AUDISANKARA COLLEGE
OF
ENGINEERING & TECHNOLOGY

NH5, BYPASS ROAD, GUDUR.

(AUTONOMOUS)

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING
LAB MANUAL

OF

Machine Learning Lab
20CS611

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

FIND-S Algorithm

1. Initialize h to the most specific hypothesis in H

2. For each positive training instance x

    For each attribute constraint a_i in h

        If the constraint a_i is satisfied by x

        Then do nothing

        Else replace a_i in h by the next more general constraint that is satisfied by x

3. Output hypothesis h

Training Examples:

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

```
import csv

num_attributes = 6
a = []
print("\n The Given Training Data Set \n")

with open('enjoysport.csv', 'r') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        a.append (row)
        print(row)

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


for j in range(0,num_attributes):
        hypothesis[j] = a[0][j];


print("\n Find S: Finding a Maximally Specific Hypothesis\n")

for i in range(0,len(a)):
    if a[i][num_attributes]=='yes':
            for j in range(0,num_attributes):
                if a[i][j]!=hypothesis[j]:
                    hypothesis[j]='?'
                else :
                    hypothesis[j]= a[i][j]
    print(" For Training instance No:{0} the hypothesis is
".format(i),hypothesis)

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

Data Set:

sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Output:

```
The Given Training Data Set

['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']

Find S: Finding a Maximally Specific Hypothesis

For Training Example No:0 the hypothesis is
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

For Training Example No:1 the hypothesis is
['sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training Example No:2 the hypothesis is
'sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training Example No:3 the hypothesis is
'sunny', 'warm', '?', 'strong', '?', '?']

The Maximally Specific Hypothesis for a given Training Examples:
```

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

ML 2 - CANDIDATE-ELIMINATION ALGORITHM

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.

SOLUTION 1

trainingdata.csv

Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

prog2.py

```
import csv

with open("trainingdata.csv") as f:
    csv_file=csv.reader(f)
    data=list(csv_file)

    s=data[1][:-1]
    g=[['?' for i in range(len(s))] for j in range(len(s))]

    for i in data:
        if i[-1]=="Yes":
            for j in range(len(s)):
                if i[j]!=s[j]:
                    s[j]='?'
                    g[j][j]='?'

        elif i[-1]=="No":
            for j in range(len(s)):
                if i[j]!=s[j]:
                    g[j][j]=s[j]
                else:
                    g[j][j]="?"
        print("\nSteps of Candidate Elimination Algorithm",data.index(i)+1)
        print(s)
        print(g)
    gh=[]
    for i in g:
        for j in i:
            if j!='?':
                gh.append(i)
                break
    print("\nFinal specific hypothesis:\n",s)

    print("\nFinal general hypothesis:\n",gh)
```

Output

```
Steps of Candidate Elimination Algorithm 1
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

Steps of Candidate Elimination Algorithm 3
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Steps of Candidate Elimination Algorithm 4
['Sunny', 'Warm', '?', 'Strong', '?', '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final specific hypothesis:
['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final general hypothesis:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

Exp. No. 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:

PlayTennis Dataset is saved as .csv (comma separated values) file in the current working directory otherwise use the complete path of the dataset set in the program:

Day	***Outlook***	***Temperature***	***Humidity***	***Wind***	***PlayTennis***
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes

D14	Rain	Mild	High	Strong	No

```
import pandas as pd

import math

import numpy as np

data = pd.read_csv("3-dataset.csv")

features = [feat for feat in data]

features.remove("answer")

class Node:

    def __init__(self):

        self.children = []

        self.value = ""

        self.isLeaf = False

        self.pred = ""

def 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])

    #print ("\n",uniq)

    gain = entropy(examples)

    #print ("\n",gain)

    for u in uniq:

        subdata = examples[examples[attr] == u]

        #print ("\n",subdata)

        sub_e = entropy(subdata)

        gain -= (float(len(subdata)) / float(len(examples))) *
sub_e

        #print ("\n",gain)
```

```
    return gain


def ID3(examples, attrs):
    root = Node()


    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
```

```
        #print ("\n",subdata)

        if entropy(subdata) == 0.0:

            newNode = Node()

            newNode.isLeaf = True

            newNode.value = u

            newNode.pred = np.unique(subdata["answer"])

            root.children.append(newNode)

        else:

            dummyNode = Node()

            dummyNode.value = u

            new_attrs = attrs.copy()

            new_attrs.remove(max_feat)

            child = ID3(subdata, new_attrs)

            dummyNode.children.append(child)

            root.children.append(dummyNode)

    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)

printTree(root)
```

```
 Outlook
  rain
       Wind
              strong
                    no
              weak
                   yes
   overcast
        yes

  sunny
       Humidity
              normal
                   yes
              high
                   no
```

