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:

Abhijnya K.G

1BM18CS002

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 **Abhijnya(1BM18CS002)**during the 6th Semester Mar-June-2021.

Signature of the Faculty In charge:

NAME OF THE FACULTY:

Dr. ASHA G R

Assistant Professor

Department of Computer Science and Engineering

B.M.S. College of Engineering, Bangalore

Table of Contents

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

```
import random
import csv


attributes = [['Sunny','Rainy'],
              ['Warm','Cold'],
              ['Normal','High'],
              ['Strong','Weak'],
              ['Same','Change']]

n = len(attributes)

a = []
print("\nGiven Data Set \n")

with open('../input/findsalgorithm/data set.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        a.append (row)
        print(row)


print("\n The initial hypothesis: ")
hypothesis = ['0'] * n
print(hypothesis)

for i in range(0,n):
        hypothesis[i] = a[0][i];

# Comparing with Remaining Training Examples of Given Data Set

print("\n Find S: Finding a Maximally Specific Hypothesis\n")
for i in range(0,len(a)):
    if a[i][n]=='Yes':
            for j in range(0,n):
                #If attributes is not same replace it with ?
                if a[i][j]!=hypothesis[j]:
                    hypothesis[j]='?'
                else :
```

```
                    hypothesis[j]= a[i][j]
    print(" For Training Example No :",i , " the hypothesis is ",hypothesi
s)

print("\n The Maximally Specific Hypothesis for a given Training Examples
:\n")
print(hypothesis)
```

Output:

```
  Weather Temperature Humidity    Wind Goes
0   Sunny        Warm     Mild  Strong  Yes
1   Rainy        Cold     Mild  Normal   No
2   Sunny    Moderate    Nomal  Normal  Yes
3   Sunny        Cold     High  Strong  Yes


 The attributes are:  [['Sunny' 'Warm' 'Mild' 'Strong']
 ['Rainy' 'Cold' 'Mild' 'Normal']
 ['Sunny' 'Moderate' 'Nomal' 'Normal']
 ['Sunny' 'Cold' 'High' 'Strong']]

 The target is:  ['Yes' 'No' 'Yes' 'Yes']

 The final hypothesis is: ['Sunny' '?' '?' '?']
```

Fig 1.1

CSV File:

1	Weather	Temperature	Humidity	Wind	Goes
2	Sunny	Warm	Mild	Strong	Yes
3	Rainy	Cold	Mild	Normal	No
4	Sunny	Moderate	Nomal	Normal	Yes
5	Sunny	Cold	High	Strong	Yes

Fig 1.2

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('../input/dataset/candidate_algo.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)
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '
?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
```

```python
    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")
```

Output:

```
PS C:\Users\KTGURUMURTY\Desktop> py candidate_elimination.py

Intializing General and specific hypothesis:

 General[0]: {('?', '?', '?', '?', '?', '?')}
 Specific[0]: {('0', '0', '0', '0', '0', '0')}


Instance 1 : ('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Y')

 General[1]: {('?', '?', '?', '?', '?', '?')}
 Specific[1]: {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}


Instance 2 : ('Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Y')

 General[2]: {('?', '?', '?', '?', '?', '?')}
 Specific[2]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}


Instance 3 : ('Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'N')

 General[3]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'Same')}
 Specific[3]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}


Instance 4 : ('Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Y')

 General[4]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}
 Specific[4]: {('Sunny', 'Warm', '?', 'Strong', '?', '?')}
PS C:\Users\KTGURUMURTY\Desktop>
```

Fig 2.1

CSV File:

1	Sunny	Warm	Normal	Strong	Warm	Same	Y
2	Sunny	Warm	High	Strong	Warm	Same	Y
3	Rainy	Cold	High	Strong	Warm	Change	N
4	Sunny	Warm	High	Strong	Cool	Change	Y

Fig 2.2

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("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)
```

Output:

```
Final decision tree:

outlook
        overcast :  ['yes']

        rain
                wind
                        strong :  ['no']

                        weak :  ['yes']

        sunny
                humidity
                        high :  ['no']

                        normal :  ['yes']
```

Fig 3.1

CSV File:

outlook	temperature	humidity	wind	answer
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes
rain	cool	normal	weak	yes
rain	cool	normal	strong	no
overcast	cool	normal	strong	yes
sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rain	mild	normal	weak	yes
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no

