DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE
(AUTONOMOUS)
(Approved by AICTE, New Delhi and Affiliated to Anna University, Chennai)
Re-Accredited with 'A' Grade By NAAC, Accredited by TCS.
Accredited by NBA (AERO, CSE, IT & MECH)
Re-Accredited by NBA (BME, ECE, EEE)
PERAMBALUR - 621212.

DEPARTMENT OF INFORMATION TECHNOLOGY

LAB MANUAL

U23CSP42 / MACHINE LEARNING LABORATORY

III YEAR / VI SEMESTER

REGUALTION:2023

Prepared By
Mrs.S.ABIRAMI ,
AP/ IT

Approved By
Mr.S.SARAVANAN
HOD / IT

TITLE OF THE EXPERIMENTS	
1	Implement and demonstrate the **FIND-Salgorithm** for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
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.
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.
4	Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.
5	Write a program to implement the **naive Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.
6	Assuming a set of documents that need to be classified, use the **naive 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.
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.
8	Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using **k-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
9	Write a program to implement **k-Nearest Neighbor algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
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.

EXP: NO: 1 **FIND-S ALGORITHM**

AIM:

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.

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 in h 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

- It is Guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples.
- Also Notice that negative examples are ignored.

Limitations of the Find-S algorithm:

- No way to determine if the only final hypothesis (found by Find-S) is consistent with data or there are more hypothesis that is consistent with data.
- Inconsistent sets of training data can mislead the finds algorithm as it ignores negative data samples.
- A good concept learning algorithm should be able to backtrack the choice of hypothesis found so that the resulting hypothesis can be improved over time. Unfortunately, Find-S provide no such method.

PROGRAM :

FindS.py

```
  import numpy as np import pandas as pd
data=pd.read_csv('finds.csv')
print('Data',data)
def train (concepts,target):
    specific_h=concepts[0]
    print('specific1',specific_h)
    for i,h in enumerate(concepts):
print('i',i)
        print('h',h)
        if target[i]=="Yes":
            for x in range(len(specific_h)):
                print('x',x)
                print('specific',specific_h)
                if h[x]==specific_h[x]:
                    pass
                else:
                    specific_h[x]="?"
    return specific_h
concepts=np.array(data.iloc[:,0:-1])
target=np.array(data.iloc[:,-1])
print('Concept',concepts)
print('Target',target)
print(train(concepts,target))
```

Output:

Data	Sky	Air	temp	Humidity	Wind	Water Forecast	Water Sport
0	Sunny	Warm	Normal	Strong	Warm	Same	Yes
1	Sunny	Warm	High	Strong	Warm	Same	Yes
2	Cloudy	Cold	High	Strong	Warm	Change	No
3	Sunny	Warm	High	Strong	Cool	Change	Yes

Concept ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
['Cloudy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
['Sunny' 'Warm' 'High' 'Strong' 'Cool' Change']]
Target ['Yes' 'Yes' 'No' 'Yes'] specific ['Sunny' 'Warm' '?' 'Strong' '?' '?']

EXP NO: 2 **CANDIDATE-ELIMINATION ALGORITHM**

AIM:

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.

ALGORITHM

1. Initialize G to the set of maximally general hypotheses in H
2. Initialize S to the set of maximally specific hypotheses in H
3. For each training example d, do

3.1 If d is a positive example Remove from G any hypothesis inconsistent with d , For each hypothesis s in S that is not consistent with d , Remove s from S .Add to S all minimal generalizations h of s such that h is consistent with d, and some member of G is more general than h Remove from S, hypothesis that is more general than another hypothesis in S

3.2 If d is a negative example Remove from S any hypothesis inconsistent with d For each hypothesis g in G that is not consistent with d Remove g from G Add to G all minimal specializations h of g such that h is consistent with d, and some member of S is more specific than h Remove from G any hypothesis that is less general than another hypothesis in G