Exp. No. 4. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Python Program to Implement and Demonstrate Backpropagation Algorithm Machine Learning

Training Examples:

Example	Sleep	Study	Expected % in Exams
1	2	9	92
2	1	5	86
3	3	6	89

Normalize the input

Example	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

```
import numpy as np
```

```
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) #maximum of X array longitudinally

y = y/100


#Sigmoid Function

def sigmoid (x):

    return 1/(1 + np.exp(-x))


#Derivative of Sigmoid Function

def derivatives_sigmoid(x):

    return x * (1 - x)


#Variable initialization

epoch=5 #Setting training iterations

lr=0.1 #Setting learning rate


inputlayer_neurons = 2 #number of features in data set

hiddenlayer_neurons = 3 #number of hidden layers neurons

output_neurons = 1 #number of neurons at output layer
```

```
#weight and bias initialization



wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neuron
s))

bh=np.random.uniform(size=(1,hiddenlayer_neurons))

wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons)
)

bout=np.random.uniform(size=(1,output_neurons))



#draws a random range of numbers uniformly of dim x*y

for i in range(epoch):

    #Forward Propogation

    hinp1=np.dot(X,wh)

    hinp=hinp1 + bh

    hlayer_act = sigmoid(hinp)

    outinp1=np.dot(hlayer_act,wout)

    outinp= outinp1+bout

    output = sigmoid(outinp)



    #Backpropagation

    EO = y-output
```

```
    outgrad = derivatives_sigmoid(output)

    d_output = EO * outgrad

    EH = d_output.dot(wout.T)

    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden
layer wts contributed to error

    d_hiddenlayer = EH * hiddengrad



    wout += hlayer_act.T.dot(d_output) *lr   # dotproduct of
nextlayererror and currentlayerop

    wh += X.T.dot(d_hiddenlayer) *lr



    print ("-----------Epoch-", i+1, "Starts----------")

    print("Input: \n" + str(X))

    print("Actual Output: \n" + str(y))

    print("Predicted Output: \n" ,output)

    print ("-----------Epoch-", i+1, "Ends----------\n")



print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)
```

Output

————Epoch- 1 Starts———-
Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.81951208]
[0.8007242]
[0.82485744]]
————Epoch- 1 Ends———-

Epoch- 2 Starts———-
Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.82033938]
[0.80153634]
[0.82568134]]
————Epoch- 2 Ends———-

————Epoch- 3 Starts———-
Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:

[[0.82115226]
[0.80233463]
[0.82649072]]
————Epoch- 3 Ends————-

Epoch- 4 Starts————-
Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.82195108]
[0.80311943]
[0.82728598]]
————Epoch- 4 Ends————-

————Epoch- 5 Starts————-
Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.8227362]
[0.80389106]
[0.82806747]]
————Epoch- 5 Ends————-

Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]
[0.89]]

Predicted Output:
[[0.8227362]
[0.80389106]
[0.82806747]]

6. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

<u>Naive Bayes algorithms for learning and classifying text</u>

LEARN_NAIVE_BAYES_TEXT (Examples, V)

Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms $P(w_k |v_j,)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

1. *collect all words, punctuation, and other tokens that occur in Examples*
   - *Vocabulary* ← ***c*** the set of all distinct words and other tokens occurring in any text document from *Examples*
2. *calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms*
   - For each target value v_j in V do
     - $docs_j$ ← the subset of documents from *Examples* for which the target value is *vj*
     - $P(v_j)$ ← $| docs_j | / |\text{Examples}|$
     - $Text_j$ ← a single document created by concatenating all members of $docs_j$
     - n ← total number of distinct word positions in $Text_j$
     - for each word w_k in *Vocabulary*
       - n_k ← number of times word $\boldsymbol{w_k}$ occurs in $Text_j$
       - $P(w_k|v_j)$ ← $(n_k + 1) / (n + | Vocabulary|)$

CLASSIFY_NAIVE_BAYES_TEXT (Doc)

Return the estimated target value for the document Doc. a_i denotes the word found in the i^{th} position within Doc.

