B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

Lab Record

MACHINE LEARNING

Submitted in partial fulfillment for the 6th Semester Laboratory

Bachelor of Technology
in
Computer Science and Engineering

Submitted by:
ANKITHA

1BM18CS016

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
Mar-June 2021

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Machine Learning (20CS6PCMAL) laboratory has been carried out by **Ankitha(1BM16CS016)**during the 6th Semester Mar-June-2021.

Signature of the Faculty Incharge:

NAME OF THE FACULTY:
Dr. ASHA GR
Department of Computer Science and Engineering
B.M.S. College of Engineering, Bangalore

TABLE OF CONTENT

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

```
import pandas as pd
import numpy as np
#to read the data in the csv file
data = pd.read_csv("/content/enjoysport.csv")
print(data)
#making an array of all the attributes
d = np.array(data)[:,:-1]
print("\n The attributes are : \n",d)
#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("\n The target is : ",target)
#training function to implement find-s algorithm
def train(c,t):

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

  return specific_hypothesis
```

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

```
     sky airtemp humidity    wind water forcast enjoysport
0  sunny    warm   normal  strong  warm    same        yes
1  sunny    warm     high  strong  warm    same        yes
2  rainy    cold     high  strong  warm  change         no
3  sunny    warm     high  strong  cool  change        yes

 The attributes are :
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

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

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

Fig 1.1 Data and output

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.**

```
import numpy as np
import pandas as pd
data = pd.read_csv('/content/enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ",
specific_h)
        print("Generic Boundary after ", i+1, "Instance is ",
general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?',
'?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

```
Specific Boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?'

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',



Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',



Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
Specific Bundary after  3 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  3 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?'



Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
Specific Bundary after  4 Instance is  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary after  4 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?'


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

Fig 2.1 Output

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.**

```
Import pandas as pd
import math
import numpy as np
import pprint

data=pd.read_csv("../input/dataset-id3/dataset.csv")
print("\n Input Data Set is:\n", data)
features = [f for f in data]
features.remove("answer")

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

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

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain = find_entropy(examples)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        sub_e = find_entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    return gain
```

```
def id3(examples, attrs):
  root = Node()

  max_gain = 0
  max_feat = “”
  for feature in attrs:
      gain = info_gain(examples, feature)
      if gain > max_gain:
          max_gain = gain
          max_feat = feature
  root.value = max_feat
  uniq = np.unique(examples[max_feat])
  for u in uniq:
      subdata = examples[examples[max_feat] == u]
      if find_entropy(subdata) == 0.0:
          newNode = Node()
          newNode.isLeaf = True
          newNode.value = u
          newNode.pred = np.unique(subdata[“answer”])
          root.children.append(newNode)
      else:
          tempNode = Node()
          tempNode.value = u
          new_attrs = attrs.copy()
          new_attrs.remove(max_feat)
          child = id3(subdata, new_attrs)
          tempNode.children.append(child)
          root.children.append(tempNode)
  return root

def printTree(root: Node, depth=0):
    for I in range(depth):
        print(“\t”, end=“”)
    print(root.value, end=“”)
    if root.isLeaf:
        print(“ : “, root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

root = id3(data, features)
print(“Final decision tree:\n”)
printTree(root)
```

```
 Input Data Set is:
     outlook temperature humidity    wind answer
0      sunny         hot     high    weak     no
1      sunny         hot     high  strong     no
2   overcast         hot     high    weak    yes
3       rain        mild     high    weak    yes
4       rain        cool   normal    weak    yes
5       rain        cool   normal  strong     no
6   overcast        cool   normal  strong    yes
7      sunny        mild     high    weak     no
8      sunny        cool   normal    weak    yes
9       rain        mild   normal    weak    yes
10     sunny        mild   normal  strong    yes
11  overcast        mild     high  strong    yes
12  overcast         hot   normal    weak    yes
13      rain        mild     high  strong     no
Final decision tree:

outlook
        overcast :  ['yes']

        rain
                wind
                        strong :  ['no']