PROGRAM :

```
import numpy as np
import pandas as pd
# Loading Data from a CSV File
data = pd.DataFrame(data=pd.read_csv('finds.csv'))
# Separating concept features from Target
concepts = np.array(data.iloc[:,0:-1])
# Isolating target into a separate DataFrame
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
        specific_h=[0,0,0,0,0,0]
```


```
import numpy as np
import pandas as pd
# Loading Data from a CSV File
data = pd.DataFrame(data=pd.read_csv('finds.csv'))
# Separating concept features from Target
concepts = np.array(data.iloc[:,0:-1])
# Isolating target into a separate DataFrame
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
```


```
        specific_h=[0,0,0,0,0,0]
         print('s0',specific_h)
         specific_h = concepts[0].copy()
         print('s1',specific_h)
         general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
         print('g0',general_h)
        for i, h in enumerate(concepts):

            if target[i] == "Yes":
               for x in range(len(specific_h)):
```

```
                # Change values in S & G only if values change
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
                    print(f"s{x}",specific_h)
                    print(f"g{x}",general_h)

            # Checking if the hypothesis has a positive target
            if target[i] == "No":
                for x in range(len(specific_h)):

                    # For negative hyposthesis change values only  in G
                    if h[x] != specific_h[x]:
                        general_h[x][x] = specific_h[x]
                    else:
                        general_h[x][x] = '?'

        # find indices where we have empty rows, meaning those that are unchanged
        indices = [i for i,val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
        for i in indices:
            # remove those rows from general_h
            general_h.remove(['?', '?', '?', '?', '?', '?'])
        print('i',indices)

        # Return final values
        return specific_h, general_h
s_final, g_final = learn(concepts, target)
```

OUTPUT:

```
s0 [0, 0, 0, 0, 0, 0]
s1 ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
g0 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
s2 ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
g2 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
s2 ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
g2 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
s4 ['Sunny' 'Warm' '?' 'Strong' '?' 'Same']
g4 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
s5 ['Sunny' 'Warm' '?' 'Strong' '?' '?']
g5 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
i [2, 3, 4, 5]

print("Final S:", s_final, sep="\n")
Final S:
['Sunny' 'Warm' '?' 'Strong' '?' '?']

print("Final G:", g_final, sep="\n")
Final G:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

EXP NO: 3 **ID3 ALGORITHM**

AIM:

- 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.

ALGORITHM:

ID3(Examples, Target Attribute, Attributes)

Input: Examples are the training examples. Target attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree.

Output: Returns a decision tree that correctly class iJies the given Examples Method:

1. Create a Root node for the tree
2. If all Examples are positive, Return the single-node tree Root, with label = +
3. If all Examples are negative, Return the single-node tree Root, with label = -
4. If Attributes is empty,

   Return the single-node tree Root, with label = most common value of Target Attribute in Examples

   Else

   A ← the attribute from Attributes that best classifies Examples The decision attribute for Root ←A

   For each possible value, vi, of A,

   Add a new tree branch below Root, corresponding to the test A = vi Let Examples vi be the subset of Examples that have value vi for A

   If Examples vi is empty Then below this new branch add a leaf node with label = most common value of Target Attribute in Examples

   Else

   below this new branch add the subtree ID3(Examples vi, Target Attribute, Attributes–{A})

End

5. Return Root

Dataset set used – playtennis.csv

outlook	temp	humidity	wind	play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

PROGRAM:

```
import csv
import math
import random
# Class Node which will be used while classify a test-instance using the tree which was built earlier
class Node():
    value = ""
    children = []
    def __init__(self, val, dictionary):
        self.value = val
        if (isinstance(dictionary, dict)):
            self.children = dictionary.keys()

# Majority Function which tells which class has more entries in given data-set
def majorClass(attributes, data, target):

    freq = {}
    index = attributes.index(target)
    for tuple in data:
        if tuple[index] in freq:
            freq[tuple[index]] += 1
        else:
            freq[tuple[index]] = 1
    max = 0
    major = ""
    for key in freq.keys():
        if freq[key]>max:
            max = freq[key]
            major = key
    return major
 # Calculates the entropy of the data given the target attribute
def entropy(attributes, data, targetAttr):
    freq = {}
    dataEntropy = 0.0
    i = 0
    for entry in attributes:
        if (targetAttr == entry):
            break
        i = i + 1
    i = i - 1
    for entry in data:
        if entry[i] in freq:
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]]  = 1.0
    for freq in freq.values():
        dataEntropy += (-freq/len(data)) * math.log(freq/len(data), 2)
    return dataEntropy
 # Calculates the information gain (reduction in entropy) in the data when a particular attribute is chosen for splitting the data.
def info_gain(attributes, data, attr, targetAttr):

    freq = {}
```

```
    subsetEntropy = 0.0
    i = attributes.index(attr)
    for entry in data:
        if entry[i] in freq:
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]]  = 1.0
    for val in freq.keys():
        valProb        = freq[val] / sum(freq.values())
        dataSubset     = [entry for entry in data if entry[i] == val]
        subsetEntropy += valProb * entropy(attributes, dataSubset, targetAttr)

    return (entropy(attributes, data, targetAttr) - subsetEntropy)