- *positions* ← all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return V_{NB}, where

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i | v_j)$$

Data set:

	Text Documents	Label
1	I love this sandwich	pos
2	This is an amazing place	pos
3	I feel very good about these beers	pos
4	This is my best work	pos
5	What an awesome view	pos
6	I do not like this restaurant	neg
7	I am tired of this stuff	neg
8	I can't deal with this	neg
9	He is my sworn enemy	neg
10	My boss is horrible	neg
11	This is an awesome place	pos
12	I do not like the taste of this juice	neg
13	I love to dance	pos
14	I am sick and tired of this place	neg
15	What a great holiday	pos
16	That is a bad locality to stay	neg
17	We will have good fun tomorrow	pos
18	I went to my enemy's house today	neg

Program:

```
import pandas as pd

msg=pd.read_csv('naivetext.csv',names=['message','label'])

print('The dimensions of the dataset',msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum

print(X)
print(y)

#splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)

print ('\n The total number of Training Data :',ytrain.shape)
print ('\n The total number of Test Data :',ytest.shape)


#output of count vectoriser is a sparse matrix
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(count_vect.get_feature_names())

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_fe
ature_names())

# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

#printing accuracy, Confusion matrix, Precision and Recall
from sklearn import metrics
print('\n Accuracy of the classifer is',
metrics.accuracy_score(ytest,predicted))
```

```
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))


print('\n The value of Precision' ,
metrics.precision_score(ytest,predicted))


print('\n The value of Recall' ,
metrics.recall_score(ytest,predicted))
```

Output:

The dimensions of the dataset (18, 2)

0	I love this sandwich
1	This is an amazing place
2	I feel very good about these beers
3	This is my best work
4	What an awesome view
5	I do not like this restaurant
6	I am tired of this stuff
7	I can't deal with this
8	He is my sworn enemy
9	My boss is horrible
10	This is an awesome place
11	I do not like the taste of this juice
12	I love to dance
13	I am sick and tired of this place
14	What a great holiday
15	That is a bad locality to stay
16	We will have good fun tomorrow
17	I went to my enemy's house today

```
Name: message, dtype: object
0    1
1    1
2    1
3    1
4    1
5    0
6    0
7    0
8    0
9    0
10    1
11    0
12    1
13    0
14    1
15    0
16    1
17    0
Name: labelnum, dtype: int64
```

The total number of Training Data: (13,)

The total number of Test Data: (5,)

The words or Tokens in the text documents

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'can', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'great', 'have', 'he', 'holiday', 'house', 'is', 'like', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sandwich', 'sick', 'sworn', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will', 'with', 'work']

Accuracy of the classifier is 0.8

Confusion matrix

[[2 1]

 [0 2]]

The value of Precision 0.6666666666666666

The value of Recall 1.0

Basic knowledge

Confusion Matrix

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

True positives: data points labelled as positive that are actually positive

False positives: data points labelled as positive that are actually negative

True negatives: data points labelled as negative that are actually negative

False negatives: data points labelled as negative that are actually positive

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Example:

		Actual	
		Positive	Negative
Predicted	Positive	1 TP	3 FP
	Negative	0 FN	1 TN

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{1}{1+3} = 0.25$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{1}{1+0} = 1$$

Accuracy: how often is the classifier correct?

$$\text{Accuracy} = \frac{\text{TP + TN}}{\text{Total}} = \frac{1+1}{5} = 0.4$$

Example: Movie Review

Doc	Text	Class
1	I loved the movie	+
2	I hated the movie	-
3	a great movie. good movie	+
4	poor acting	-
5	great acting. good movie	+

Unique word

< I, loved, the, movie, hated, a, great, good, poor, acting>

Doc	I	loved	the	movie	hated	a	great	good	poor	acting	Class
1	1	1	1	1							+
2	1		1	1	1						-
3				2		1	1	1			+
4									1	1	-
5				1			1	1		1	+