Fig 3.2

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 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 zi
p(*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('../input/naive-bayes/data_set.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)
```

Output:

```
Split 768 rows into train=514 and test=254 rows
The Accuracy of the classifier is :32.677165354330704 %
```

Fig 4.1

CSV File:

num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1

Fig 4.2

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

```
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

#read Cleveland Heart Disease data
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

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

#display the Attributes names and datatyes
print('\n Attributes and datatypes')
print(heartDisease.dtypes)

#Creat Model- Bayesian Network
model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exa
ng','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('hea
rtdisease','chol')])

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

# Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

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

Output:

```
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
```

```
 Learning CPD using Maximum likelihood estimators
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 834.39it/s]
Eliminating: sex: 100%|██████████| 5/5 [00:00<00:00, 70.42it/s]
 Inferencing with Bayesian Network:

 1.Probability of HeartDisease given evidence= restecg :1
+-----------------+---------------------+
| 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:2
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 605.29it/s]
Eliminating: sex: 100%|██████████| 5/5 [00:00<00:00, 138.98it/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 5.1

CSV File:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
53	1	4	140	203	1	2	155	1	3.1	3	0	7	1
57	1	4	140	192	0	0	148	0	0.4	2	0	6	0
56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
56	1	3	130	256	1	2	142	1	0.6	2	1	6	2
44	1	2	120	263	0	0	173	0	0	1	0	7	0

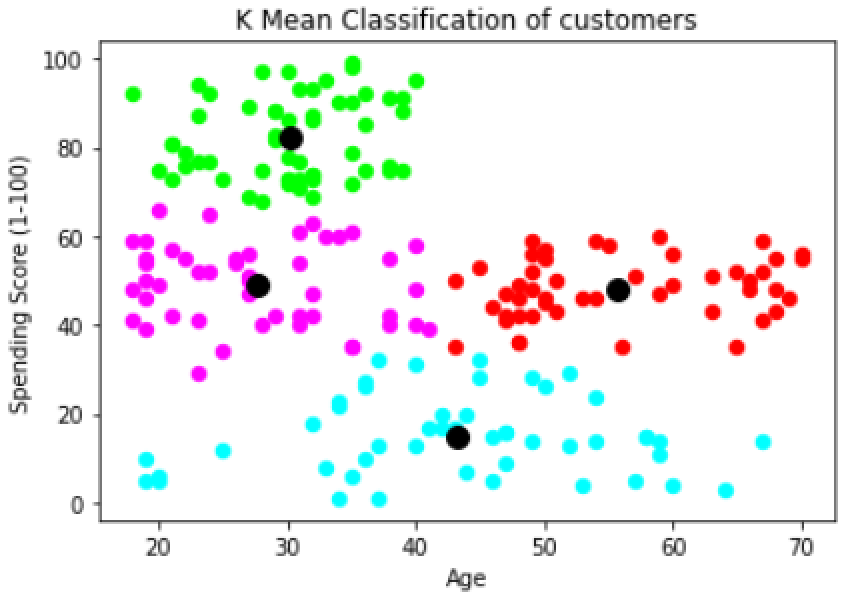
Fig 5.2

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('mall_customers.csv')
dataset.head()


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 = co
lormap[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)


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

Output:

```
In [9]: #k=4 clusters based on age and spending score
        kmeans(4,False)
```