                        weak :  ['yes']

        sunny
                humidity
                        high :  ['no']

                        normal :  ['yes']
```

Fig 3.1 Input data and output

4. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import pandas as pd
import csv
import random
import math

def read_csv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

#splitting the dataset into train and test data
def split_dataset(dataset, splitratio):
    trainsize = int(len(dataset) * splitratio);
    trainset = []
    copy = list(dataset);
    while len(trainset) < trainsize:
        index = random.randrange(len(copy));
        trainset.append(copy.pop(index))
    return [trainset, copy]

def separate_by_class(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated

def mean(numbers):
    return sum(numbers)/float(len(numbers))
def mean(numbers):
    return sum(numbers)/float(len(numbers))

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

def summarize(dataset):
    summaries = [(mean(attribute), std_dev(attribute)) for attribute
in zip(*dataset)];
    del summaries[-1] #excluding labels +ve or -ve
    return summaries

def summarize_by_class(dataset):
    separated = separate_by_class(dataset);
```

```
    summaries = {}
    for classvalue, instances in separated.items():
        summaries[classvalue] = summarize(instances)
    return summaries

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

# probabilities contains the all prob of all class of test data
def calculate_class_probabilities(summaries, inputvector):
    probabilities = {}
    for classvalue, classsummaries in summaries.items():
        probabilities[classvalue] = 1
    for i in range(len(classsummaries)):
        mean, stdev = classsummaries[i]
        x = inputvector[i]
        probabilities[classvalue]  *=  calculate_probability(x,  mean,
stdev)
    return probabilities
def  predict(summaries,  inputvector):     #training  and  test  data  is
passed
    probabilities     =     calculate_class_probabilities(summaries,
inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel

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

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

splitratio = 0.67
dataset = read_csv('/content/pima-indians-diabetes.csv');

trainingset, testset = split_dataset(dataset, splitratio)

print(f'Split  {len(dataset)}  rows  into  train={len(trainingset)}  and
test={len(testset)} rows')
```

```
summaries = summarize_by_class(trainingset);
#find the predictions of test data with the training data
predictions = get_predictions(summaries, testset)
accuracy = get_accuracy(testset, predictions)

print(f'The Accuracy of the classifier is :{accuracy} %')
```

OUTPUT

```
Split 768 rows into train=514 and test=254 rows
The Accuracy of the classifier is :70.47244094488188 %
```

Fig 4.1 output

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

```
!pip install pgmpy
```

```
Collecting pgmpy
  Downloading https://files.pythonhosted.org/packages/a3/0e/d9fadbfaa35e010c04d43acd3ae9fbefec98897dd7d61a6b7eb5a8b34072/pgmpy-0.1.14-py3-none-any.whl (331kB)
     |████████████████████████████████| 337kB 2.9MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.4.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.1.5)
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.8.1+cu101)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages (from pgmpy) (0.10.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.19.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from pgmpy) (0.22.2.post1)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.7/dist-packages (from pgmpy) (2.4.7)
Requirement already satisfied: networkx in /usr/local/lib/python3.7/dist-packages (from pgmpy) (2.5.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from pgmpy) (4.41.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.0.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas->pgmpy) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->pgmpy) (2.8.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch->pgmpy) (3.7.4.3)
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from statsmodels->pgmpy) (0.5.1)
Requirement already satisfied: decorator<5,>=4.3 in /usr/local/lib/python3.7/dist-packages (from networkx->pgmpy) (4.4.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas->pgmpy) (1.15.0)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.14
```

```python
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
heartDisease = pd.read_csv('/content/heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  ...  oldpeak  slope  ca  thal  heartdisease
0   63    1   1       145   233  ...      2.3      3   0     6             0
1   67    1   4       160   286  ...      1.5      2   3     3             2
2   67    1   4       120   229  ...      2.6      2   2     7             1
3   37    1   3       130   250  ...      3.5      3   0     3             0
4   41    0   2       130   204  ...      1.4      1   0     3             0

[5 rows x 14 columns]

 Attributes and datatypes
age                 int64
sex                 int64
cp                  int64
trestbps            int64
chol                int64
fbs                 int64
restecg             int64
thalach             int64
exang               int64
oldpeak           float64
slope               int64
ca                 object
thal               object
heartdisease        int64
dtype: object
```

