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:

NILANSHU RANJAN
1BM18CS062

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 NILANSHU RANJAN (1BM18CS062) during the 6th Semester Mar-June-2021.

Signature of the Faculty Incharge:
Dr. Asha G R
Department of Computer Science and Engineering
B.M.S. College of Engineering, Bangalore

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

DATASET

1	Sunny	Mild	High	Strong	Same	Yes
2	Rainy	Hot	High	Normal	Same	No
3	Sunny	Mild	Normal	Strong	Change	Yes
4	Sunny	Hot	High	Strong	Change	Yes
5	Sunny	Cool	Normal	Normal	Change	No
6	Overcast	Cool	Normal	Normal	Same	No
7	Rainy	Hot	Normal	Strong	Same	Yes

CODE

```
import numpy as np
import pandas as pd
def get_input(input_type, path):
    if input_type == "csv":
        df = pd.read_csv(path, header=None)
        print(df)
        train_set = df.values
        return train_set
    else:
        n = int(input("Enter the number of training examples: "))
        attr_no = int(input("Enter the number of attributes: "))
        train_set = [["" for _ in range(attr_no)] for __ in range(n)]
        for i in range(n):
            for j in range(attr_no):
                train_set[i][j] = input()
        return train_set


def find_s(train_set):
    hypo = ["phi" for _ in range(len(train_set[0])-1)]
    for i in range(len(train_set)):
        if train_set[i][-1] == "Yes":
            for j in range(len(train_set[0])-1):
                if hypo[j] != train_set[i][j] and hypo[j] == "phi":
```

```
            hypo[j] = train_set[i][j]
        elif hypo[j] != train_set[i][j] and hypo[j] != "phi":
            hypo[j] = "?"
    return hypo

train_set = get_input("csv", "../input/dataset/dataset.csv")
hypothesis = find_s(train_set)
print(hypothesis)
```

OUTPUT

```
        0     1       2       3     4       5    6
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
['Sunny', 'Warm', 'que', 'Strong', 'que', 'que']
```

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.

DATASET

	Outlook	Temperature	Humidity	Wind	Forecast	Enjoy sport
2	Sunny	Mild	High	Strong	Same	Yes
3	Rainy	Hot	High	Normal	Same	No
4	Sunny	Mild	Normal	Strong	Change	Yes
5	Sunny	Hot	High	Strong	Change	Yes
6	Sunny	Cool	Normal	Normal	Change	No
7	Overcast	Cool	Normal	Normal	Same	No
8	Rainy	Hot	Normal	Strong	Same	Yes

CODE

```
import numpy as np
import pandas as pd

data = pd.read_csv('../input/datasetforcea/data.csv', header=None)
train_data = np.array(data.iloc[:,0:-1])
```

```
print("Data :\n",train_data)
target = np.array(data.iloc[:,-1])
print("\nTarget Values : ",target)

def learn(train_data, target):
    s_h = train_data[0].copy()
    print("\nSpecific Boundary: ", s_h)
    g_h = [["?" for i in range(len(s_h))] for i in range(len(s_h))]
    print("\nGeneric Boundary: ",g_h)

    for i, h in enumerate(train_data):
        print("\nData instance", i+1 , "is ", h)
        if target[i] == "yes":
            for x in range(len(s_h)):
                if h[x]!= s_h[x]:
                    s_h[x] ='?'
                    g_h[x][x] ='?'

        if target[i] == "no":
            for x in range(len(s_h)):
                if h[x]!= s_h[x]:
                    g_h[x][x] = s_h[x]
                else:
                    g_h[x][x] = '?'

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

    indices = [i for i, val in enumerate(g_h) if val == ['?', '?', '?', '?', '?', '?']]
```

```
    for i in indices:
        g_h.remove(['?', '?', '?', '?', '?', '?'])
    return s_h, g_h


s_final, g_final = learn(train_data, target)


print("Final Specific hypothesis: ", s_final)
print("Final General hypothesis: ", g_final)
```

OUTPUT

```
\Data :
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values :  ['yes' 'yes' 'no' 'yes']

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

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

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



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



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



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


Final Specific hypothesis:  ['sunny' 'warm' '?' 'strong' '?' '?']
Final General hypothesis:  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

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.

DATASET

	outlook	temperature	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

CODE

```
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

```
  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']
```

Program 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.