 # This function chooses the attribute among the remaining attributes which has the maximum
information gain.
def attr_choose(data, attributes, target):
    best = attributes[0]
    maxGain = 0;
    for attr in attributes:
        newGain = info_gain(attributes, data, attr, target)
        if newGain>maxGain:
            maxGain = newGain
            best = attr
    return best

 # This function will get unique values for that particular attribute from the given data
def get_values(data, attributes, attr):
    index = attributes.index(attr)
    values = []
    for entry in data:
        if entry[index] not in values:
            values.append(entry[index])
    return values
# This function will get all the rows of the data where the chosen "best" attribute has a value
"val"
def get_data(data, attributes, best, val):
    new_data = [[]]
    index = attributes.index(best)
    for entry in data:
        if (entry[index] == val):
            newEntry = []
            for i in range(0,len(entry)):
                if(i != index):
                    newEntry.append(entry[i])
            new_data.append(newEntry)

    new_data.remove([])
    return new_data
# This function is used to build the decision tree using the given data, attributes and the target
attributes. It returns the decision tree in the end.
def build_tree(data, attributes, target):
    data = data[:]
    vals = [record[attributes.index(target)] for record in data]
    default = majorClass(attributes, data, target)
```

```
    if not data or (len(attributes) - 1) <= 0:
        return default
    elif vals.count(vals[0]) == len(vals):
        return vals[0]
    else:
        best = attr_choose(data, attributes, target)
        tree = {best:{}}
            for val in get_values(data, attributes, best):
            new_data = get_data(data, attributes, best, val)
            newAttr = attributes[:]
            newAttr.remove(best)
            subtree = build_tree(new_data, newAttr, target)
            tree[best][val] = subtree

    return tree

#Main function
def execute_decision_tree():
    data = []
    #load file
    with open("weather.csv") as tsv:
        for line in csv.reader(tsv):
            data.append(tuple(line))
        print("Number of records:",len(data))

        #set attributes
        attributes=['outlook','temperature','humidity','wind','play']
        target = attributes[-1]
        #set training data
        acc = []
        training_set = [x for i, x in enumerate(data)]
        tree = build_tree( training_set, attributes, target )
        #execute algorithm on test data
        results = []
        test_set = [('rainy','mild','high','strong')]
        for entry in test_set:
            tempDict = tree.copy()
result = ""
            while(isinstance(tempDict, dict)):
                root = Node(next(iter(tempDict)), tempDict[next(iter(tempDict))])
                tempDict = tempDict[next(iter(tempDict))]
                index = attributes.index(root.value)
                value = entry[index]
                if(value in tempDict.keys()):
                    child = Node(value, tempDict[value])
                    result = tempDict[value]
                    tempDict = tempDict[value]
                else:
                    result = "Null"
                    break
            if result != "Null":
                results.append(result == entry[-1])
        print(result)
if __name___== "__main__":
    execute_decision_tree()
```

INPUT	OUTPUT
for the input Rain Mild High Strong	**Output 1:** **Number of records: 15** **No**
for the input Rain Cool Normal Weak	Output 2: Number of records: 15 Yes
for the input Overcast Hot Normal strong	Output 3: Number of records: 15 Null

EXP NO: 4 **BACKPROPAGATION ALGORITHM**

AIM:

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

ALGORITHM

1. Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
2. Initialize each wi to some small random value (e.g., between -.05 and .05).
3. Until the termination condition is met, do

   For each training example <(x1,…xn),t>, do

   // Propagate the input forward through the network:

   a. Input the instance (x1, ..,xn) to the n/w & compute the n/w outputs ok for every unit // Propagate the errors backward through the network:

   b. For each output unit k, calculate its error term k ; k = ok(1-ok)(tk-ok)

   c. For each hidden unit h, calculate its error term h; h=oh(1-oh) k wh,k kj xi,j

PROGRAM :

