



Kammavari Sangham (R)-1952

K.S.GROUP OF INSTITUTIONS

K.S. INSTITUTE OF TECHNOLOGY

ACCREDITED BY NAAC

(Approved by AICTE & Affiliated to VTU)

KANAKPURA ROAD, BENGALURU - 560 109

A P J ABDUL KALAM

"Tell me and I'll forget;
Show me and I may
remember; Involve me
and I'll understand"

GRAHAM BELL

MARIE CURIE

SRINIVASA
RAMANUJAN

J R D TATA

THOMAS ALVA EDISON

DENNIS RITCHIE

PRACTICAL RECORD

NAME : ANUSHREE . J.

SEM/ BRANCH : 7 'A' / COMPUTER SCIENCE
MACHINE LEARNING (17CSL76)

SUBJECT & CODE : RECORD

USN :

1	K	S	I	4	C	S	O	I	I
---	---	---	---	---	---	---	---	---	---

K.S. INSTITUTE OF TECHNOLOGY

VISION

“To impart quality technical education with ethical values, employable skills and research to achieve excellence”

MISSION

- * To attract and retain highly qualified, experienced & committed faculty.
- * To create relevant infrastructure
- * Network with industry & premier institutions to encourage emergence of new ideas by providing research & development facilities to strive for academic excellence
- * To inculcate the professional & ethical values among young students with employable skills & knowledge acquired to transform the society



KSIT
K.S. INSTITUTE OF TECHNOLOGY

Kammavari Sangham (R)-1952

K.S. GROUP OF INSTITUTIONS

K.S. INSTITUTE OF TECHNOLOGY

ACCREDITED BY NAAC

(Approved by AICTE & Affiliated to VTU)

KANAKPURA ROAD, BENGALURU - 560 109

Laboratory Certificate

This is to certify that *Mr./Ms. ...ANUSHREE.J...*
has satisfactorily completed the course of experiments in
MACHINE LEARNING laboratory, Code: *17CSL76*
prescribed by Visvesvaraya Technological University, Belgaum for
the *4th A* Semester *B.E. COMPUTER SCIENCE ENGG* Branch
in this College during the academic year *20.20. - 20.21.*

Name of the Candidate : *ANUSHREE.J.*

USN : *1KS17CS011* Subject (with code) : *MACHINE LEARNING (17CSL76) LABORATORY*

Internal assessment marks awarded :

30	10	40
30	10	40

Signature of Staff Incharge

Date: *15/01/2021*

[Signature]
Signature of Head of
Head of the Department
Dept. of Computer Science & Engg.
K.S. Institute of Technology
Bengaluru - 560 109

INSTRUCTIONS FOR MAINTAINING THE PRACTICAL RECORD

1. Record should be written neatly in ink on the right side page only, left side pages being reserved for diagrams and graphs in pencil.
2. Record should contain
 - Number and name of the Experiment
 - The date
 - Principle
 - Procedure
 - Observation calculation (to be entered on the left hand side pages in neat tabular forms wherever applicable)
 - Results

CONTENTS AND EVALUATION

Sl.No.	Date of Conducting Experiment	Date of Submission	Page No.	Title of the Experiment	Assessment Marks			Signature of Staff with Date
					Experiment 10 Marks	Record 20 Marks	Total	
01	14/9/20	21/9/20	1-3	FIND-S ALGORITHM	10	20	30	S. Ragh 15/01/2021
02	21/9/20	12/10/20	4-6	Candidate Elimination Algorithm	10	20	30	
03	12/10/20	2/11/20	7-13	ID3 Algorithm	10	20	30	
04	2/11/20	9/11/20	14-17	Artificial Neural Networks	10	20	30	
05	09/11/20	13/11/20	18-24	Naive Bayes Classifier	10	20	30	
06	13/11/20	30/11/20	25-27	Accuracy, precision of Naive Bayes Classifier	10	20	30	
07	30/11/20	7/12/20	28-31	Bayesian Network	10	20	30	
08	07/12/20	14/12/20	32-34	EM Algorithm	10	20	30	
09	14/12/20	21/12/20	35-37	K-Nearest Neighbour	10	20	30	
10	21/12/20	20/1/21	38-40	Locally weighted Regression	10	20	30	

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.

FIND-S ALGORITHM:

Step 1: Initialize h to the most specific hypothesis in H .

Step 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 .

Step 3: Output hypothesis h .

