner	2		
9	26.88	3.12	Male	No	Sun	Dinner	4		
10	15.04	1.96	Male	No	Sun	Dinner	2		
11	14.78	3.23	Male	No	Sun	Dinner	2		
12	10.27	1.71	Male	No	Sun	Dinner	2		
13	35.26	5	Female	No	Sun	Dinner	4		
14	15.42	1.57	Male	No	Sun	Dinner	2		
15	18.43	3	Male	No	Sun	Dinner	4		
16	14.83	3.02	Female	No	Sun	Dinner	2		
17	21.58	3.92	Male	No	Sun	Dinner	2		
18	10.33	1.67	Female	No	Sun	Dinner	3		
19	16.29	3.71	Male	No	Sun	Dinner	3		
20	16.97	3.5	Female	No	Sun	Dinner	3		
21	20.65	3.35	Male	No	Sat	Dinner	3		
22	17.92	4.08	Male	No	Sat	Dinner	2		
23	20.29	2.75	Female	No	Sat	Dinner	2		
24	15.77	2.23	Female	No	Sat	Dinner	2		
25	39.42	7.58	Male	No	Sat	Dinner	4		

tips



```

In [2]: import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook
from matplotlib import pyplot as plt

def local_regression(x0, X, Y, tau):# add bias term
    x0 = np.r_[1, x0] # Add one to avoid the loss in information
    X = np.c_[np.ones(len(X)), X]

    # fit model: normal equations with kernel
    xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W

    beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product

    # predict value
    return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernel Bias Function

n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])
def plot_lwr(tau):
    # prediction through regression
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot

```

```

plot.title.text= tau=%g % tau
plot.scatter(X, Y, alpha=.3)
plot.line(domain, prediction, line_width=2, color='red')
return plot

show(gridplot([
    [plot_lwr(10.), plot_lwr(1.)],
    [plot_lwr(0.1), plot_lwr(0.01)]]))
plt.title('K Mean Classification')
plt.xlabel('Petal Length')

```

The Data Set (10 Samples) X :

```

[-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
-2.95795796 -2.95195195 -2.94594595]

```

The Fitting Curve Data Set (10 Samples) Y :

```

[2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
2.11015444 2.10584249 2.10152068]

```

Normalised (10 Samples) X :

```

[-2.88440998 -2.97461063 -2.97639127 -2.9042727 -3.1194782 -3.06506157
-2.8349021 -2.90676221 -2.92454458]

```

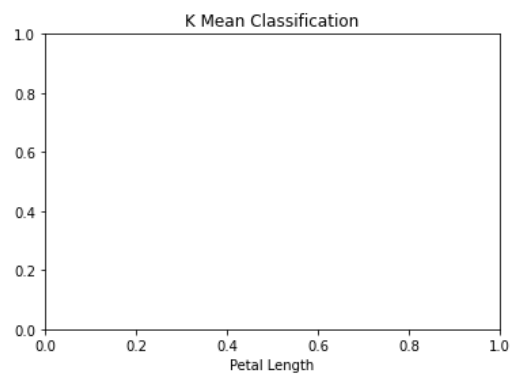
Xo Domain Space(10 Samples) :

```

[-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
-2.85953177 -2.83946488 -2.81939799]

```

Out[2]: Text(0.5, 0, 'Petal Length')



```

In [3]: import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,yamat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*yamat.T))
    return W

def localWeightRegression(xmat,yamat,k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,yamat,k)
    return ypred

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();

# Load data points
data = pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 10\tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m = np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols
plt.ylabel('Tip')
plt.show();

# Load data points
data = pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 10\tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m = np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

# increase k to get smooth curves
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)

```