Doc	I	loved	the	movie	hated	a	great	good	poor	acting	Class
1	1	1	1	1							+
3				2		1	1	1			+
5				1			1	1		1	+

$$P(+) = \frac{3}{5} = 0.6$$

$$P(I\ |+) = \frac{1+1}{14+10} = 0.0833$$

$$P(a\ |+) = \frac{1+1}{14+10} = 0.0833$$

$$P(loved\ |+) = \frac{1+1}{14+10} = 0.0833$$

$$P(great\ |+) = \frac{2+1}{14+10} = 0.125$$

$$P(the\ |+) = \frac{1+1}{14+10} = 0.0833$$

$$P(good\ |+) = \frac{2+1}{14+10} = 0.125$$

$$P(movie\ |+) = \frac{4+1}{14+10} = 0.2083$$

$$P(poor\ |+) = \frac{0+1}{14+10} = 0.0416$$

$$P(hated\ |+) = \frac{0+1}{14+10} = 0.0416$$

$$P(acting\ |+) = \frac{1+1}{14+10} = 0.0833$$

Doc	I	loved	the	movie	hated	a	great	good	poor	acting	Class
2	1		1	1	1						-
4									1	1	-

$$P(-) = \frac{2}{5} = 0.4$$

$$P(I\ |-) = \frac{1+1}{6+10} = 0.125$$

$$P(a\ |-) = \frac{0+1}{6+10} = 0.0625$$

$$P(loved\ |-) = \frac{0+1}{6+10} = 0.0625$$

$$P(great\ |-) = \frac{0+1}{6+10} = 0.0625$$

$$P(the\ |-) = \frac{1+1}{6+10} = 0.125$$

$$P(good\ |-) = \frac{0+1}{6+10} = 0.0625$$

$$P(movie|-) = \frac{1+1}{6+10} = 0.125$$

$$P(poor|-) = \frac{1+1}{6+10} = 0.125$$

$$P(hated\ |-) = \frac{1+1}{6+10} = 0.125$$

$$P(acting|-) = \frac{1+1}{6+10} = 0.125$$

Let's classify the new document

I hated the poor acting

If $V_j = +$

then,

$= P(+)\ P(I\mid +)\ P(hated\mid +)\ P(the\mid +)\ P(poor\mid +)\ P(acting\mid +)$

= 0.6 * 0.0833 * 0.0416 * 0.0833 * 0.0416 * 0.0833

= $6.03 \text{ X } 10^{-2}$

If $V_j = -$

then,

$= P(-)\ P(I\mid -)\ P(hated\mid -)\ P(the\mid -)\ P(poor\mid -)\ P(acting\mid -)$

= 0.4 * 0.125 * 0.125 * 0.125 * 0.125 * 0.125

= $1.22 \text{ X } 10^{-5}$

= $1.22 \text{ X } 10^{-5} > 6.03 \text{ X } 10^{-2}$

So, the new document belongs to $(-)$ class

7. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API

Theory

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable.

Bayesian network consists of two major parts: a directed acyclic graph and a set of conditional probability distributions

- The directed acyclic graph is a set of random variables represented by nodes.
- The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).

For illustration, consider the following example. Suppose we attempt to turn on our computer, but the computer does not start (observation/evidence). We would like to know which of the possible causes of computer failure is more likely. In this simplified illustration, we assume only two possible causes of this misfortune: electricity failure and computer malfunction.
The corresponding directed acyclic graph is depicted in below figure.

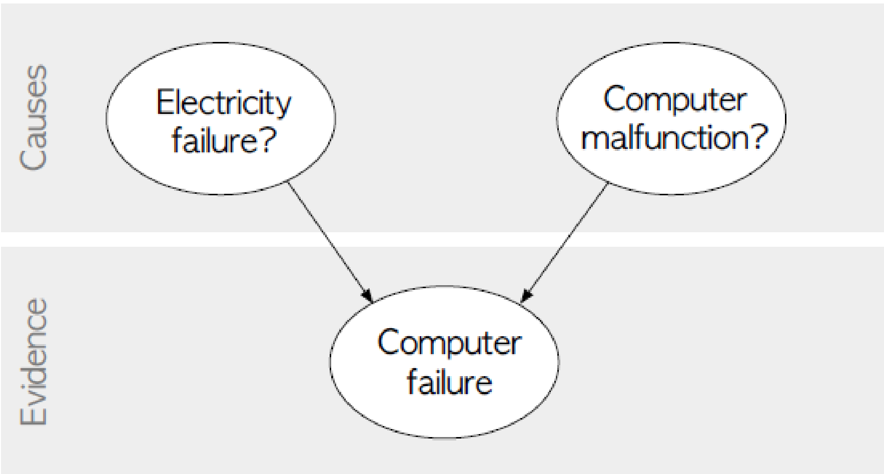


Fig: Directed acyclic graph representing two independent possible causes of a computer failure.