CODE:

```
import numpy as np
import pandas as pd
data = pd.read_csv('1 finds.csv')
print(data)
```



```
concepts = data.iloc[:, 0:-1].values.  
print("-----").  
print(concepts)  
print("-----")  
target = data.iloc[:, -1].values  
print(target)
```

```
def train (concepts, target):  
    count = 0  
    specific_h = concepts [0]  
    for i, h in enumerate (concepts):  
        print(i)  
        print(h)  
        if target[i] == "Yes":  
            for x in range (len (specific_h)):  
                if h[x] == specific_h[x]:  
                    pass  
                else:  
                    specific_h[x] = "?"  
            count = count + 1  
            print (f " Hypothesis after sample number: {count}  
                processed : {specific_h}")  
        else:  
            pass  
            count = count + 1  
            print (f " Negative sample number: {count}  
                same Hypothesis : {specific_h}")
```

DATE 14/9/20
EXP. NO. 01

EXPT. TITLE :

PAGE NO. 03

return specific-h

specific-h = ~~train~~ (concepts, target)

OBSERVATION	10
RECORD	10
VIVA	10
TOTAL	30
SIGNATURE	S. Ragh

DATASET:-

1	Sky	AirTemp	Humidity	Wind	Water	Forecast	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:

Input:
Finds.csv

Output:

```
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']  
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']  
 ['Cloudy' 'Cold' 'High' 'Strong' 'Warm' 'Change']  
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]  
  
['Yes' 'Yes' 'No' 'Yes']  
0  
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']  
Hypothesis after sample Number2 is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm'  
'Same']  
1  
['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']  
Hypothesis after sample Number3 is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Sam  
e']  
2  
['Cloudy' 'Cold' 'High' 'Strong' 'Warm' 'Change']  
Negative Hypothesis after sample Number4 is ['Sunny' 'Warm' '?' 'Strong' 'W  
arm' 'Same']  
3  
['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']  
Hypothesis after sample Number5 is ['Sunny' 'Warm' '?' 'Strong' '?' '?']
```


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 example.

CANDIDATE-ELIMINATION ALGORITHM:

Initialize C_1 to the set of maximally general hypotheses in H

Initialize S to the set of maximally specific hypotheses in H

For each training examples d , do

* If d is a positive example

- Remove from C_1 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 C_1 is more general than h

- Remove from S any hypothesis that is more general than another hypothesis in S

* If d is a negative example.

- Remove from S any hypothesis inconsistent with d
- For each hypothesis g in C_1 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 C is more general than h
- Remove from S any hypothesis that is more general than another hypothesis in S .

CODE :-

```
import numpy as np
import pandas as pd
```

```
data = pd.DataFrame(data = pd.read_csv('finds.csv'))
concepts = np.array(data.iloc[:, 0:-1])
target = np.array(data.iloc[:, -1])
```

```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in hypothesis(specific_h)]
                 for i in range(len(specific_h))]
```

```
    for i, h in range(len(specific_h)):
        if h[x] != specific_h[x]:
            specific_h[x] = '?'
            general_h[x][x] = '?'
```



```

if target[i] == "No";
    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] = '?'

```

```

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 S:", s-final, sep="\n")
print ("Final G:", g-final, sep="\n")

OBSERVATION	10	
RECORD	10	
VIVA	10	
TOTAL	30	
SIGNATURE	S. Ray	

DATASET:

1	Sky	Airtemp	Humidity	Wind	Water	Forecast	WaterSport
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes
3	Sunny	Warm	High	Strong	Warm	Same	Yes
4	Cloudy	Cold	High	Strong	Warm	Change	No
5	Sunny	Warm	High	Strong	Cool	Change	Yes

OUTPUT:

out put

Final S:

['Sunny' 'Warm' '?' 'Strong' '?' '?']

Final G:

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

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

ID3 ALGORITHM:

ID3(Examples, Target-attribute, Attributes)

- * Create a Root node for a tree
- * If all Examples are positive, Return the single-node tree Root, with label = +
- * If all Examples are negative, Return the single-node tree Root, with label = -
- * If Attributes is empty, Return the single-node tree Root, with label = most common value of Target-attribute in Examples
 - * otherwise Begin
- * $A \leftarrow$ the attribute from Attributes that best classifies Examples.
- * The decision attribute for Root $\leftarrow A$
- * For each possible value, v_i , of A ,
 - Add a new tree branch below Root, corresponding to the test $A = v_i$

- Let Examples v_i , be the subset of Examples that have value v_i for A.
 - If Examples v_i , 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 v_i , Target-attribute, Attributes - A)
- End
Return Root.

ID3 program:

```
import csv
import math
import random
```

```
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.
```

```
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
```



```
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)
```

```
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
```



```
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.
```

```
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.
```

```
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[1:]  
    newAttr.remove(best)  
    subtree = build_tree(new_data, newAttr, target)  
    tree[best][val] = subtree  
return tree.
```

```
def execute_decision_tree():  
    data = []
```

```
    with open("weather.csv") as tsv:  
        for line in csv.reader(tsv):  
            data.append(tuple(line))  
    print("Number of records: ", len(data))
```