DATASET

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
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0

CODE

```
import csv
import random
import math


def load_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


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 zip(*dataset)];
    del summaries[-1]
    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

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):
    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 = load_csv('../input/diabetes-data/Lab 4/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);
predictions = get_predictions(summaries, testset)
accuracy = get_accuracy(testset, predictions)

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

OUTPUT

```
Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is :67.71653543307087%
```

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

DATASET

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
2	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
3	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
4	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
5	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
6	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
7	56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
8	62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
9	57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
10	63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
11	53	1	4	140	203	1	2	155	1	3.1	3	0	7	1
12	57	1	4	140	192	0	0	148	0	0.4	2	0	6	0
13	56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
14	56	1	3	130	256	1	2	142	1	0.6	2	1	6	2
15	44	1	2	120	263	0	0	173	0	0	1	0	7	0
16	52	1	3	172	199	1	0	162	0	0.5	1	0	7	0
17	57	1	3	150	168	0	0	174	0	1.6	1	0	3	0
18	48	1	2	110	229	0	0	168	0	1	3	0	7	1
19	54	1	4	140	239	0	0	160	0	1.2	1	0	3	0
20	48	0	3	130	275	0	0	139	0	0.2	1	0	3	0
21	49	1	2	130	266	0	0	171	0	0.6	1	0	3	0
22	64	1	1	110	211	0	2	144	1	1.8	2	0	3	0

CODE

```
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

trainingData = pd.read_csv('/content/bayesian-dataset.csv')

trainingData = trainingData.replace('?',np.nan)

print('The sample instances from the dataset are:')

print(trainingData.head())

print('\n Attributes and datatypes: ')

print(trainingData.dtypes)

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

```
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(trainingData,estimator=MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
print('\n 2.Probability of HeartDisease given evidence = chol (Cholestorol): 100 ')
q2 = HeartDiseasetest_infer.query(variables = ['heartdisease'], evidence={'chol':100})
print(q2)
```

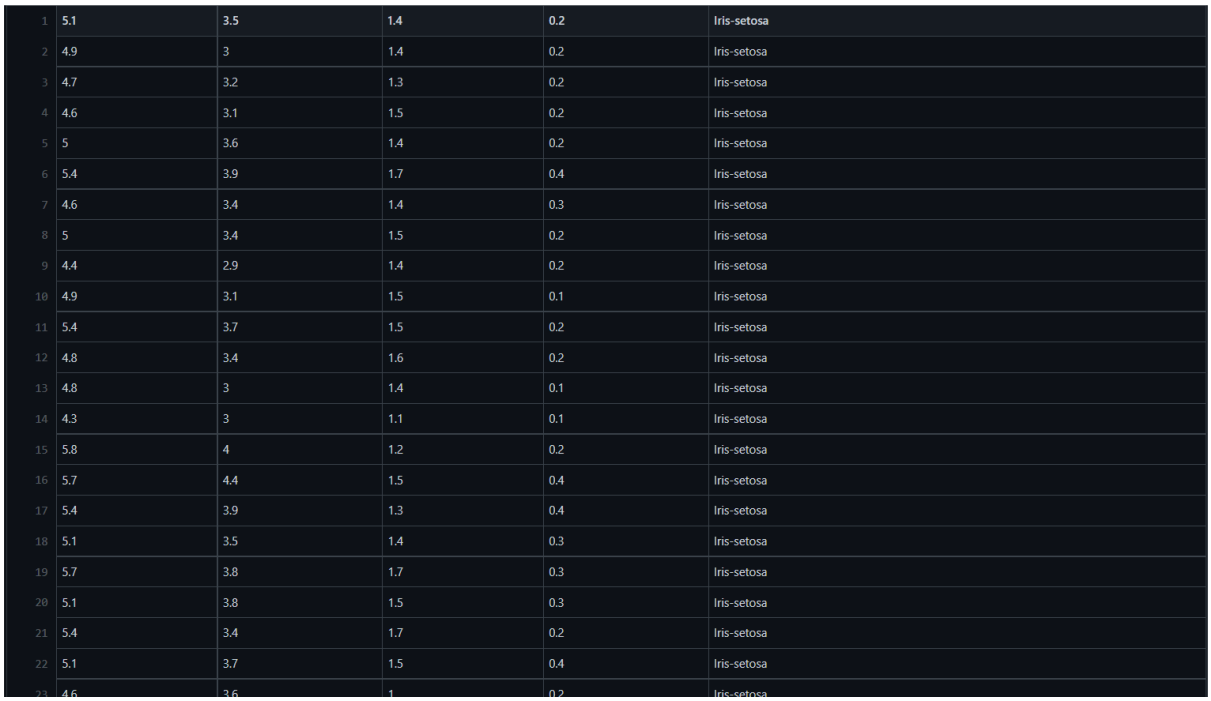
OUTPUT

```
 1.Probability of HeartDisease given evidence = restecg (Rest ECG): 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 = chol (Cholestorol): 100
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              1.0000 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.0000 |
+-----------------+---------------------+
| heartdisease(3) |              0.0000 |
+-----------------+---------------------+
| heartdisease(4) |              0.0000 |
+-----------------+---------------------+
```