Fig 5.1 Instance of data

```
model =
BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang
','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),(
'heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\nInferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
print('\n1.Probability of HeartDisease given evidence = restecg :')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence=
{'restecg':1})
print(q1)

print('\n2.Probability of HeartDisease given evidence = cp :')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence=
{'cp':2})
print(q2)
```

```
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 564.83it/s]
Eliminating: chol: 100%|██████████| 5/5 [00:00<00:00, 124.93it/s]
1.Probability of HeartDisease given evidence = restecg :
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+

2.Probability of HeartDisease given evidence = cp :

Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 480.62it/s]
Eliminating: chol: 100%|██████████| 5/5 [00:00<00:00, 242.86it/s]+-----------
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Fig 5.2 Output

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

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
dataset = pd.read_csv('/content/mall_customers.csv')
dataset.head()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Fig 6.1 Data

```
colormap = np.array(['red', 'lime',
'cyan','magenta','blue','purple'])
def kmeans(k,flag):
  if flag:
    x = dataset.iloc[:, [3, 4]].values
    plt.xlabel('Annual Income (k$)')
    plt.ylabel('Spending Score (1-100)')
  else:
    x = dataset.iloc[:, [2, 4]].values
    plt.xlabel('Age')
    plt.ylabel('Spending Score (1-100)')

  model = KMeans(n_clusters=k)
  y_predict= model.fit_predict(x)

  plt.title('K Mean Classification of customers')
  for i in range(0,k):
    plt.scatter(x[y_predict == i, 0], x[y_predict == i, 1], s = 40,
c = colormap[i])
  plt.scatter(model.cluster_centers_[:, 0],
model.cluster_centers_[:, 1], s = 100, c = 'black')
  plt.show()
```

```
#k=4 clusters based on age and spending score
kmeans(4,False)
```

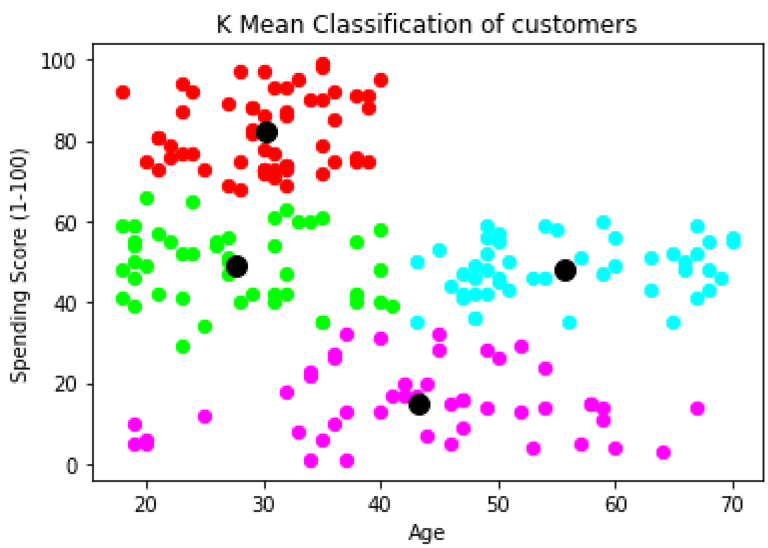

Fig 6.2 output