```
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=7000 #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_neurons))
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
  # bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    #bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

//output

Input:

[[0.66666667 1.   ]

 [0.33333333 0.55555556]

 [1.   0.66666667]]

Actual Output:

[[0.92]

 [0.86]

 [0.89]]

Predicted Output:

 [[0.90224607]

 [0.87341151]

 [0.8925683]]

EXP NO: 5 **NAÏVE BAYESIAN CLASSIFIER**

AIM:

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.

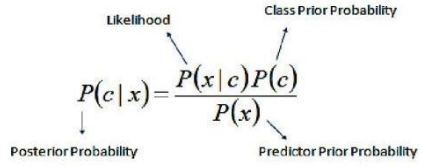
ALGORITHM:

1. Naive Bayes algorithm is a classification technique based on Bayes' Theorem with an assumption of independence among predictors.

2. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

3. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter.

4. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.
5. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.
6. Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

where

P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes). P(c) is the prior probability of class.

P(x|c) is the likelihood which is the probability of predictor given class. P(x) is the prior probability of predictor.

PROGRAM :

Pima Indians Diabetes Database

- This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.
- Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

- Pregnancies(P): Number of times pregnant
- Glucose(G): Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- BloodPressure(BP): Diastolic blood pressure (mm Hg)
- SkinThickness(ST): Triceps skin fold thickness (mm)
- Insulin(I): 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)^2)
- DiabetesPedigreeFunction(DPF): Diabetes pedigree function
- Age(A): Age (years)
- Outcome(O): Class variable (0 or 1)

P	G	BP	ST	I	BMI	DPF	A	O
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

```
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)

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 (float(correct)/float(len(testSet))) * 100.0


  def main():
        filename = 'pima-indians-diabetes.csv'
        dataset = loadCsv(filename)
        trainingSet=dataset
        testSet=loadCsv('pima-indians-diabetes-test-1.csv')
        print('Records in training data={1} and test data={2} rows'.format(len(dataset),
  len(trainingSet), len(testSet)))
        # prepare model
        summaries = summarizeByClass(trainingSet)
        # test model
        predictions = getPredictions(summaries, testSet)
        print(predictions)
        accuracy = getAccuracy(testSet, predictions)
        print("Accuracy:",accuracy,"%")

  main()
```

Output:

```
Records in training data=768 and test data=11 rows
[1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0]
Accuracy: 81.81818181818183 %
```

EXP NO: 6 DOCUMENT CLASSIFICATION USING NAÏVE BAYESIAN CLASSIFIER

AIM:

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.

THEORY:

For the theory of the naive Bayesian classifier refer Experiment No. 5. Theory of performance analysis analysis is elaborated here.

Analysis of Document Classification

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

- For classification tasks, the terms true positives, true negatives, false positives, and false negatives compare the results of the classifier under test with trusted external judgments. The terms positive and negative refer to the classifier's prediction (sometimes known as the expectation), and the terms true and false refer to whether that prediction corresponds to the external judgment (sometimes known as the observation).

- Precision - Precision is the ratio of correctly predicted positive documents to the total predicted positive documents. High precision relates to the low false positive rate.

  Precision = (Σ True positive) / (Σ True positive + Σ False positive)

- Recall (Sensitivity) - Recall is the ratio of correctly predicted positive documents to the all observations in actual class.

  Recall = (Σ True positive) / (Σ True positive + Σ False negative)

- Accuracy - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model. For our model, we have got 0.803 which means our model is approx. 80% accurate.