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

DATASET



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
21	5.4	3.4	1.7	0.2	Iris-setosa
22	5.1	3.7	1.5	0.4	Iris-setosa
23	4.6	3.6	1	0.2	Iris-setosa

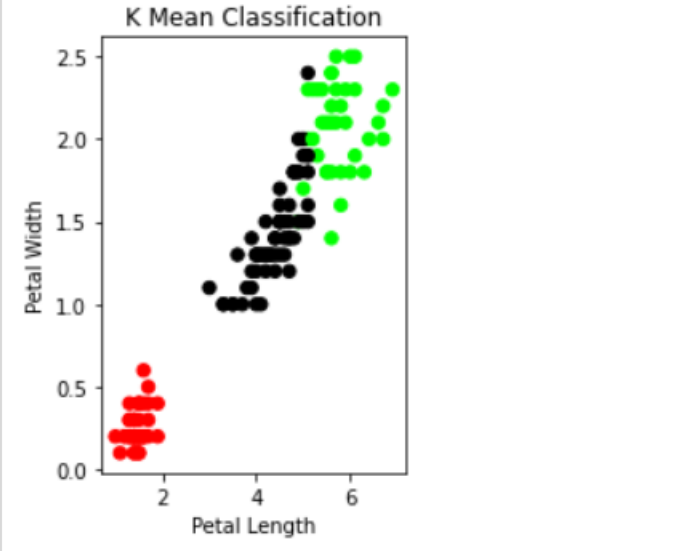
CODE

```
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'])
```

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

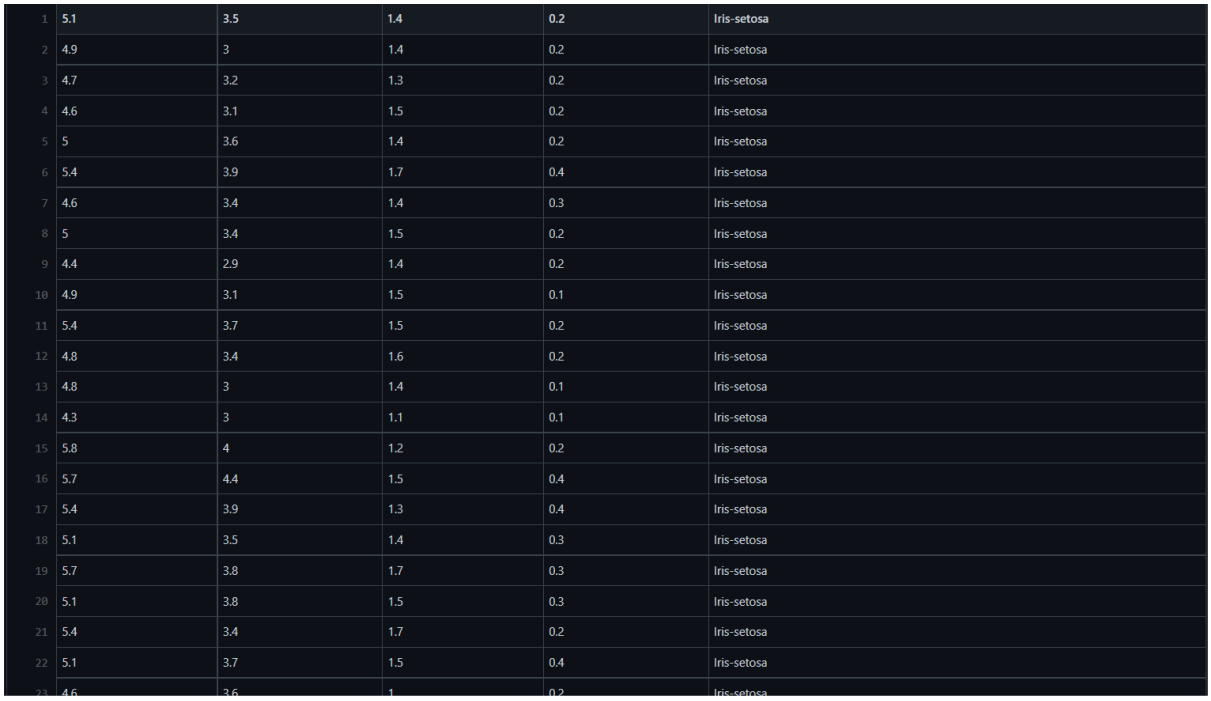
OUTPUT

```
The accuracy score of K-Mean:  0.44
The Confusion matrixof K-Mean:  [[50  0  0]
 [ 0  2 48]
 [ 0 36 14]]
```