```
    attributes = ['outlook', 'temperature', 'humidity', 'wind', 'play']  
    target = attributes[-1]
```

```
    acc = []
```

```
    training_set = [x for i, x in enumerate(data)]  
    tree = build_tree(training_set, attributes, target)  
    print(tree).
```



```
results = []
```

```
test-set = [('rainy', 'mild', 'high', 'strong')]
```

```
for entry in test-set:
```

```
    tempDict = tree.copy()
```

```
    result = ""
```

```
    while (isinstance(tempDict, dict)):
```

```
        child = []
```

```
        nodeVal = next(iter(tempDict))
```

```
        child = tempDict[next(iter(tempDict))].keys()
```

```
        tempDict = tempDict[next(iter(tempDict))]
```

```
        index = attributes.index(nodeVal)
```

```
        value = entry[index]
```

```
        if (value in tempDict.keys()):
```

```
            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()
```

OBSERVATION	10	
RECORD	10	
VIVA	10	
TOTAL	30	
SIGNATURE	<i>[Signature]</i>	

DATASET:

1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rainy	mild	high	weak	yes
5	rainy	cool	normal	weak	yes
6	rainy	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rainy	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rainy	mild	high	strong	no

OUT PUT:

Input:

Weather.csv

Output:

Number of records: 15

{'wind': {'wind': 'play', 'weak': {'humidity': {'high': {'temperature': {'hot': {'outlook': {'sunny': 'no', 'overcast': 'yes'}}}, 'mild': {'outlook': {'rainy': 'yes', 'sunny': 'no'}}}}, 'normal': 'yes'}}}, 'strong': {'humidity': {'high': {'outlook': {'sunny': 'no', 'overcast': 'yes', 'rainy': 'no'}}}, 'normal': {'outlook': {'rainy': 'no', 'overcast': 'yes', 'sunny': 'yes'}}}}}

no

AIM:-

Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data set.

ANN program :

```
from math import exp
from random import seed
from random import random

def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{ 'weights': [random() for i in range(
        n_inputs + 1)] } for i in range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{ 'weights': [random() for i in range(
        n_hidden + 1)] } for i in range(n_outputs)]
    network.append(output_layer)
    return network

def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

def forward_propagate(network, row):
    inputs = row
```

```
for layer in network:
    new_inputs = []
    for neuron in layer:
        activation = activate(neuron['weights'], inputs)
        neuron['output'] = transfer(activation)
        new_inputs.append(neuron['output'])
    inputs = new_inputs
return inputs
```

```
def transfer_derivative(output):
    return output * (1.0 - output)
```

```
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network) - 1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j]
                              * neuron['delta'])
                errors.append(error)
            else:
                for j in range(len(layer)):
                    neuron = layer[j]
                    errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
```



```
neuron = layers[j]
```

```
neuron['delta'] = errors[j] * transfer-derivative  
                      (neuron['output'])
```

```
def update_weights(network, row, l_rate):
```

```
    for i in range(len(network)):
```

```
        inputs = row[:-1]
```

```
        if i != 0:
```

```
            inputs = [neuron['output'] for neuron  
                      in network[i-1]]
```

```
        for neuron in network[i]:
```

```
            for j in range(len(inputs)):
```

```
                neuron['weights'][j] += l_rate *  
                    neuron['delta'] * inputs[j]
```

```
            neuron['weights'][-1] += l_rate * neuron['delta']
```

```
def train_network(network, train, l_rate, n_epoch, n_outputs):
```

```
    for epoch in range(n_epoch):
```

```
        sum_error = 0
```

```
        for row in train:
```

```
            outputs = forward-propagate(network, row)
```

```
            expected = [0 for i in range(n_outputs)]
```

```
            expected[row[-1]] = 1
```

```
            sum_error += sum([(expected[i] - outputs[i])**2
```

```
                              for i in range(len(expected))])
```

```
            backward-propagate-errors(network, expected)
```

```
            update_weights(network, row, l_rate)
```

```
            print('> epoch=%d, lrate=%0.3f,
```

```
                  error=%0.3f' % (epoch, l_rate, sum_error))
```


seed(1)

```
dataset = [[2.7810836, 2.550537003, 0],
            [1.4658942, 2.362125076, 0],
            [3.396561688, 4.400293529, 0],
            [1.32807019, 1.850220317, 0],
            [3.06407232, 3.005305973, 0],
            [7.6275312, 2.759262235, 1],
            [5.33241248, 2.088626775, 1],
            [6.922596716, 1.77106367, 1],
            [8.675418651, -0.242068655, 1],
            [7.673756466, 3.508563011, 1]]
```

n_inputs = len(dataset[0]) - 1

n_outputs = len(set([row[-1] for row in dataset]))

network = initialize_network(n_inputs, 2, n_outputs)

print(network)

train_network(network, dataset, 0.5, 20, n_outputs)

for layer in network:

print(layer)

OBSERVATION	10
RECORD	10
VIVA	10
TOTAL	30
SIGNATURE	SRAD

OUTPUT:

Output