```
#k=5 clusters based on income and spending score
kmeans(5,True)
```

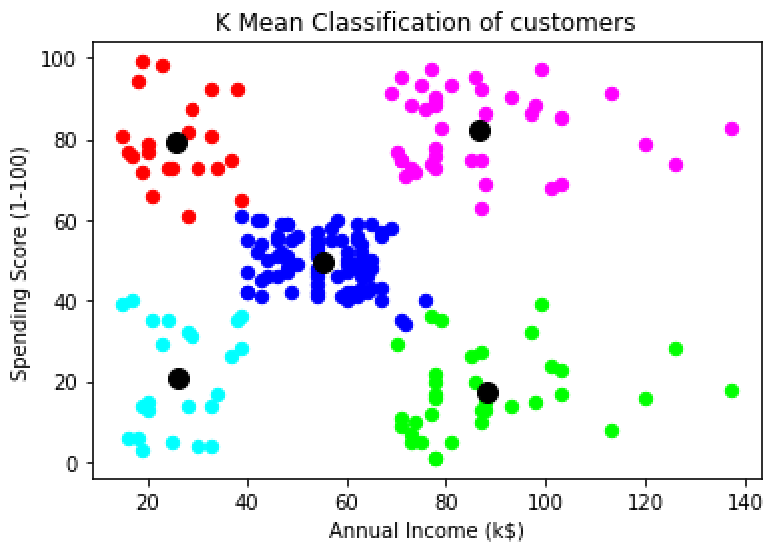

Fig 6.2 output

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

```
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
from sklearn import preprocessing



iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns                                                                   =
['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
print(X.sample(10))
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', 'blue', 'black','magenta'])

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')

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_))
```

```
     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
127           6.1          3.0           4.9          1.8
15            5.7          4.4           1.5          0.4
28            5.2          3.4           1.4          0.2
98            5.1          2.5           3.0          1.1
128           6.4          2.8           5.6          2.1
88            5.6          3.0           4.1          1.3
43            5.0          3.5           1.6          0.6
70            5.9          3.2           4.8          1.8
92            5.8          2.6           4.0          1.2
53            5.5          2.3           4.0          1.3
The accuracy score of K-Mean:  0.8933333333333333
```

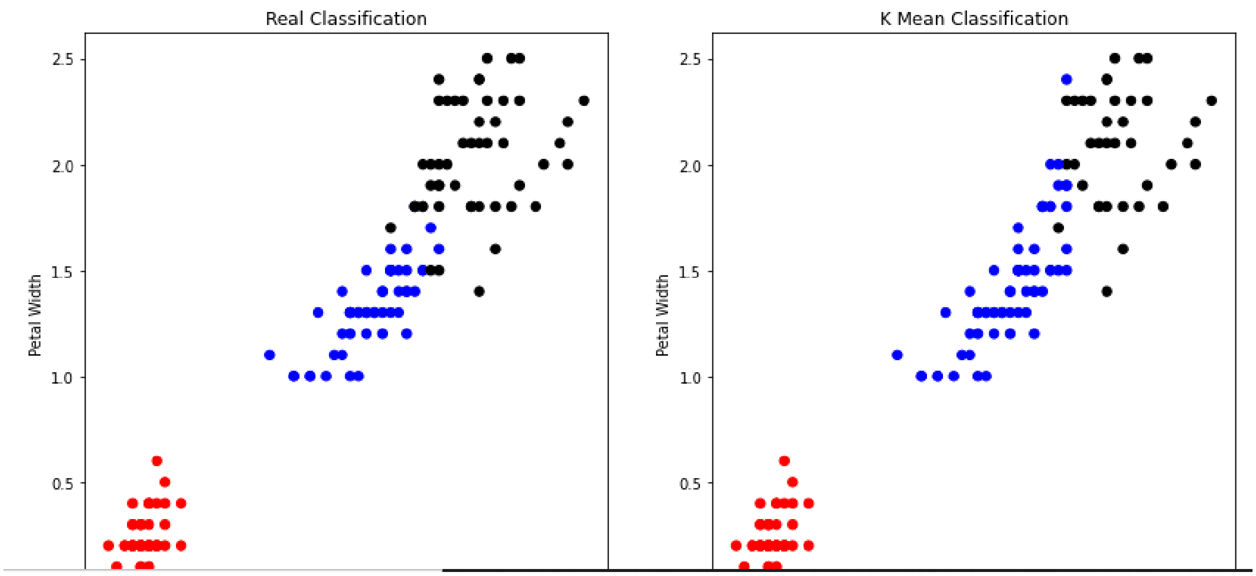

Fig 7.1 output