The goal is to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. P [Cause | Evidence].

Data Set:

Title: Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Database:	0	1	2	3	4	Total
Cleveland:	164	55	36	35	13	303

Attribute Information:

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
    - Value 1: typical angina
    - Value 2: atypical angina
    - Value 3: non-anginal pain
    - Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
    - Value 0: normal
    - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
13. Heartdisease: It is integer valued from 0 (no presence) to 4.

Some instance from the dataset:

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
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Program:

```
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'),(
'exang','heartdisease'),('cp','heartdisease'),('heartdisease',
'restecg'),('heartdisease','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'],evi
dence={'restecg':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'],evi
dence={'cp':2})
print(q2)
```

Output:

```
================ RESTART: E:\ML Lab - 2020-21\MLLab-7\ML7.py ================
Few examples 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

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg

    +-----------------+---------------------+
    | heartdisease    |   phi(heartdisease) |
    +=================+=====================+
    | heartdisease(0) |              0.1012 |
    +-----------------+---------------------+
    | heartdisease(1) |              0.0000 |
    +-----------------+---------------------+
    | heartdisease(2) |              0.2392 |
    +-----------------+---------------------+
    | heartdisease(3) |              0.2015 |
    +-----------------+---------------------+
    | heartdisease(4) |              0.4581 |
    +-----------------+---------------------+
```

```
2. Probability of HeartDisease given evidence= cp
+-----------------+----------------------+
| heartdisease    |   phi(heartdisease)  |
+=================+======================+
| heartdisease(0) |               0.3610 |
+-----------------+----------------------+
| heartdisease(1) |               0.2159 |
+-----------------+----------------------+
| heartdisease(2) |               0.1373 |
+-----------------+----------------------+
| heartdisease(3) |               0.1537 |
+-----------------+----------------------+
| heartdisease(4) |               0.1321 |
+-----------------+----------------------+
```

```
8) Apply EM algorithm to cluster a set of data stored in a .CSV file.
Use the same data


DATA SET
```

5.7	2.8	4.1	1.3	Iris-versicolor
6.3	3.3	6	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3	5.9	2.1	Iris-virginica
6.3	2.9	5.6	1.8	Iris-virginica
6.5	3	5.8	2.2	Iris-virginica
7.6	3	6.6	2.1	Iris-virginica
4.9	2.5	4.5	1.7	Iris-virginica
7.3	2.9	6.3	1.8	Iris-virginica
6.7	2.5	5.8	1.8	Iris-virginica
7.2	3.6	6.1	2.5	Iris-virginica
6.5	3.2	5.1	2	Iris-virginica
6.4	2.7	5.3	1.9	Iris-virginica
6.8	3	5.5	2.1	Iris-virginica
5.7	2.5	5	2	Iris-virginica
5.8	2.8	5.1	2.4	Iris-virginica
6.4	3.2	5.3	2.3	Iris-virginica
6.5	3	5.5	1.8	Iris-virginica
7.7	3.8	6.7	2.2	Iris-virginica
7.7	2.6	6.9	2.3	Iris-virginica
6	2.2	5	1.5	Iris-virginica
6.9	3.2	5.7	2.3	Iris-virginica
5.6	2.8	4.9	2	Iris-virginica
7.7	2.8	6.7	2	Iris-virginica
6.3	2.7	4.9	1.8	Iris-virginica
6.7	3.3	5.7	2.1	Iris-virginica
7.2	3.2	6	1.8	Iris-virginica
6.2	2.8	4.8	1.8	Iris-virginica
6.1	3	4.9	1.8	Iris-virginica
6.4	2.8	5.6	2.1	Iris-virginica
7.2	3	5.8	1.6	Iris-virginica
7.4	2.8	6.1	1.9	Iris-virginica
7.9	3.8	6.4	2	Iris-virginica
6.4	2.8	5.6	2.2	Iris-virginica
6.3	2.8	5.1	1.5	Iris-virginica
6.1	2.6	5.6	1.4	Iris-virginica
7.7	3	6.1	2.3	Iris-virginica
6.3	3.4	5.6	2.4	Iris-virginica
6.4	3.1	5.5	1.8	Iris-virginica