```
>epoch=0, lr=0.500, error=6.350
>epoch=1, lr=0.500, error=5.531
>epoch=2, lr=0.500, error=5.22147
>epoch=3, lr=0.500, error=4.951
>epoch=4, lr=0.500, error=4.519
>epoch=5, lr=0.500, error=4.173
>epoch=6, lr=0.500, error=3.835
>epoch=7, lr=0.500, error=3.506
>epoch=8, lr=0.500, error=3.192
>epoch=9, lr=0.500, error=2.898
>epoch=10, lr=0.500, error=2.626
>epoch=11, lr=0.500, error=2.377
>epoch=12, lr=0.500, error=2.153
>epoch=13, lr=0.500, error=1.953
>epoch=14, lr=0.500, error=1.774
>epoch=15, lr=0.500, error=1.614
>epoch=16, lr=0.500, error=1.472
>epoch=17, lr=0.500, error=1.346
>epoch=18, lr=0.500, error=1.233
>epoch=19, lr=0.500, error=1.132
[{'weights': [-1.4688375095432327, 1.850887325439514, 1.0858178629550297], 'output':
0.029980305604426185, 'delta': -0.0059546604162323625}, {'weights':
[0.37711098142462157, -0.0625909894552989, 0.2765123702642716], 'output':
0.9456229000211323, 'delta':
0.0026279652850863837}]
[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275], 'output':
0.23648794202357587, 'delta': -0.04270059278364587}, {'weights': [-2.5584149848484263,
1.0036422106209202, 0.42383086467582715], 'output': 0.7790535202438367, 'delta':
0.03803132596437354}]
```

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.

Bayes theorem provides a way of calculating posterior probability $P(c|x)$ from $P(c)$, $P(x)$ & $P(x|c)$. Look at the equation below:

1) Handling of Data:

- * Load the data from the CSV file and split it into training and test data set.
- * Training data set can be used to by Naïve Bayes to make predictions.
- * And Test data set can be used to evaluate the accuracy of the model.

2) Summarize Data:

The summary of the training data collected involves the mean and the standard deviation for each attribute, by class value.

- * These are required when making predictions to calculate the probability of specific attribute values belonging to each class value.
- * Summary data can be break down into the following sub-tasks:

* Separate Data By Class: The first task is to separate the training dataset instances by class value so that we can calculate statistics for each class. We can do that by creating a map of each class. We can do that class and sort the entire dataset of instances into the appropriate lists.

* Calculate Mean: We need to calculate the mean of each attribute for a class value. The mean is the central middle or central tendency of the data, and we will use it as the middle of our gaussian distribution when calculating probabilities.

* Calculate Standard Deviation: We also need to calculate deviation of each attribute for a class value. The standard deviation describes the variation or spread of the data, and we will use it to characterize the expected spread of each attribute in our gaussian distribution when calculating probabilities.

* Summarize Dataset: For a given list of instances we can calculate the mean and the standard deviation for each attribute.

* Summarize Attributes By Class: We can pull it all together by first separating our training dataset

into instances grouped by class. Then calculate the summaries for each attribute.

3) Make Predictions:

- * Making predictions involves calculating the probability that a given data instance belongs to each class,

- * then selecting the class with the largest probability as the prediction.

- * Finally, estimation of the accuracy of the model by making predictions for each data instance in the test dataset.

4) Evaluate Accuracy: The predictions can be compared to the class values in the test dataset and a classification/accuracy can be calculated as an accuracy ratio b/w 0 & 100%.

CODE:-