Program 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.

DATASET



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
21	5.4	3.4	1.7	0.2	Iris-setosa
22	5.1	3.7	1.5	0.4	Iris-setosa
23	4.6	3.6	1	0.2	Iris-setosa

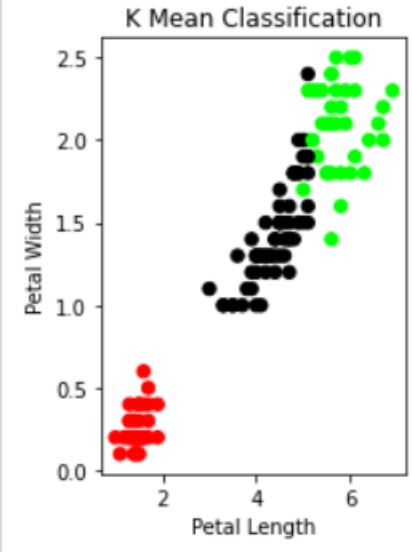
CODE

```
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))
```

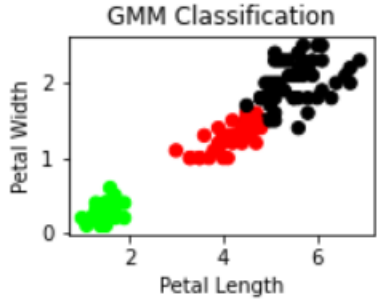
```
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], 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_))
print('The Confusion matrixof 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)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
y_gmm = gmm.predict(xs)
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))
```

OUTPUT

```
The accuracy score of K-Mean:  0.44
The Confusion matrixof K-Mean:  [[50  0  0]
 [ 0  2 48]
 [ 0 36 14]]
```




```
The accuracy score of EM:  0.3333333333333333
The Confusion matrix of EM:  [[ 0 50  0]
 [45  0  5]
 [ 0  0 50]]
```



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

CODE

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn import metrics

```
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv("/gdrive/MyDrive/8-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0
print ("\n-------------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("-------------------------------------------------------------------------")
for label in ytest:
    print ('%-25s %-25s' % (label, ypred[i]), end="")
    if (label == ypred[i]):
        print (' %-25s' % ('Correct'))
    else:
        print (' %-25s' % ('Wrong'))
    i = i + 1
print ("-------------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("-------------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("-------------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
```