6	3	4.8	1.8	Iris-virginica
6.9	3.1	5.4	2.1	Iris-virginica
6.7	3.1	5.6	2.4	Iris-virginica
6.9	3.1	5.1	2.3	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
6.8	3.2	5.9	2.3	Iris-virginica
6.7	3.3	5.7	2.5	Iris-virginica
6.7	3	5.2	2.3	Iris-virginica
6.3	2.5	5	1.9	Iris-virginica
6.5	3	5.2	2	Iris-virginica
6.2	3.4	5.4	2.3	Iris-virginica
5.9	3	5.1	1.8	Iris-virginica

```
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("8-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(['red','lime','black'])

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

# K-PLOT
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 PLOT
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_cluster_gmm))
```

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]]

9. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

K-Nearest Neighbor Algorithm

Training algorithm:

- For each training example (x, f (x)), add the example to the list training examples

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

- Where, $f(x_i)$ function to calculate the mean value of the k nearest training examples.

Data Set:

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes)
Number of Attributes: 4 numeric, predictive attributes and the Class

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Program:

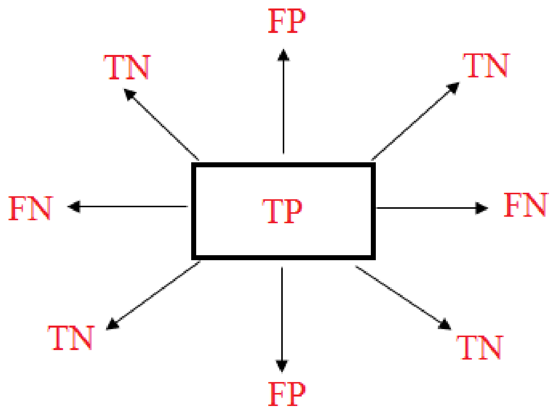
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

""" Iris Plants Dataset, dataset contains 150 (50 in each of three
classes)Number of Attributes: 4 numeric, predictive attributes and
the Class
"""
iris=datasets.load_iris()

""" The x variable contains the first four columns of the dataset
(i.e. attributes) while y contains the labels.
"""
x = iris.data
y = iris.target

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

""" Splits the dataset into 70% train data and 30% test data. This
means that out of total 150 records, the training set will contain
105 records and the test set contains 45 of those records
"""
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.3)

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

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

""" For evaluating an algorithm, confusion matrix, precision, recall
and f1 score are the most commonly used metrics.
"""
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

Output:

```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 .   .    .    .    .
 .   .    .    .    .

 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]

class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 ………0 0 1 1 1 …………1 1 2 2 2 ………… 2 2]

Confusion Matrix
[[20  0  0]
 [ 0 10  0]
 [ 0  1 14]]

Accuracy Metrics

                Precision  recall   f1-score   support

          0       1.00      1.00      1.00       20
          1       0.91      1.00      0.95       10
          2       1.00      0.93      0.97       15

avg / total       0.98      0.98      0.98       45
```

Basic knowledge

Confusion Matrix



True positives: data points labelled as positive that are actually positive
False positives: data points labelled as positive that are actually negative
True negatives: data points labelled as negative that are actually negative
False negatives: data points labelled as negative that are actually positive

$$\text{Recall} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Negative}}$$

$$= \frac{\textit{True Positive}}{\textit{Total Actual Positive}}$$

$$\text{Precision} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Positive}}$$

$$= \frac{\textit{True Positive}}{\textit{Total Predicted Positive}}$$

Accuracy: how often is the classifier correct?