```
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
print(xs.sample(10))
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)

plt.figure(figsize=(14,7))
plt.subplot(1,2,2)
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))
```

```
90      -0.416010    -1.052767     0.364896     0.000878
70       0.068662     0.328414     0.592246     0.790671
95      -0.173674    -0.131979     0.251221     0.000878
30      -1.264185     0.098217    -1.226552    -1.315444
50       1.401508     0.328414     0.535409     0.264142
75       0.916837    -0.131979     0.364896     0.264142
24      -1.264185     0.788808    -1.056039    -1.315444
94      -0.294842    -0.822570     0.251221     0.132510
The accuracy score of EM:  0.3333333333333333
```

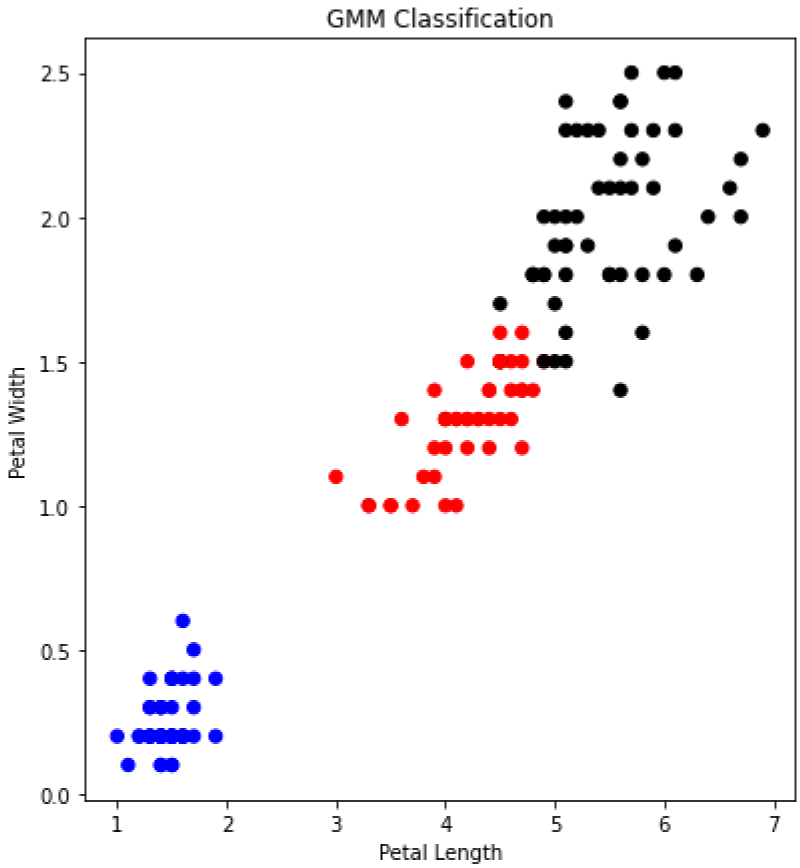

Fig 7.2 output

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

```
import sklearn
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier


iris=load_iris()
iris.keys()
df=pd.DataFrame(iris['data'])
print("The data looks like this:\n")
print(df)
print("\nThe Target Features are:\n")
print(iris['target_names'])
iris['feature_names']
X=df
y=iris['target']
```

```
The data looks like this:

       0    1    2    3
0    5.1  3.5  1.4  0.2
1    4.9  3.0  1.4  0.2
2    4.7  3.2  1.3  0.2
3    4.6  3.1  1.5  0.2
4    5.0  3.6  1.4  0.2
..   ...  ...  ...  ...
145  6.7  3.0  5.2  2.3
146  6.3  2.5  5.0  1.9
147  6.5  3.0  5.2  2.0
148  6.2  3.4  5.4  2.3
149  5.9  3.0  5.1  1.8

[150 rows x 4 columns]

The Traget Features are:

['setosa' 'versicolor' 'virginica']
```

Fig 8.1 Data

```
from sklearn.model_selection import train_test_split
X_train,   X_test,   y_train,   y_test   =   train_test_split(X,   y,
test_size=0.33, random_state=42)
#Training the model with Nearest neighbors K=3
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
y_pred=knn.predict(X_test)
cm=confusion_matrix(y_test,y_pred)
print("1. Confusion matrix:\n",cm)
print("2. Correct prediction",accuracy_score(y_test,y_pred))
print("3. Wrong prediction",(1-accuracy_score(y_test,y_pred)))
print('4. Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

OUTPUT

```
1. Confusion matrix:
 [[19  0  0]
 [ 0 15  0]
 [ 0  1 15]]
2. Correct predicition 0.98
3. Wrong predicition 0.020000000000000018
4. Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```

Fig 8.2 output

9. **Linear Regression**

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

#DATA

df = pd.read_csv('/content/kc_house_data.csv')
df.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	3	7	1180	0	1955
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	3	7	2170	400	1951
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	3	6	770	0	1933
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	5	7	1050	910	1965
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	3	8	1680	0	1987

Fig 9.1 Data

```
X = np.array(df['sqft_living']).reshape(-1, 1)
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

#MODEL

model = LinearRegression()
model.fit(X_train, y_train)
# model.fit(X, y)
```

```
LinearRegression()
```

```
#ANALYSIS

y_pred = model.predict(X_test)
print(f"Mean Squared Error: {mean_squared_error(y_pred, y_test)}")
```

```
Mean Squared Error: 76715223988.35832
```

```
#Visualizing the training Test Results
plt.scatter(X_train, y_train, color= 'red')
plt.plot(X_train, model.predict(X_train), color = 'blue')
plt.title ("Visuals for Training Dataset")
plt.xlabel("Space")
plt.ylabel("Price")
plt.show()
```

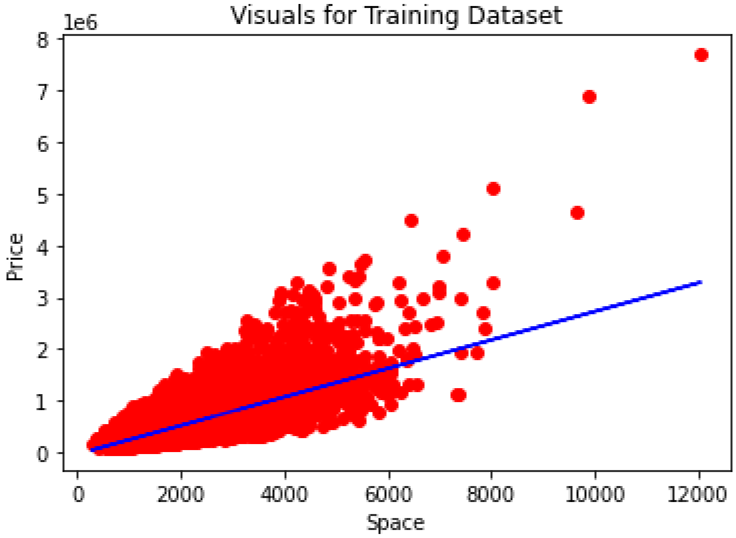

Fig 9.2 output