```
import csv
import random
import math
```

```
def safe-div(x,y):
    if y == 0:
        return 0
    return x/y
```



```
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 = separatedByClass(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 = 'pima-indians-diabetes.data.csv'
    splitRatio = 0.67
    dataset = loadCsv(filename)
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print('Split %03 rows into train = %13 and
    test = %23 rows'.format(len(dataset), len(trainingSet),
    len(testSet)))
    summaries = summarizeByClass(trainingSet)
    predictions = getPredictions(summaries, testSet)
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: %03%', format(accuracy))
```

main()

OBSERVATION	10	
RECORD	10	
VIVA	10	
TOTAL	30	
SIGNATURE	S. Rajl	

DATASET:

1	1	1	1	5
1	1	1	2	5
2	1	1	2	10
3	2	1	1	5
3	3	2	1	5
3	3	2	2	5
2	3	2	2	10
1	2	1	1	5
1	3	2	1	10
3	2	2	2	10
1	2	2	2	10
2	2	1	2	10
2	1	2	1	10
3	2	1	2	5
1	2	1	2	10
1	2	1	2	5

OUTPUT:

OUTPUT:

Split 768 rows into train=576 and test=192 rows

Accuracy: 77.604 %

AIM:

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.

ALGORITHM:LEARN-NAIVE-BAYES-TEXT(Examples, V)

1. collect all words, punctuation and other tokens that occur in Examples.
 - Vocabulary $\leftarrow$ 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 $\leftarrow$ the subset of documents from Examples for which the target value is v_j
 - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
 - Text_j $\leftarrow$ a single document created by concatenating all members of docs_j
 - $n \leftarrow$ total number of distinct word positions in Text_j

- for each word w_k in Vocabulary
 - $n_k \leftarrow$ number of times word w_k occurs in Text $_j$
 - $P(w_k | v_j) \leftarrow \frac{n_k + 1}{n + |\text{Vocabulary}|}$

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 $\leftarrow$ all word positions in Doc that contain tokens found in Vocabulary.
- Return V_{NB} , where

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

PROGRAM:

```
import pandas as pd
msg = pd.read_csv('6pg.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)
```



```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y)
print(xtest.shape)
print(xtrain.shape)
print(ytest.shape)
print(ytrain.shape)
```

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm = count_vect.transform(xtest)
print(count_vect.get_feature_names())
df = pd.DataFrame(xtrain_dtm.toarray(), columns =
    count_vect.get_feature_names())
print(df)
print(xtrain_dtm)
```

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm, ytrain)
predicted = clf.predict(xtest_dtm)
```

```
from sklearn import metrics
print('Accuracy metrics')
print('Accuracy of the classifier is', metrics.accuracy_score(ytest, predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest, predicted))
print('Recall and Precision')
print(metrics.recall_score(ytest, predicted))
print(metrics.precision_score(ytest, predicted))
```

DATASET:

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

OUTPUT:

Output

2357

1502

[alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']

From: sd345@city.ac.uk (Michael Collier)

Subject: Converting images to HP LaserJet III?

Nntp-Posting-Host: hampton

Organization: The City University

Lines: 14

Does anyone know of a good way (standard PC application/PD utility) to convert tif/ing/tga files into LaserJet III format. We would also like to do the same, converting to HPGL (HP plotter) files.

Please email any response.

Is this the correct group?

Thanks in advance. Michael.

--

Michael Collier (Programmer) The Computer Unit,

Email: M.P.Collier@uk.ac.city The City University,

Tel: 071 477-8000 x3769 London,

Fax: 071 477-8565 EC1V 0HB.

1

Accuracy: 0.8348868175765646

precision recall f1-score support

alt.atheism 0.97 0.60 0.74 319

comp.graphics 0.96 0.89 0.92 389

sci.med 0.97 0.81 0.88 396

soc.religion.christian 0.65 0.99 0.78 398

avg / total 0.88 0.83 0.84 1502

confusion matrix is

[[192 26 119]

[2 347 4 36]

[2 11 322 61]

[2 2 1 393]]

OBSERVATION	10
RECORD	10
VIVA	10
TOTAL	30
SIGNATURE	S. R. J. L.

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.

Following steps are used to build the Bayesian network.

Step 1: Identify the variables which is set of attributes specified in the dataset (or Medical Dataset)

Step 2: Determine the domain of each variable that is set of values a variable may take

Step 3: Create a directed graph network of nodes where each node represents the attribute and edges represent parent child relationship. Edge represents that the child variable is conditionally dependent on the parent.

Step 4: determine the prior and conditional probability for each attribute

Step 5: perform the inference on the model and determine the marginal probabilities.

PROGRAM:

```
import bayespy  
as by import  
numpy as np  
import csv
```

```
from colorama import init  
from colorama import Fore, Back, style  
init()
```

```
ageEnum = {'superSeniorCitizen': 0, 'seniorCitizen': 1, 'Middle Aged': 2,  
           'Youth': 3, 'Teen': 4}
```

```
genderEnum = {'Male': 0, 'Female': 1}
```

```
familyHistoryEnum = {'Yes': 0, 'No': 1}
```

```
dietEnum = {'High': 0, 'Medium': 1, 'Low': 2}
```

```
lifestyleEnum = {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedentary': 3}
```

```
cholesterolEnum = {'High': 0, 'Border Line': 1, 'Normal': 2}
```

```
heartDiseaseEnum = {'Yes': 0, 'No': 1}
```

```
with open('heart-disease-data.csv') as csvfile:  
    lines = csv.reader(csvfile)
```

```
dataset = list(lines)
```

```
data = []
```

```
for x in dataset:
```

```
    data.append([ageEnum[x[0]], genderEnum[x[1]],  
                familyHistoryEnum[x[2]], dietEnum[x[3]], lifestyle  
                Enum[x[4]], cholesterolEnum[x[5]], heartDiseaseEnum[x[6]])
```