Accuracy = (Σ True positive + Σ True negative) / Σ Total population

PROGRAM :

Data set

I love this sandwich,pos
This is an amazing place,pos
I feel very good about these beers,pos
This is my best work,pos
What an awesome view,pos
I do not like this restaurant,
neg I am tired of this stuff,neg
I can't deal with this,neg
He is my sworn enemy, neg
My boss is horrible, neg
This is an awesome place,pos
I do not like the taste of this juice,neg
I love to dance,pos
I am sick and tired of this place,neg
What a great holiday,pos
That is a bad locality to stay,neg
We will have good fun tomorrow,pos
I went to my enemy's house today,neg

```
import pandas as pd
msg=pd.read_csv('data61.csv',names=['message','label']) #Tabular form data
print('Total instances in the dataset:',msg.shape[0])

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

print('\nThe message and its label of first 5 instances are listed below')
X5, Y5 = X[0:5], msg.label[0:5]
for x, y in zip(X5,Y5):
    print(x,',',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('\nDataset is split into Training and Testing samples')
print('Total training instances :', xtrain.shape[0])
print('Total testing instances :', xtest.shape[0])


# Output of count vectoriser is a sparse matrix
 # CountVectorizer - stands for 'feature extraction'
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain) #Sparse matrix
xtest_dtm = count_vect.transform(xtest)
print('\nTotal features extracted using CountVectorizer:',xtrain_dtm.shape[1])
```

```
print('\nFeatures for first 5 training instances are listed below')
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df[0:5])#tabular representation
#print(xtrain_dtm) #Same as above but sparse matrix representation

# 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)

print('\nClassstification results of testing samples are given below')
for doc, p in zip(xtest, predicted):
    pred = 'pos' if p==1 else 'neg'
    print('%s -> %s ' % (doc, pred))

#printing accuracy metrics
from sklearn import metrics
print('\nAccuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))

print('Recall :',metrics.recall_score(ytest,predicted),
    '\nPrecison :',metrics.precision_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
```

Output:

```
Total instances in the dataset: 18

The message and its label of first 5 instances are listed below
I love this sandwich , pos
This is an amazing place , pos
I feel very good about these beers , pos
This is my best work , pos
What an awesome view , pos

Dataset is split into Training and Testing samples
Total training instances : 13
Total testing instances : 5

Total features extracted using CountVectorizer: 48

Features for first 5 training instances are listed below
   about  am  an  and  awesome  bad  beers  boss  can  deal  ...  this  \
0      0   1   0    1        0    0      0     0    0     0  ...     1
1      0   0   1    0        1    0      0     0    0     0  ...     0
2      0   1   0    0        0    0      0     0    0     0  ...     1
3      0   0   0    0        0    1      0     0    0     0  ...     0
4      1   0   0    0        0    0      1     0    0     0  ...     0
```

tired	to	tomorrow	very	view	we	what	will	with	
0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	0	0
2	1	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0

[5 rows x 48 columns]
Classstification results of testing samples are given below
This is my best work -> neg
I went to my enemy's house today -> neg
I love to dance -> pos
I do not like this restaurant -> neg
This is an amazing place -> pos

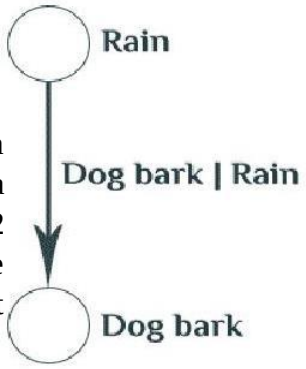
Accuracy metrics
Accuracy of the classifer is 0.8
Recall : 0.666666666667
Precison : 1.0
Confusion matrix
[[2 0]
 [1 2]]

EXP NO: 7 **BAYESIAN NETWORK**

AIM:

- 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.

PROCEDURE:



- Bayesian networks are very convenient for representing similar probabilistic relationships between multiple events.
- Bayesian networks as graphs - People usually represent Bayesian networks as directed graphs in which each node is a hypothesis or a random process. In other words, something that takes at least 2 possible values you can assign probabilities to. For example, there can be a node that represents the state of the dog (barking or not barking at the window), the weather (raining or not raining), etc.
- The arrows between nodes represent the conditional probabilities between them — how information about the state of one node changes the probability distribution of another node it's connected to.