```
In [10]: #k=5 clusters based on income and spending score
         kmeans(5,True)
```

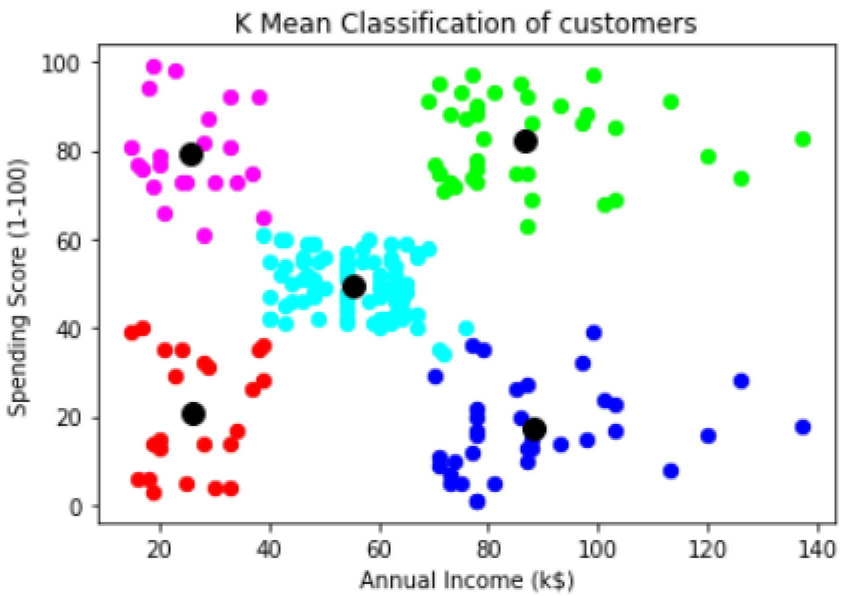


Fig 6.1

CSV File:

CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3
10	Female	30	19	72
11	Male	67	19	14
12	Female	35	19	99
13	Female	58	20	15
14	Female	24	20	77
15	Male	37	20	13

Fig 5.2

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.

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class
']

dataset = pd.read_csv("dataset.csv", names=names)

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

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))
colormap=np.array(['purple','lime','black'])


plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])


model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-
Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-
Mean:\n',metrics.confusion_matrix(y, model.labels_))


gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])
```

```
print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm
))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_clust
er_gmm))
```

Output:

```
The accuracy score of K-Mean:  0.09333333333333334
The Confusion matrixof K-Mean:
 [[ 0 50  0]
 [ 2  0 48]
 [36  0 14]]
The accuracy score of EM:  0.9666666666666667
The Confusion matrix of EM:
  [[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
```

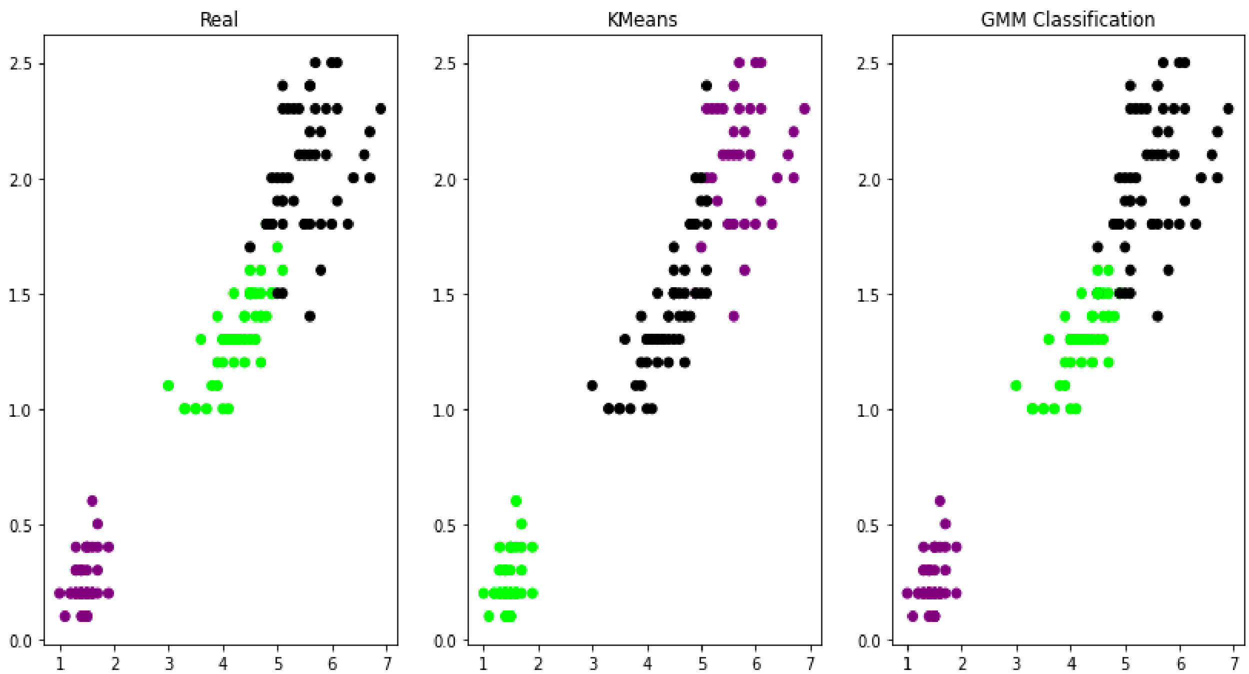


Fig 7.1

Data set:

1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3	1.4	0.1	Iris-setosa
14	4.3	3	1.1	0.1	Iris-setosa
15	5.8	4	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
19	5.7	3.8	1.7	0.3	Iris-setosa
20	5.1	3.8	1.5	0.3	Iris-setosa

Fig 7.2

8. Write a program to implement k-Nearest Neighbor 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 Traget Features are:\n")
print(iris['target_names'])
iris['feature_names']
X=df
y=iris['target']


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 nighbors 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 predicition",accuracy_score(y_test,y_pred))
print("3. Wrong predicition",(1-accuracy_score(y_test,y_pred)))
print('4. Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

Output:

```
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']
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

```
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 8.1

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

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('salary_dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3,
random_state=0)
```

```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)


# Visualizing the Training set results viz_train = plt
viz_train.scatter(X_train, y_train, color='red') viz_train.plot(X_train,
regressor.predict(X_train), color='green') viz_train.title('Salary VS
Experience (Training set)') viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary') viz_train.show()

# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='blue')
viz_test.plot(X_train, regressor.predict(X_train), color='green')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

Output:

Fig 9.1

Data set:

YearsExperience	Salary
1.1	39343
1.3	46205
1.5	37731
2.0	43525
2.2	39891
2.9	56642
3.0	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4.0	55794
4.0	56957
4.1	57081
4.5	61111
4.9	67938
5.1	66029

Fig 9.2

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.

```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr


def kernel(point,xmat, k):
 m,n = np1.shape(xmat)
 weights = np1.mat(np1.eye((m)))
 for j in range(m):
    diff = point - X[j]
    weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
 return weights


def localWeight(point,xmat,ymat,k):
 wei = kernel(point,xmat,k)
 W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
 return W
def localWeightRegression(xmat,ymat,k):
 m,n = np1.shape(xmat)
 ypred = np1.zeros(m)
 for i in range(m):
    ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
 return ypred


# load data points
data = pd.read_csv('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
```

```
#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimension
al array form
m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
#print(X)
#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]



fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='purple')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

Output:

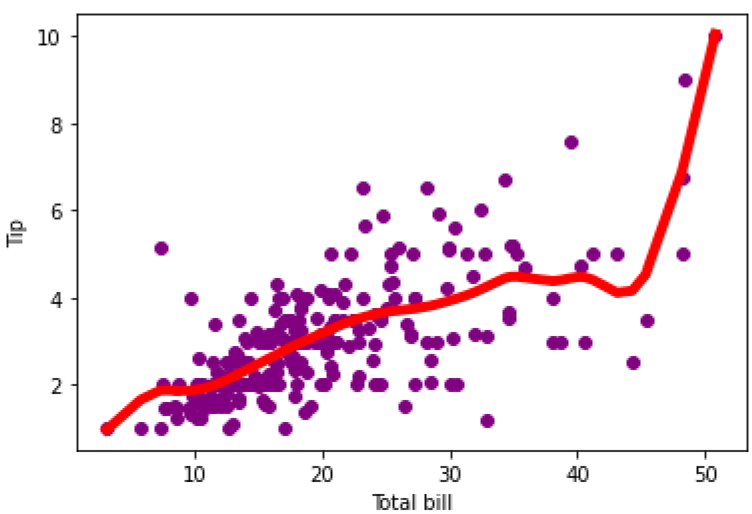


Fig 10.1

Data set:

total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2.0	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2
14.78	3.23	Male	No	Sun	Dinner	2
10.27	1.71	Male	No	Sun	Dinner	2
35.26	5.0	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3.0	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2
21.58	3.92	Male	No	Sun	Dinner	2
10.33	1.67	Female	No	Sun	Dinner	3
16.29	3.71	Male	No	Sun	Dinner	3
16.97	3.5	Female	No	Sun	Dinner	3

Fig 10.2