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}}$$

F1-Score:

$$\text{F1 Score} = \frac{2.\text{TP}}{2.\text{TP} + \text{FP} + \text{FN}}$$

Support: Total Predicted of Class.

$$\text{Support} = \text{TP} + \text{FN}$$

Example:

	GoldLabel_A	GoldLabel_B	GoldLabel_C	
Predicted_A	**30**	20	10	TotalPredicted_A=60
Predicted_B	50	**60**	10	TotalPredicted_B=120
Predicted_C	20	20	**80**	TotalPredicted_C=120
	TotalGoldLabel_A=100	TotalGoldLabel_B=100	TotalGoldLabel_C=100	

This is an example confusion matrix for 3 labels: A,B and C

- Now, let us compute **recall** for Label A:

```
= TP_A/(TP_A+FN_A)
= TP_A/(Total Gold for A)
= TP_A/TotalGoldLabel_A
= 30/100
= 0.3
```

- Now, let us compute **precision** for Label A:

```
= TP_A/(TP_A+FP_A)
= TP_A/(Total predicted as A)
= TP_A/TotalPredicted_A
= 30/60
= 0.5
```

- Now, let us compute **F1-score** for Label A:

$$\text{F1 Score} = \frac{2.\text{TP}}{2.\text{TP} + \text{FP} + \text{FN}}$$

= 2*30 / (2*30 + 60 + 100)

= 0.27

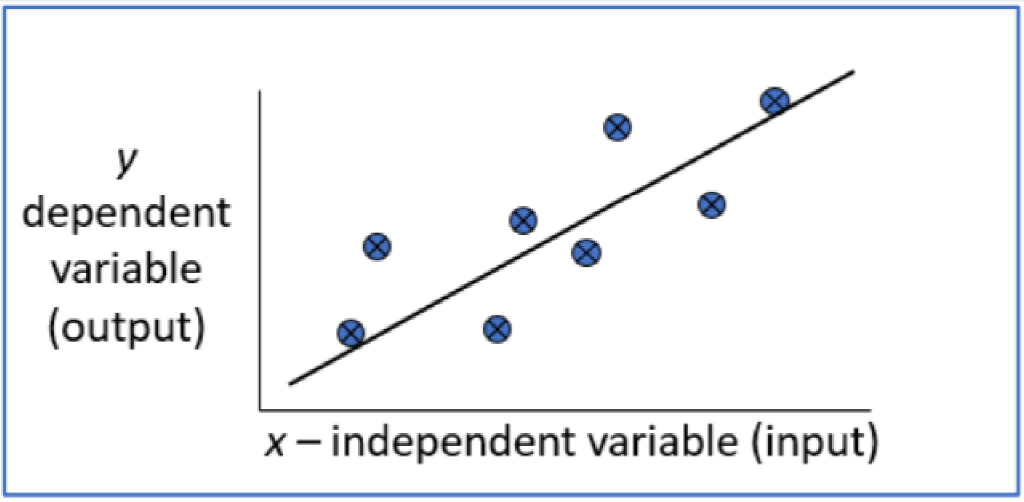
- Support _ A = TP_A + FN_A
  = 30 + (20 + 10)
  = 60

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.

<u>Locally Weighted Regression Algorithm</u>

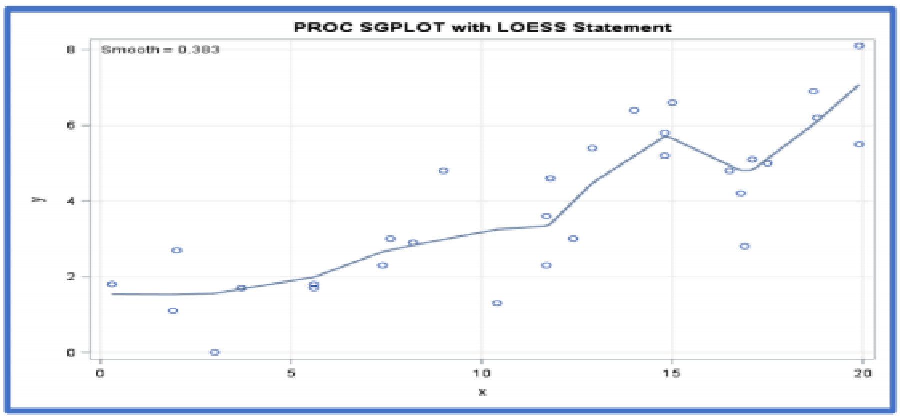
Regression:

- Regression is a technique from statistics that is used to predict values of a desired target quantity when the target quantity is continuous.
- In regression, we seek to identify (or estimate) a continuous variable y associated with a given input vector x.
    - y is called the dependent variable.
    - x is called the independent variable.



Loess/Lowess Regression:

Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot.