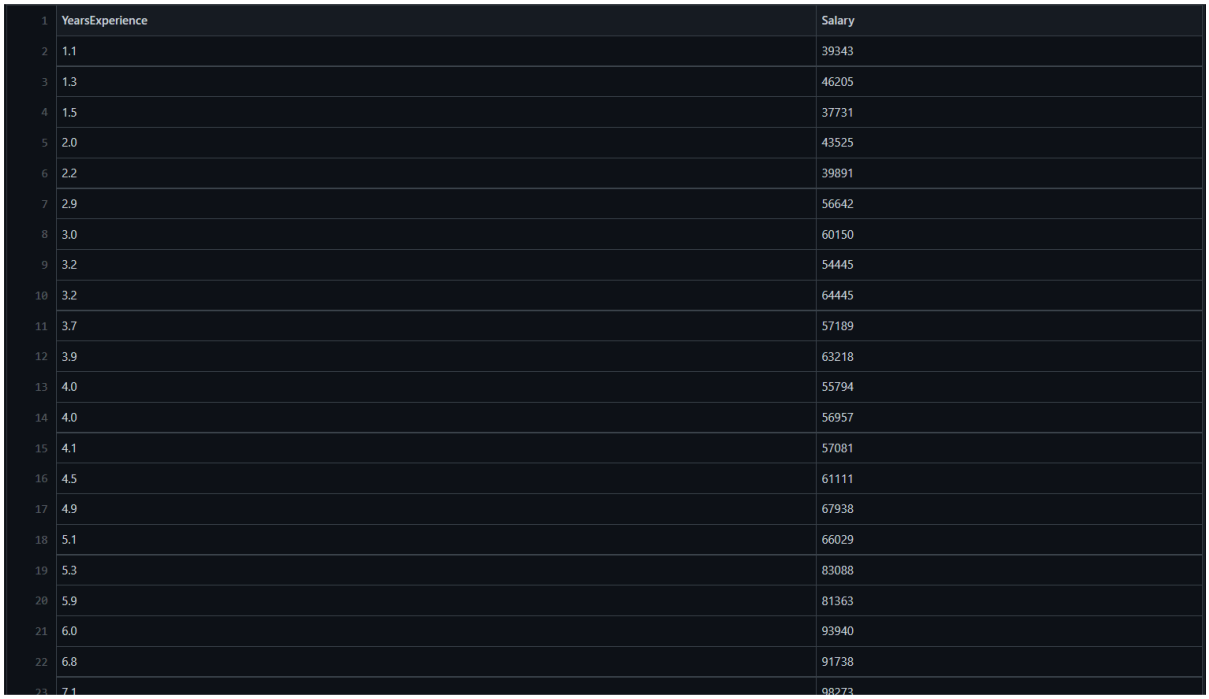
print ("--")

OUTPUT

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

------------------------------------------------------------------------
Original Label            Predicted Label           Correct/Wrong
------------------------------------------------------------------------
Iris-versicolor           Iris-versicolor           Correct
Iris-virginica            Iris-virginica            Correct
Iris-virginica            Iris-virginica            Correct
Iris-setosa               Iris-setosa               Correct
Iris-versicolor           Iris-virginica            Wrong
Iris-setosa               Iris-setosa               Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-setosa               Iris-setosa               Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-setosa               Iris-setosa               Correct
Iris-virginica            Iris-virginica            Correct
------------------------------------------------------------------------

Confusion Matrix:
 [[4 0 0]
 [0 7 1]
 [0 0 3]]
------------------------------------------------------------------------

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         4
Iris-versicolor       1.00      0.88      0.93         8
 Iris-virginica       0.75      1.00      0.86         3

       accuracy                           0.93        15
      macro avg       0.92      0.96      0.93        15
   weighted avg       0.95      0.93      0.94        15

------------------------------------------------------------------------
Accuracy of the classifer is 0.93
------------------------------------------------------------------------
```

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

DATASET



	YearsExperience	Salary
2	1.1	39343
3	1.3	46205
4	1.5	37731
5	2.0	43525
6	2.2	39891
7	2.9	56642
8	3.0	60150
9	3.2	54445
10	3.2	64445
11	3.7	57189
12	3.9	63218
13	4.0	55794
14	4.0	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	6.0	93940
22	6.8	91738
23	7.1	98273

CODE

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

OUTPUT

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.

DATASET

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
20.65	3.35	Male	No	Sat	Dinner	3
17.92	4.08	Male	No	Sat	Dinner	2
20.29	2.75	Female	No	Sat	Dinner	2

CODE

```
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)))
    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, 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 = 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 = pd.read_csv('/gdrive/MyDrive/10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

mbill = np.mat(bill)
mtip = np.mat(tip)

m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
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()
```

OUTPUT