```
#Visualizing the Test Results
plt.scatter(X_test, y_test, color= 'red')
plt.plot(X_test, model.predict(X_test), color = 'blue')
plt.title("Visuals for Test DataSet")
plt.xlabel("Space")
plt.ylabel("Price")
plt.show()
```

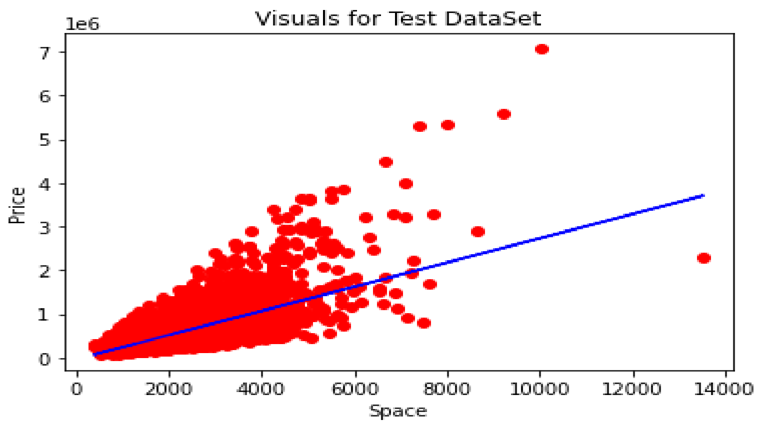

Fig 9.3 output

10. **Locally Weighted Regression**

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

# kernel smoothing function
def kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))

    for j in range(m):
        diff = point - X[j]
        weights[j, j] = np.exp(diff * diff.T / (-2.0 * k**2))

    return weights

# function to return local weight of eah traiining example
def localWeight(point, xmat, ymat, k):
    wt = kernel(point, xmat, k)
    W = (X.T * (wt*X)).I * (X.T * wt * ymat.T)
    return W

# root function that drives the algorithm
def localWeightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)

    for i in range(m):
        ypred[i] = xmat[i] * localWeight(xmat[i], xmat, ymat, k)

    return ypred

#DATA
#import data
data = pd.read_csv('/content/tips.csv)


# place them in suitable data types
colA = np.array(data.total_bill)
colB = np.array(data.tip)

mcolA = np.mat(colA)
mcolB = np.mat(colB)

m = np.shape(mcolB)[1]
one = np.ones((1, m), dtype = int)


# horizontal stacking
X = np.hstack((one.T, mcolA.T))
print(X.shape)
```

```
(244, 2)
```

#MODEL

```
# predicting values using LWLR
ypred = localWeightRegression(X, mcolB, 0.8)
```

#ANALYSIS

```
# plotting the predicted graph
xsort = X.copy()
xsort.sort(axis=0)
plt.scatter(colA, colB, color='red')
plt.plot(xsort[:,  1],  ypred[X[:,  1].argsort(0)],  color='blue',
linewidth=5)
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```

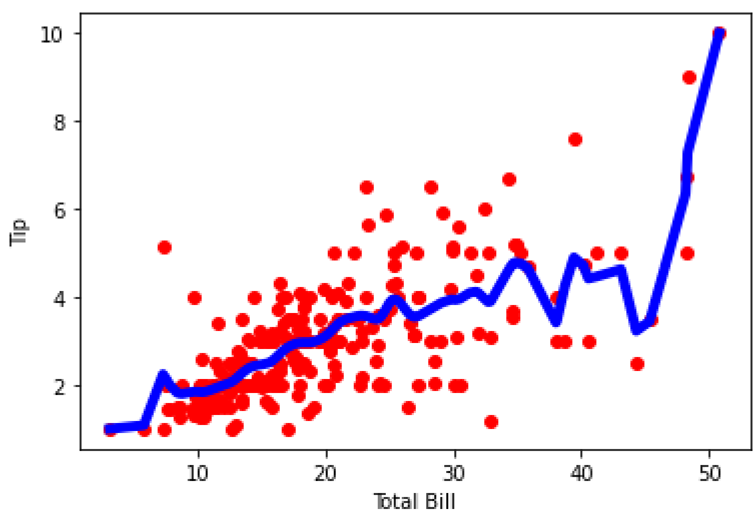

Fig 10.1 output