```
p_age = bp.nodes.Dirichlet(1.0 * np.ones(5))
```

```
age = bp.nodes.Categorical(p_age, plates=(N, 1)) age.observe  
      (data[:, 0])
```

```
p_gender = bp.nodes.Dirichlet(1.0 * np.ones(2))
```

```
gender = bp.nodes.Categorical(p_gender, plates=(N, 1))
```

```
age.observe(data[:, 1])
```

```
p_familyHistory = bp.nodes.Dirichlet(1.0 * np.ones(2))
```

```
familyHistory = bp.nodes.Categorical(p_familyHistory, plates=(N, 1))
```

```
familyHistory.observe(data[:, 2])
```

```
p_diet = bp.nodes.Dirichlet(1.0 * np.ones(3))
```

```
diet = bp.nodes.Categorical(p_diet, plates=(N, 1))
```

```
diet.observe(data[:, 3])
```

```
p_lifestyle = bp.nodes.Dirichlet(1.0 * np.ones(4))
```

```
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N, 1))
```

```
lifestyle.observe(data[:, 4])
```



```
p-cholesterol = bp.nodes.Dirichlet(1.0 * np.ones(3))
cholesterol = bp.nodes.Categorical(p-cholesterol, plates=(N,))
cholesterol.observe(data[:, 5])
```

```
p-heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=
(5, 2, 2, 3, 4, 3))
```

```
heartdisease = bp.nodes.MultiMixture([age, gender, family
history, diet, lifestyle, cholesterol], bp.nodes.Categorical,
p-heartdisease)
```

```
heartdisease.observe(data[:, 6]) p-heartdisease.update()
```

m = 0

while m == 0:

```
print("\n")
```

```
res = bp.nodes.MultiMixture([int(input('Enter Age: '
+ str(ageEnum))), int(input('Enter Gender: ' + str(genderEnum))),
int(input('Enter Family History: ' + str(familyHistoryEnum))),
int(input('Enter DietEnum: ' + str(dietEnum))), int(input(
'Enter Lifestyle: ' + str(lifestyleEnum))), int(input('Enter
Cholesterol: ' + str(cholesterolEnum))], bp.nodes.
Categorical, p-heartdisease).get_moments()[0][heartDisease
Enum['Yes']]
```

```
print('Probability (HeartDisease) = ' + str(res))
```

```
m = int(input("Enter for continue : 0, Exit : 1"))
```

DATASETS:

1	SuperSeniorCitizen	Male	Yes	Medium	Sedetary	High	Yes
2	SuperSeniorCitizen	Female	Yes	Medium	Sedetary	High	Yes
3	SeniorCitizen	Male	No	High	Moderate	BorderLine	Yes
4	Teen	Male	Yes	Medium	Sedetary	Normal	No
5	Youth	Female	Yes	High	Athlete	Normal	No
6	MiddleAged	Male	Yes	Medium	Active	High	Yes
7	Teen	Male	Yes	High	Moderate	High	Yes
8	SuperSeniorCitizen	Male	Yes	Medium	Sedetary	High	Yes
9	Youth	Female	Yes	High	Athlete	Normal	No
10	SeniorCitizen	Female	No	High	Athlete	Normal	Yes
11	Teen	Female	No	Medium	Moderate	High	Yes
12	Teen	Male	Yes	Medium	Sedetary	Normal	No
13	MiddleAged	Female	No	High	Athlete	High	No
14	MiddleAged	Male	Yes	Medium	Active	High	Yes
15	Youth	Female	Yes	High	Athlete	BorderLine	No
16	SuperSeniorCitizen	Male	Yes	High	Athlete	Normal	Yes
17	SeniorCitizen	Female	No	Medium	Moderate	BorderLine	Yes
18	Youth	Female	Yes	Medium	Athlete	BorderLine	No
19	Teen	Male	Yes	Medium	Sedetary	Normal	No

OUTPUT:

Output:

Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4} 1

Enter Gender: {'Male': 0, 'Female': 1} 1

Enter FamilyHistory: {'Yes': 0, 'No': 1} 1

Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2} 2

Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3} 2

Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2} 1

Probability(HeartDisease) = 0.5

Enter for Continue 0, Exit : 1 0

Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4} 0

Enter Gender: {'Male': 0, 'Female': 1} 0

Enter FamilyHistory: {'Yes': 0, 'No': 1} 0

Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2} 0

Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3} 3

Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2} 0

Probability(HeartDisease) = 0.5

Enter for Continue 0, Exit : 1

OBSERVATION	10
RECORD	10
VIVA	10
TOTAL	30
SIGNATURE	S. Ravi

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.

EXPECTATION-MAXIMIZATION (EM) ALGORITHM:

Step 1: An initial guess is made for the model's parameters and a probability distribution is created. This is sometimes called the "E-step" for the "Expected" distribution.

Step 2: Newly observed data is fed into the model

Step 3: The probability distribution from the E-step is drawn to include the new data. This is sometimes called the "M-step".

Step 4: Steps 2 through 4 are repeated until stability.

PROGRAM:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
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=(14,14))
colormap = np.array(['red', 'lime', 'black'])

plt.subplot(2,2,1)
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')
```

