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:

Dhanraj K
1BM18CS027

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 Dhanraj K(1BM18CS027) during the 6th Semester Mar-June-2021.

Signature of the Faculty Incharge:

Saritha A.N
Assistant Professor, CSE Dept.
Department of Computer Science and Engineering
B.M.S. College of Engineering, Bangalore

Table of Contents

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

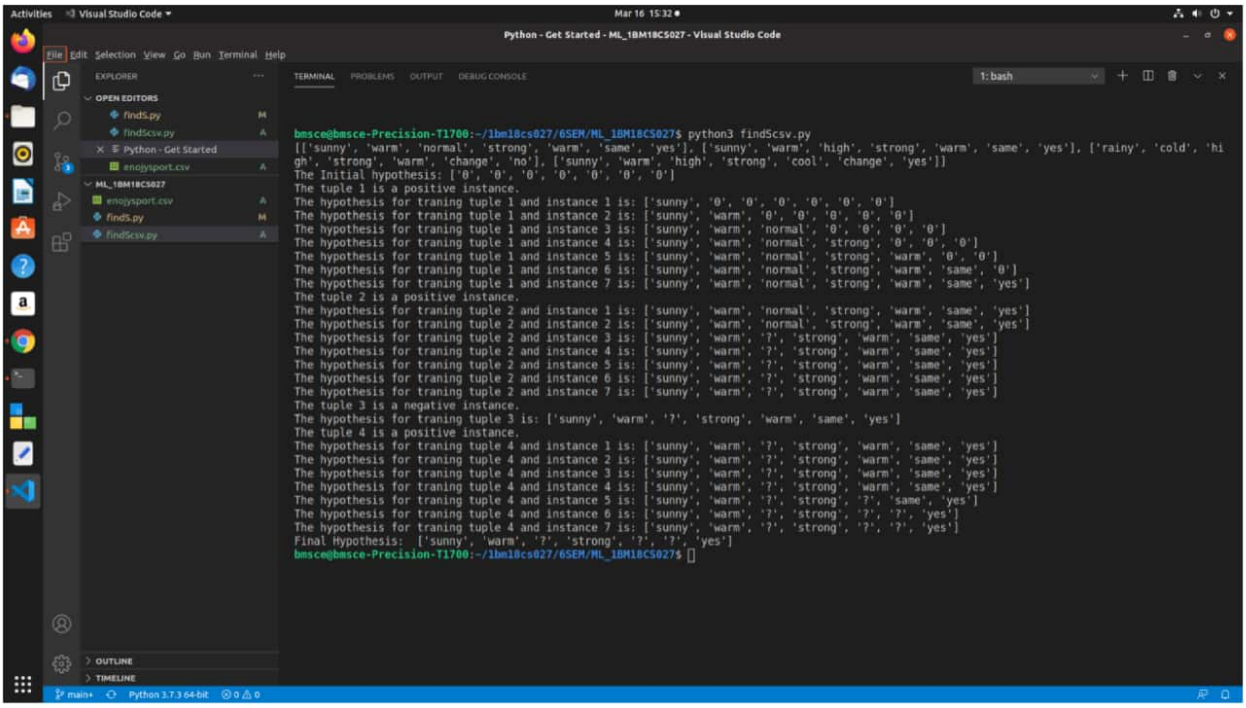
Program:

```
import csv

def findS(dataset, hypothesis):
    for i in range(len(dataset)):
        if dataset[i][-1] == 'yes':
            print('The tuple', i+1, 'is a positive instance.')
            for j in range(len(hypothesis)):
                if hypothesis[j] == '0'  or dataset[i][j] == hypothesis[j]:
                    hypothesis[j] = dataset[i][j]
                else:
                    hypothesis[j] = '?'
                print('The hypothesis for traning tuple',i+1,'and instance',j+1, 'is:', hypothesis)
        elif dataset[i][-1] == 'no':
            print('The tuple', i+1, 'is a negative instance.')
            print('The hypothesis for traning tuple',i+1, 'is:', hypothesis)
    return hypothesis


def main():

    dataset = []
    with open('enojysport.csv', 'r') as csvfile:
        next(csvfile)
        for row in csv.reader(csvfile):
            dataset.append(row)
        print(dataset)
    hypothesis = ['0']*len(dataset[0])
    print('The Initial hypothesis:', hypothesis)
    hypothesis = findS(dataset, hypothesis)
    print('Final Hypothesis: ', hypothesis)

if __name__ == "__main__":
    main()
```