PROGRAM :

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import Maximum LikelihoodEstimator
from pgmpy.models import BayesianModel
from adpgmpy.inference import VariableElimination
#Read the attributes
lines = list(csv.reader(open('data7_names.csv', 'r')));
attributes = lines[0]
#attributes = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang',
# 'oldpeak', 'slope', 'ca', 'thal', 'heartdisease']
heartDisease = pd.read_csv('data7_heart.csv', names = attributes)
heartDisease = heartDisease.replace('?', np.nan)

# Display the data
print('Few examples from the dataset are given below')
print(heartDisease.head())
print('\nAttributes and datatypes')
print(heartDisease.dtypes)

# Model Baysian Network
model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),
('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),
('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])
```

```
# Learning CPDs using Maximum Likelihood Estimators
print('\nLearning CPDs using Maximum Likelihood Estimators...');
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network
print('\nInferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(model)

# Computing the probability of bronc given smoke.
print('\n1.Probability of HeartDisease given Age=20')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q['heartdisease'])
print('\n2. Probability of HeartDisease given chol (Cholestoral) =100')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'chol': 100})
print(q['heartdisease'])
```

OUTPUT:

Few examples from the dataset are given below

	age	sex	cp	trestbps	...	slope	ca	thal	heartdisease
0	63.0	1.0	1.0	145.0	...	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	...	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	...	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	...	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	...	1.0	0.0	3.0	0

[5 rows x 14 columns]

Attributes and datatypes

age	float64
sex	float64
cp	float64
trestbps	float64
chol	float64
fbs	float64
restecg	float64
thalach	float64
exang	float64
oldpeak	float64
slope	float64
ca	object
thal	object
heartdisease	int64

Learing CPDs using Maximum Likelihood Estimators...

Inferencing with Bayesian Network:

1.Probability of HeartDisease given Age=20

heartdisease	phi(heartdisease)
heartdisease_0	0.6791
heartdisease_1	0.1212
heartdisease_2	0.0810
heartdisease_3	0.0939
heartdisease_4	0.0247

2. Probability of HeartDisease given chol (Cholestoral) =100

heartdisease	phi(heartdisease)
heartdisease_0	0.5400
heartdisease_1	0.1533
heartdisease_2	0.1303
heartdisease_3	0.1259
heartdisease_4	0.0506

EXP. NO: 8 CLUSTERING BASED ON EM ALGORITHM AND K-MEANS

AIM:

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

ALGORITHM:

1. First we initialize k points, called means, randomly.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

- The "points" mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set (if for a feature x the items have values in [0,3], we will initialize the means with values for x at [0,3]).
- Pseudocode:

1. Initialize k means with random values
2. For a given number of iterations:

Iterate through items:

Find the mean closest to the item Assign item to mean

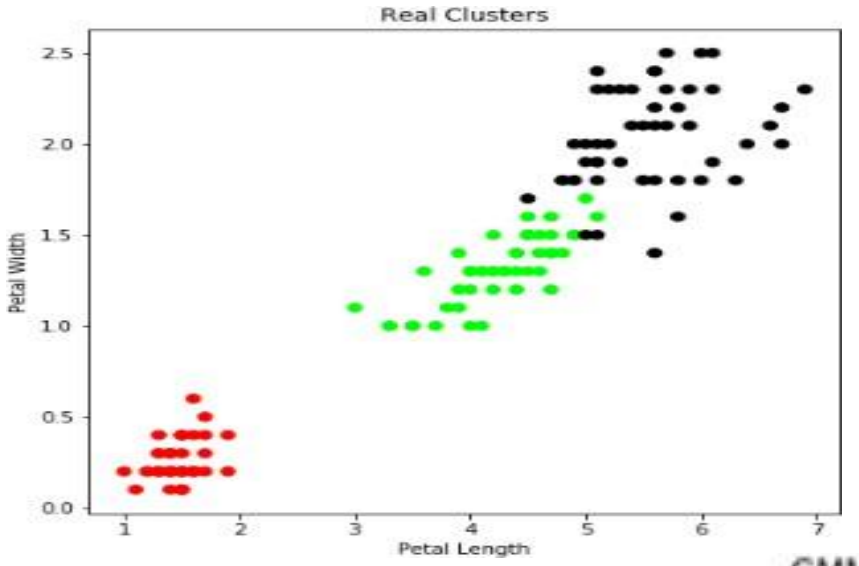
Update mean

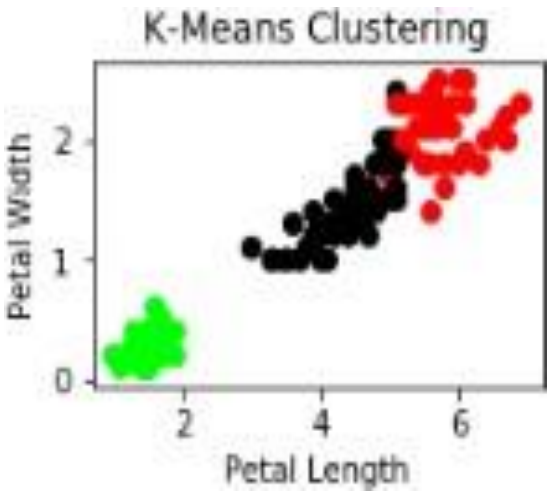
PROGRAM :

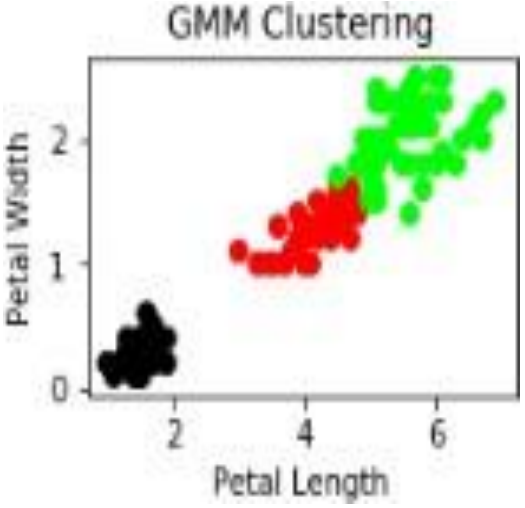
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

# import some data to play with
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']

# Build the K Means Model
model = KMeans(n_clusters=3)
# model.labels_ : Gives cluster no for which samples belongs to
model.fit(X)

# Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications using Petal features
```

```
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()

# Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()

# General EM for GMM
from sklearn import preprocessing
# transform your data such that its distribution will have a
# mean value 0 and standard deviation of 1.
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)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
print('Observation: The GMM using EM algorithm based clustering matched the true labels
more closely than the Kmeans.')
```