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(x)
xsa = scaler.transform(x)
xs = pd.DataFrame(xsa, columns = x.columns)
```

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components = 3)
gmm.fit(xs)
gmm-y = gmm.predict(xs)
```

```
plt.subplot(2, 2, 3)
plt.scatter(X.Petal.Length, X.Petal.Width, c = color map
            [gmm-y], s = 40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

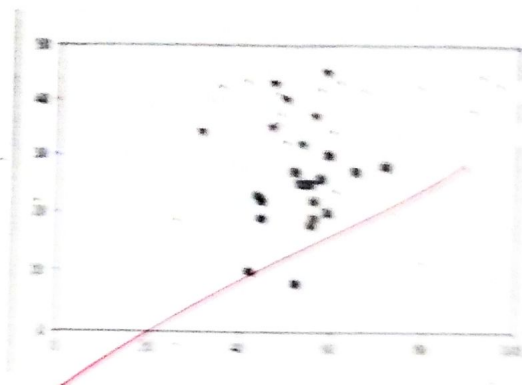
OBSERVATION	10
RECORD	10
VIVA	10
TOTAL	30
SIGNATURE	S. Ray

DATASET

1	total_bill	tip	sex	smoker	day	time	size	26	19.82	3.18	Male	No	Sat	Dinner	2
2	16.99	1.01	Female	No	Sun	Dinner	2	27	17.81	2.34	Male	No	Sat	Dinner	2
3	10.34	1.66	Male	No	Sun	Dinner	3	28	13.37	2	Male	No	Sat	Dinner	4
4	21.01	3.5	Male	No	Sun	Dinner	3	29	12.69	2	Male	No	Sat	Dinner	2
5	23.68	3.31	Male	No	Sun	Dinner	2	30	21.7	4.3	Male	No	Sat	Dinner	2
6	24.59	3.61	Female	No	Sun	Dinner	4	31	19.65	3	Female	No	Sat	Dinner	2
7	25.29	4.71	Male	No	Sun	Dinner	4	32	9.55	1.45	Male	No	Sat	Dinner	2
8	8.77	2	Male	No	Sun	Dinner	2	33	18.35	2.5	Male	No	Sat	Dinner	2
9	26.88	3.12	Male	No	Sun	Dinner	4	34	15.06	3	Female	No	Sat	Dinner	4
10	15.04	1.96	Male	No	Sun	Dinner	2	35	20.69	2.45	Female	No	Sat	Dinner	2
11	14.78	3.23	Male	No	Sun	Dinner	2	36	17.78	3.27	Male	No	Sat	Dinner	4
12	10.27	1.71	Male	No	Sun	Dinner	2	37	24.06	3.6	Male	No	Sat	Dinner	2
13	35.26	5	Female	No	Sun	Dinner	4	38	16.31	2	Male	No	Sat	Dinner	3
14	15.42	1.57	Male	No	Sun	Dinner	2	39	16.93	3.07	Female	No	Sat	Dinner	3
15	18.43	3	Male	No	Sun	Dinner	4	40	18.69	2.31	Male	No	Sat	Dinner	3
16	14.83	3.02	Female	No	Sun	Dinner	2	41	31.27	5	Male	No	Sat	Dinner	3
17	21.58	3.92	Male	No	Sun	Dinner	2	42	16.04	2.24	Male	No	Sat	Dinner	3
18	10.33	1.67	Female	No	Sun	Dinner	3	43	17.46	2.54	Male	No	Sun	Dinner	3
19	16.29	3.71	Male	No	Sun	Dinner	3	44	13.94	3.06	Male	No	Sun	Dinner	2
20	16.97	3.5	Female	No	Sun	Dinner	3	45	9.68	1.32	Male	No	Sun	Dinner	2
21	20.65	3.35	Male	No	Sat	Dinner	3	46	30.4	5.6	Male	No	Sun	Dinner	2
22	17.92	4.08	Male	No	Sat	Dinner	2	47	18.29	3	Male	No	Sun	Dinner	4
23	20.29	2.75	Female	No	Sat	Dinner	2	48	22.23	5	Male	No	Sun	Dinner	2
24	15.77	2.23	Female	No	Sat	Dinner	2	49	32.4	6	Male	No	Sun	Dinner	2
25	39.42	7.58	Male	No	Sat	Dinner	4	50	28.55	2.05	Male	No	Sun	Dinner	4

OUTPUT:

Output



AIM:-

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.

ALGORITHM:

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

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 corresponds to an already classified data point.
4. Return The majority label among S .

PROGRAM:-

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