Lowess Algorithm:

- Locally weighted regression is a very powerful nonparametric model used in statistical learning.
- Given a dataset X, y, we attempt to find a model parameter β(x) that minimizes residual sum of weighted squared errors.
- The weights are given by a kernel function (k or w) which can be chosen arbitrarily

Algorithm

1. Read the Given data Sample to X and the curve (linear or non linear) to Y
2. Set the value for Smoothening parameter or Free parameter say τ
3. Set the bias /Point of interest set x0 which is a subset of X
4. Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using :

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

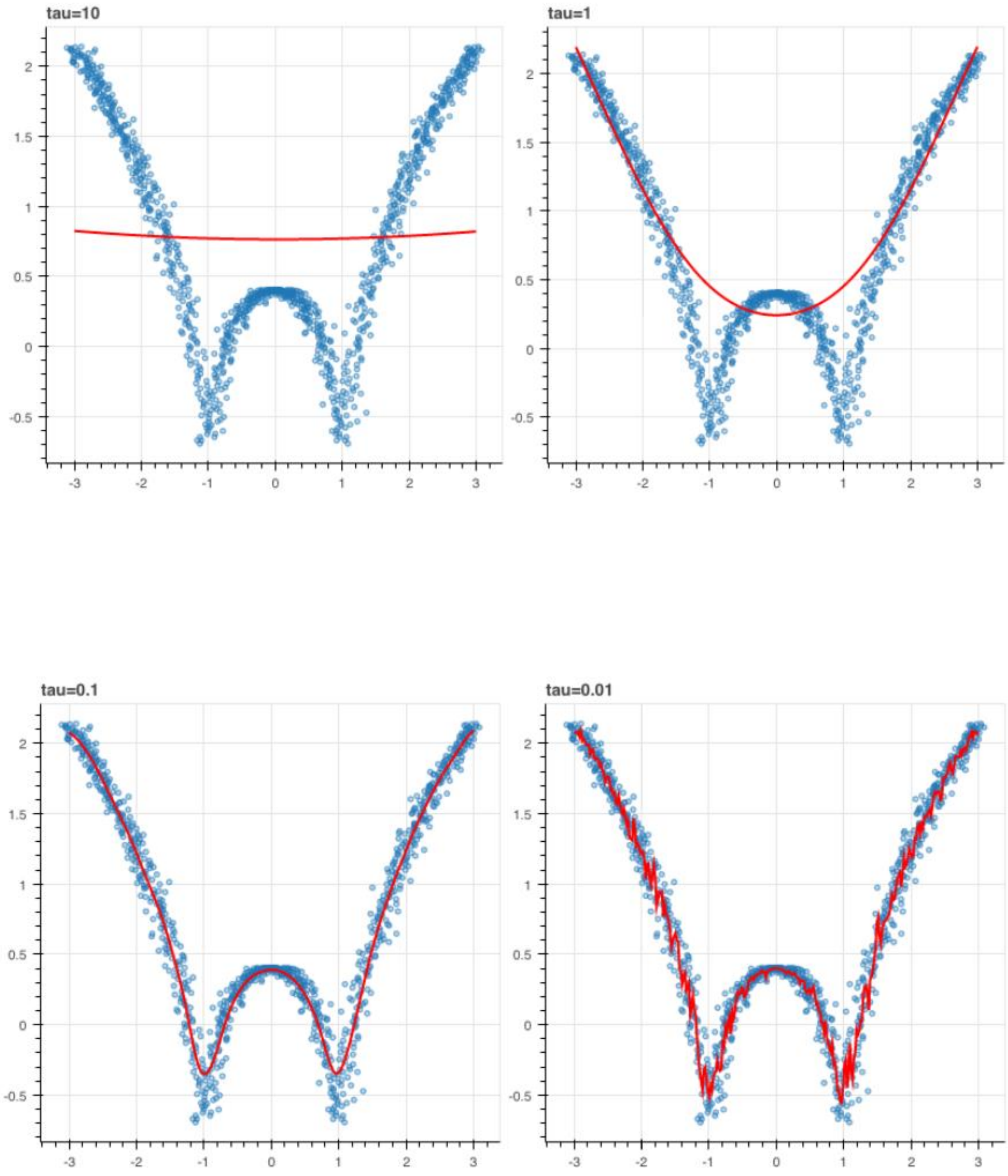
6. Prediction = x0*β:

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)]]))
```

Output



```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
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 dimensional 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,0.3)
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();
```