OUTPUT:


Real Clusters
Petal Width
Petal Length
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7


K-Means Clustering
Petal Width
Petal Length
0
1
2
2
4
6


GMM Clustering
Petal Width
Petal Length
0
1
2
2
4
6

EXP. NO: 9 **K-NEAREST NEIGHBOUR**

AIM:

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.

ALGORITHM:

Input: Let m be the number of training data samples. Let p be an unknown point.
Method:

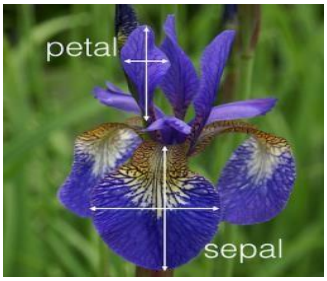
1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).
2. for i=0 to m
   Calculate Euclidean distance d(arr[i], p).
3. Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.
4. Return the majority label among S.

PROGRAM :

Dataset: (150 Rows)

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
1	5.1	3.5	1.4	0.2	*I. setosa*
2	4.9	3.0	1.4	0.2	*I. setosa*
3	4.7	3.2	1.3	0.2	*I. setosa*
98	6.2	2.9	4.3	1.3	*I. versicolor*
99	5.1	2.5	3.0	1.1	*I. versicolor*
100	5.7	2.8	4.1	1.3	*I. versicolor*
101	6.3	3.3	6.0	2.5	*I. virginica*
102	5.8	2.7	5.1	1.9	*I. virginica*
103	7.1	3.0	5.9	2.1	*I. virginica*

Iris flower sample image




```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

# Load dataset
iris=datasets.load_iris()
print("Iris Data set loaded...")

# Split the data into train and test samples
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)
print("Dataset is split into training and testing...")
print("Size of trainng data and its label",x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape)
```

```
# Prints Label no. and their names
for i in range(len(iris.target_names)):
        print("Label", i , "-",str(iris.target_names[i]))
# Create object of KNN classifier
classifier = KNeighborsClassifier(n_neighbors=1)

# Perform Training
classifier.fit(x_train,  y_train)

# Perform testing
y_pred=classifier.predict(x_test)

# Display the results
print("Results of Classification using K-nn with
K=1 ") for r in range(0,len(x_test)):
    print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r]), " Predicted-
label:", str(y_pred[r]))
print("Classification Accuracy :" , classifier.score(x_test,y_test));

from sklearn.metrics import classification_report, confusion_matrix
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

OUTPUT:

```
Iris Data set loaded...
Dataset is split into training and testing...
Size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
Label 1 - versicolor
Label 2 – virginica

Results of Classification using K-nn with K=1

Sample: [ 6.   2.2  4.   1. ]  Actual-label: 1  Predicted-label: 1
 Sample: [ 4.8  3.   1.4  0.3]  Actual-label: 0  Predicted-label: 0
 Sample: [ 5.8  2.6  4.   1.2]  Actual-label: 1  Predicted-label: 1
 Sample: [ 5.8  2.7  3.9  1.2]  Actual-label: 1  Predicted-label: 1
 Sample: [ 5.1  3.5  1.4  0.2]  Actual-label: 0  Predicted-label: 0
 Sample: [ 6.3  2.3  4.4  1.3]  Actual-label: 1  Predicted-label: 1
 Sample: [ 6.   2.7  5.1  1.6]  Actual-label: 1  Predicted-label: 2
 Sample: [ 6.7  3.1  5.6  2.4]  Actual-label: 2  Predicted-label: 2
 Sample: [ 5.5  4.2  1.4  0.2]  Actual-label: 0  Predicted-label: 0
 Sample: [ 5.6  3.   4.5  1.5]  Actual-label: 1  Predicted-label: 1
 Sample: [ 6.1  3.   4.9  1.8]  Actual-label: 2  Predicted-label: 2
 Sample: [ 6.   3.4  4.5  1.6]  Actual-label: 1  Predicted-label: 1
 Sample: [ 6.1  2.6  5.6  1.4]  Actual-label: 2  Predicted-label: 2
 Sample: [ 5.5  2.4  3.8  1.1]  Actual-label: 1  Predicted-label: 1
 Sample: [ 5.6  2.9  3.6  1.3]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.933333333333

Confusion Matrix
[[3 0 0]
 [0 8 1]
 [0 0 3]]
Accuracy Metrics
             precision    recall  f1-score   support

          0       1.00      1.00      1.00         3
          1       1.00      0.89      0.94         9
          2       0.75      1.00      0.86         3

avg / total       0.95      0.93      0.94        15
```

EXP NO: 10 LOCALLY WEIGHTED REGRESSION ALGORITHM

AIM:

- Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

ALGORITHM:

- Given a dataset X, y, we attempt to find a linear model h(x) that minimizes residual sum of squared errors. The solution is given by Normal equations.
- Linear model can only fit a straight line, however, it can be empowered by polynomial features to get more powerful models. Still, we have to decide and fix the number and types of features ahead.
- Alternate approach is given by locally weighted regression.
- Given a dataset X, y, we attempt to find a model h(x) that minimizes residual sum of weighted squared errors.
- The weights are given by a kernel function which can be chosen arbitrarily and in my case I chose a Gaussian kernel.
- The solution is very similar to Normal equations, we only need to insert diagonal weight matrix W.

PROGRAM : Dataset: (244 Rows)

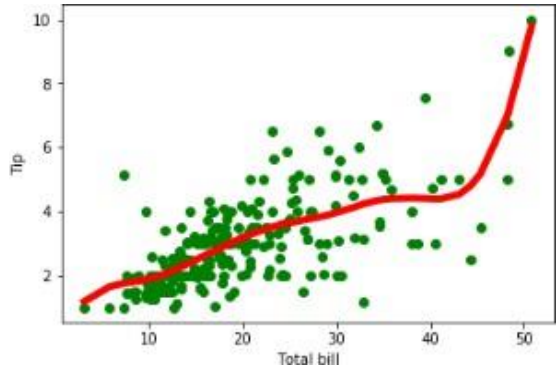
total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	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	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2
21.58	3.92	Male	No	Sun	Dinner	2

```
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))) # eye - identity matrix
    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

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    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();


# load data points
data = pd.read_csv('data10_tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)
mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols
# increase k to get smooth curves
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```

OUTPUT:

Regession with parameter k = 3



Regession with parameter k = 9