Dataset:

	A	B	C	D	E	F	G
1	sky	airtemp	humidity	wind	water	forcast	enjoysport
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

Output:



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.

Program:

```
import numpy as np
import pandas as pd

data = pd.read_csv('/home/dhanrz/LAB/6SEM/ML_1BM18CS027/lab02/enojysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)

    for i, h in enumerate(concepts):
        print("For Loop Starts")
        if target[i] == "yes":
            print("If 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("If 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(" steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
        print("\n")
        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")
```

Dataset:

	A	B	C	D	E	F	G
1	sky	airtemp	humidity	wind	water	forcast	enjoysport
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

Output:

```
(venv) dhanrz@dhanrz-G7-7588:~/LAB/6SEM/ML_1BM18CS027/lab02$ python src.py
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Negative
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'sa
me']]



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'
]]



Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
(venv) dhanrz@dhanrz-G7-7588:~/LAB/6SEM/ML_1BM18CS027/lab02$
```

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.

```python
#ID3 algorithm
import math
import csv
def load_csv(filename):
    lines=csv.reader(open(filename,"r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers

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

def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic

```

```python

def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums

def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy

def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

```

```python
    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]


    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node

def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)


def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)
```

```python
dataset,features=load_csv("src_inp.csv")
node1=build_tree(dataset,features)

print("Decision tree: ")
print_tree(node1,0)
testdata,features=load_csv("src_inp_1.csv")

for test in testdata:
    print("The test instance:",test)
    print("The label for test instance:",end="   ")
    classify(node1,test,features)
```

Dataset:

	A	B	C	D	E
1	Outlook	Tempera	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

Output:

```
dhanrz@dhanrz-G7-7588:~/LAB/6SEM/ML_1BM18CS027/lab03$ python id3algo.py
Decision tree:
 Outlook
   sunny
     Humidity
       high
         no
       normal
         yes
   rain
     Wind
       weak
         yes
       strong
         no
   overcast
     yes
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:   no
The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance:   yes
dhanrz@dhanrz-G7-7588:~/LAB/6SEM/ML_1BM18CS027/lab03$
```

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.

Program :

```python
import csv
import random
import math

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

```python

def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
    del summaries[-1]
    return summaries

def summarizebyclass(dataset):
    separated = separatebyclass(dataset);
    summaries = {}
    for classvalue, instances in separated.items():
        summaries[classvalue] = summarize(instances)
    return summaries

def calculateprobability(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 calculateclassprobabilities(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] *= calculateprobability(x, mean, stdev)
    return probabilities

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

def getaccuracy(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

def main():
    filename = input('Enter Filename: ')
    splitratio = 0.67
    dataset = loadcsv(filename);

    trainingset, testset = splitdataset(dataset, splitratio)
    print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset), len(testset)))
    # prepare model
    summaries = summarizebyclass(trainingset);
    # test model
    predictions = getpredictions(summaries, testset)
    accuracy = getaccuracy(testset, predictions)
    print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

Dataset:

	A	B	C	D	E	F	G	H	I
1	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
2	6	148	72	35	0	33.6	0.627	50	1
3	1	85	66	29	0	26.6	0.351	31	0
4	8	183	64	0	0	23.3	0.672	32	1
5	1	89	66	23	94	28.1	0.167	21	0
6	0	137	40	35	168	43.1	2.288	33	1
7	5	116	74	0	0	25.6	0.201	30	0
8	3	78	50	32	88	31	0.248	26	1
9	10	115	0	0	0	35.3	0.134	29	0
10	2	197	70	45	543	30.5	0.158	53	1
11	8	125	96	0	0	0	0.232	54	1
12	4	110	92	0	0	37.6	0.191	30	0
13	10	168	74	0	0	38	0.537	34	1
14	10	139	80	0	0	27.1	1.441	57	0
15	1	189	60	23	846	30.1	0.398	59	1
16	5	166	72	19	175	25.8	0.587	51	1
17	7	100	0	0	0	30	0.484	32	1
18	0	118	84	47	230	45.8	0.551	31	1
19	7	107	74	0	0	29.6	0.254	31	1
20	1	103	30	38	83	43.3	0.183	33	0
21	1	115	70	30	96	34.6	0.529	32	1
22	3	126	88	41	235	39.3	0.704	27	0
23	8	99	84	0	0	35.4	0.388	50	0
24	7	196	90	0	0	39.8	0.451	41	1
25	9	119	80	35	0	29	0.263	29	1
26	11	143	94	33	146	36.6	0.254	51	1
27	10	125	70	26	115	31.1	0.205	41	1
28	7	147	76	0	0	39.4	0.257	43	1
29	1	97	66	15	140	23.2	0.487	22	0
30	13	145	82	19	110	22.2	0.245	57	0

Output:

```
(venv) dhanrz@dhanrz-G7-7588:~/LAB/6SEM/ML_1BM18CS027/lab04$ python src.py
Enter Filename: naivedata.csv
Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is : 72.04724409448819%
(venv) dhanrz@dhanrz-G7-7588:~/LAB/6SEM/ML_1BM18CS027/lab04$
```

Program 5:

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

Program:

```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('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)

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

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

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

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

Dataset:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	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	
23	58	0	1	150	283	1	2	162	0	1	1	0	3	0	
24	58	1	2	120	284	0	2	160	0	1.8	2	0	3	1	
25	58	1	3	132	224	0	2	173	0	3.2	1	2	7	3	
26	60	1	4	130	206	0	2	132	1	2.4	2	2	7	4	
27	50	0	3	120	219	0	0	158	0	1.6	2	0	3	0	
28	58	0	3	120	340	0	0	172	0	0	1	0	3	0	
29	66	0	1	150	226	0	0	114	0	2.6	3	0	3	0	
30	43	1	4	150	247	0	0	171	0	1.5	1	0	3	0	
31	40	1	4	110	167	0	2	114	1	2	2	0	7	3	
32	69	0	1	140	239	0	0	151	0	1.8	1	2	3	0	
33	60	1	4	117	230	1	0	160	1	1.4	1	2	7	2	
34	64	1	3	140	335	0	0	158	0	0	1	0	3	1	
35	59	1	4	135	234	0	0	161	0	0.5	2	0	7	0	
36	44	1	3	130	233	0	0	179	1	0.4	1	0	3	0	
37	42	1	4	140	226	0	0	178	0	0	1	0	3	0	

Output:

```
(venv) dhanrz@dhanrz-G7-7588:~/LAB/6SEM/ML_1BM18CS027/lab05$ python src.py
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope ca thal  heartdisease
0   63    1   1       145   233    1        2      150      0      2.3      3  0    6             0
1   67    1   4       160   286    0        2      108      1      1.5      2  3    3             2
2   67    1   4       120   229    0        2      129      1      2.6      2  2    7             1
3   37    1   3       130   250    0        0      187      0      3.5      3  0    3             0
4   41    0   2       130   204    0        2      172      0      1.4      1  0    3             0

 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

  Inferencing with Bayesian Network:

  1. Probability of HeartDisease given evidence= restecg
 Finding Elimination Order: : 100%|| 5/5 [00:00<00:00, 2144.11it/s]
 Eliminating: sex: 100%|| 5/5 [00:00<00:00, 120.80it/s]
 +-----------------+---------------------+             | 0/5 [00:00<?, ?it/s]
 | 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, 2715.46it/s]
 Eliminating: sex: 100%|| 5/5 [00:00<00:00, 497.58it/s]
 +-----------------+---------------------+             | 0/5 [00:00<?, ?it/s]
 | heartdisease    |   phi(heartdisease) |
 +=================+=====================+
 | heartdisease(0) |              0.3610 |
 +-----------------+---------------------+
 | heartdisease(1) |              0.2159 |
 +-----------------+---------------------+
 | heartdisease(2) |              0.1373 |
 +-----------------+---------------------+
 | heartdisease(3) |              0.1537 |
 +-----------------+---------------------+
 | heartdisease(4) |              0.1321 |
 +-----------------+---------------------+
 (venv) dhanrz@dhanrz-G7-7588:~/LAB/6SEM/ML_1BM18CS027/lab05$
```

Program 6:

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

Program:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a vers
ion using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

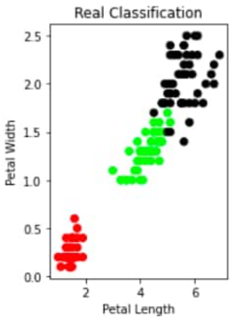
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
```

```
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'])
```

```
<Figure size 1008x504 with 0 Axes>
```

```
# 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')
```

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



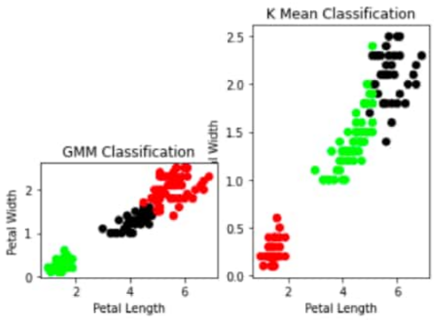

```
# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The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

Dataset:

	A	B	C	D	E	F
1	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
2	1	5.1	3.5	1.4	0.2	Iris-setosa
3	2	4.9	3	1.4	0.2	Iris-setosa
4	3	4.7	3.2	1.3	0.2	Iris-setosa
5	4	4.6	3.1	1.5	0.2	Iris-setosa
6	5	5	3.6	1.4	0.2	Iris-setosa
7	6	5.4	3.9	1.7	0.4	Iris-setosa
8	7	4.6	3.4	1.4	0.3	Iris-setosa
9	8	5	3.4	1.5	0.2	Iris-setosa
10	9	4.4	2.9	1.4	0.2	Iris-setosa
11	10	4.9	3.1	1.5	0.1	Iris-setosa
12	11	5.4	3.7	1.5	0.2	Iris-setosa
13	12	4.8	3.4	1.6	0.2	Iris-setosa
14	13	4.8	3	1.4	0.1	Iris-setosa
15	14	4.3	3	1.1	0.1	Iris-setosa
16	15	5.8	4	1.2	0.2	Iris-setosa
17	16	5.7	4.4	1.5	0.4	Iris-setosa
18	17	5.4	3.9	1.3	0.4	Iris-setosa
19	18	5.1	3.5	1.4	0.3	Iris-setosa
20	19	5.7	3.8	1.7	0.3	Iris-setosa
21	20	5.1	3.8	1.5	0.3	Iris-setosa
22	21	5.4	3.4	1.7	0.2	Iris-setosa
23	22	5.1	3.7	1.5	0.4	Iris-setosa
24	23	4.6	3.6	1	0.2	Iris-setosa
25	24	5.1	3.3	1.7	0.5	Iris-setosa
26	25	4.8	3.4	1.9	0.2	Iris-setosa
27	26	5	3	1.6	0.2	Iris-setosa
28	27	5	3.4	1.6	0.4	Iris-setosa
29	28	5.2	3.5	1.5	0.2	Iris-setosa
30	29	5.2	3.4	1.4	0.2	Iris-setosa
31	30	4.7	3.2	1.6	0.2	Iris-setosa
32	31	4.8	3.1	1.6	0.2	Iris-setosa
33	32	5.4	3.4	1.5	0.4	Iris-setosa
34	33	5.2	4.1	1.5	0.1	Iris-setosa
35	34	5.5	4.2	1.4	0.2	Iris-setosa

Output:

```
The accuracy score of K-Mean:  0.8933333333333333
The Confusion matrixof K-Mean:  [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
The accuracy score of EM:  0.0
The Confusion matrix of EM:  [[ 0 50  0]
 [ 5  0 45]
 [50  0  0]]
```



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.

Program:

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

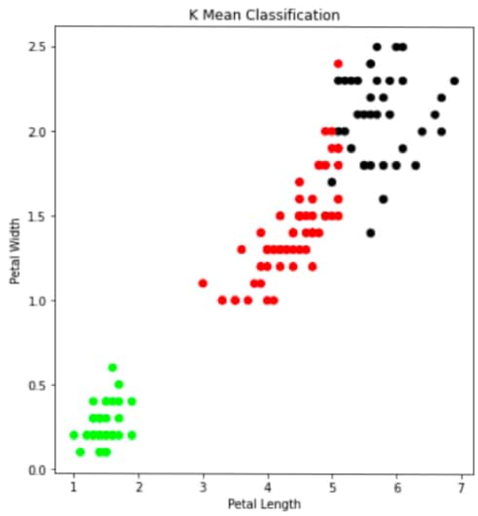
```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

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))
```

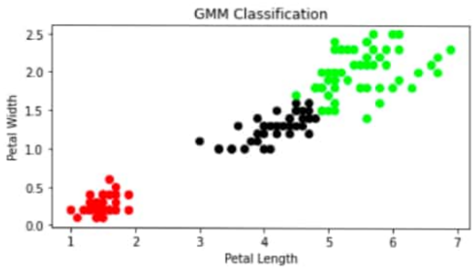
Dataset:

	A	B	C	D	E	F
1	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
2	1	5.1	3.5	1.4	0.2	Iris-setosa
3	2	4.9	3	1.4	0.2	Iris-setosa
4	3	4.7	3.2	1.3	0.2	Iris-setosa
5	4	4.6	3.1	1.5	0.2	Iris-setosa
6	5	5	3.6	1.4	0.2	Iris-setosa
7	6	5.4	3.9	1.7	0.4	Iris-setosa
8	7	4.6	3.4	1.4	0.3	Iris-setosa
9	8	5	3.4	1.5	0.2	Iris-setosa
10	9	4.4	2.9	1.4	0.2	Iris-setosa
11	10	4.9	3.1	1.5	0.1	Iris-setosa
12	11	5.4	3.7	1.5	0.2	Iris-setosa
13	12	4.8	3.4	1.6	0.2	Iris-setosa
14	13	4.8	3	1.4	0.1	Iris-setosa
15	14	4.3	3	1.1	0.1	Iris-setosa
16	15	5.8	4	1.2	0.2	Iris-setosa
17	16	5.7	4.4	1.5	0.4	Iris-setosa
18	17	5.4	3.9	1.3	0.4	Iris-setosa
19	18	5.1	3.5	1.4	0.3	Iris-setosa
20	19	5.7	3.8	1.7	0.3	Iris-setosa
21	20	5.1	3.8	1.5	0.3	Iris-setosa
22	21	5.4	3.4	1.7	0.2	Iris-setosa
23	22	5.1	3.7	1.5	0.4	Iris-setosa
24	23	4.6	3.6	1	0.2	Iris-setosa
25	24	5.1	3.3	1.7	0.5	Iris-setosa
26	25	4.8	3.4	1.9	0.2	Iris-setosa
27	26	5	3	1.6	0.2	Iris-setosa
28	27	5	3.4	1.6	0.4	Iris-setosa

Output:

```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
The accuracy score of EM:  0.36666666666666664
The Confusion matrix of EM:  [[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
```





Program 8:

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

Program:

```python
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("iris.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 ("-------------------------------------------------------------------------")
```

Dataset:

	A	B	C	D	E	F
1	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
2	1	5.1	3.5	1.4	0.2	Iris-setosa
3	2	4.9	3	1.4	0.2	Iris-setosa
4	3	4.7	3.2	1.3	0.2	Iris-setosa
5	4	4.6	3.1	1.5	0.2	Iris-setosa
6	5	5	3.6	1.4	0.2	Iris-setosa
7	6	5.4	3.9	1.7	0.4	Iris-setosa
8	7	4.6	3.4	1.4	0.3	Iris-setosa
9	8	5	3.4	1.5	0.2	Iris-setosa
10	9	4.4	2.9	1.4	0.2	Iris-setosa
11	10	4.9	3.1	1.5	0.1	Iris-setosa
12	11	5.4	3.7	1.5	0.2	Iris-setosa
13	12	4.8	3.4	1.6	0.2	Iris-setosa
14	13	4.8	3	1.4	0.1	Iris-setosa
15	14	4.3	3	1.1	0.1	Iris-setosa
16	15	5.8	4	1.2	0.2	Iris-setosa
17	16	5.7	4.4	1.5	0.4	Iris-setosa
18	17	5.4	3.9	1.3	0.4	Iris-setosa
19	18	5.1	3.5	1.4	0.3	Iris-setosa
20	19	5.7	3.8	1.7	0.3	Iris-setosa
21	20	5.1	3.8	1.5	0.3	Iris-setosa
22	21	5.4	3.4	1.7	0.2	Iris-setosa
23	22	5.1	3.7	1.5	0.4	Iris-setosa
24	23	4.6	3.6	1	0.2	Iris-setosa
25	24	5.1	3.3	1.7	0.5	Iris-setosa
26	25	4.8	3.4	1.9	0.2	Iris-setosa
27	26	5	3	1.6	0.2	Iris-setosa
28	27	5	3.4	1.6	0.4	Iris-setosa

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-setosa                 Iris-setosa                 Correct
Iris-setosa                 Iris-setosa                 Correct
Iris-setosa                 Iris-setosa                 Correct
Iris-virginica              Iris-virginica              Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-virginica              Iris-virginica              Correct
Iris-virginica              Iris-virginica              Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-setosa                 Iris-setosa                 Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-virginica              Iris-virginica              Correct
------------------------------------------------------------------------

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


Classification Report:
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         4
Iris-versicolor       1.00      1.00      1.00         7
 Iris-virginica       1.00      1.00      1.00         4

       accuracy                           1.00        15
      macro avg       1.00      1.00      1.00        15
   weighted avg       1.00      1.00      1.00        15

------------------------------------------------------------------------
Accuracy of the classifer is 1.00
------------------------------------------------------------------------
```

Program 9:

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

Program:

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

# Importing the dataset
#dataset = pd.read_csv('181105_missing-data.csv')
dataset = pd.read_csv('salary_data.csv')
X = dataset.iloc[:, :-1].values #get a copy of dataset exclude last column
y = dataset.iloc[:, 1].values #get array of dataset in column 1st

# Splitting the dataset into the Training set and Test set
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='blue')
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='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()
```

Dataset:

	A	B	C	D
1	YearsExp	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		
26	8.7	109431		
27	9	105582		
28	9.5	116969		
29	9.6	112635		
30	10.3	122391		
31	10.5	121872		

Output:


Salary VS Experience (Training set)
Salary
Year of Experience
120000
100000
80000
60000
40000
2
4
6
8
10


Salary VS Experience (Test set)
Salary
Year of Experience
120000
100000
80000
60000
40000
2
4
6
8
10

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.

Program:

```
import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook

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)]]))
```

Dataset:

```
The Data Set ( 10 Samples) X :
           0
0 -2.993994
1 -2.987988
2 -2.981982
3 -2.975976
4 -2.969970
5 -2.963964
6 -2.957958
7 -2.951952
8 -2.945946
```

Output:

```
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 :
 [-2.72466233 -3.00545859 -3.04154251 -2.87617738 -2.83877412 -3.00294342
 -3.00884493 -2.91189305 -2.8651739 ]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```