```
from sklearn.datasets import load-iris
data = load-iris()
```

```
df = pd.DataFrame(data.data, columns=data.feature_names)
df['class'] = data.target_names[data.target]
df.head()
```

```
x = df.iloc[:, :-1].values
y = df['class'].values
print(x[:5])
print(y[:5])
```

```
from sklearn.model-selection import train-test-split
x_train, x_test, y_train, y_test = train-test-split(x, y,
                                                    test-size=0.2)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n-neighbors=5)
knn_classifier.fit(x_train, y_train)
predictions = knn_classifier.predict(x_test)
print(predictions)
```



```

from sklearn.metrics import accuracy_score, confusion-
matrix
print("Training accuracy Score is :", accuracy_score(y_train,
knn_classifier.predict(x_train)))
print("Testing accuracy Score is :", accuracy_score(
y_test, knn_classifier.predict(x_test)))
print("Training Confusion Matrix is : \n", confusion-
matrix(y_train, knn_classifier.predict(x_train)))
print("Testing Confusion Matrix is : \n", confusion-
matrix(y_test, knn_classifier.predict(x_test)))
  
```

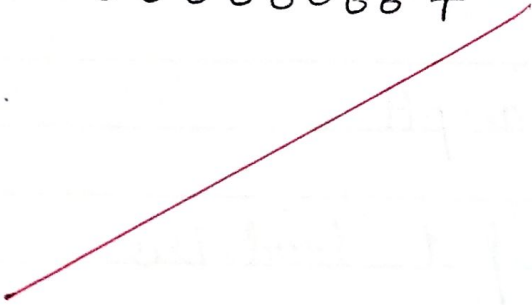
OBSERVATION	10
RECORD	10
VIVA	10
TOTAL	30
SIGNATURE	S. Rajk

DATASET1

51	18.04	3 Male	No	Sun	Dinner	2	76	14.73	2.2 Female	No	Sat	Dinner	3
52	12.54	2.5 Male	No	Sun	Dinner	2	77	10.51	1.25 Male	No	Sat	Dinner	3
53	10.29	2.6 Female	No	Sun	Dinner	2	78	17.92	3.08 Male	Yes	Sat	Dinner	3
54	34.81	5.2 Female	No	Sun	Dinner	4	79	27.2	4 Male	No	Thur	Lunch	3
55	9.94	1.56 Male	No	Sun	Dinner	2	80	22.76	3 Male	No	Thur	Lunch	4
56	25.56	4.34 Male	No	Sun	Dinner	4	81	17.29	2.71 Male	No	Thur	Lunch	3
57	19.49	3.51 Male	No	Sun	Dinner	2	82	19.44	3 Male	Yes	Thur	Lunch	3
58	38.01	3 Male	Yes	Sat	Dinner	4	83	16.66	3.4 Male	No	Thur	Lunch	3
59	26.41	1.5 Female	No	Sat	Dinner	2	84	10.07	1.83 Female	No	Thur	Lunch	3
60	11.24	1.76 Male	Yes	Sat	Dinner	2	85	32.68	5 Male	Yes	Thur	Lunch	1
61	48.27	6.73 Male	No	Sat	Dinner	4	86	15.98	2.03 Male	No	Thur	Lunch	3
62	20.29	3.21 Male	Yes	Sat	Dinner	2	87	34.83	5.17 Female	No	Thur	Lunch	3
63	13.81	2 Male	Yes	Sat	Dinner	2	88	13.03	2 Male	No	Thur	Lunch	4
64	11.02	1.98 Male	Yes	Sat	Dinner	2	89	18.28	4 Male	No	Thur	Lunch	3
65	18.29	3.76 Male	Yes	Sat	Dinner	4	90	24.71	5.85 Male	No	Thur	Lunch	3
66	17.59	2.64 Male	No	Sat	Dinner	3	91	21.16	3 Male	No	Thur	Lunch	3
67	20.08	3.15 Male	No	Sat	Dinner	3	92	28.97	3 Male	Yes	Fri	Dinner	3
68	16.45	2.47 Female	No	Sat	Dinner	2	93	22.49	3.5 Male	No	Fri	Dinner	3
69	3.07	1 Female	Yes	Sat	Dinner	1	94	5.75	1 Female	Yes	Fri	Dinner	3
70	20.23	2.01 Male	No	Sat	Dinner	2	95	16.32	4.3 Female	Yes	Fri	Dinner	3
71	15.01	2.09 Male	Yes	Sat	Dinner	2	96	22.75	3.25 Female	No	Fri	Dinner	3
72	12.02	1.97 Male	No	Sat	Dinner	2	97	40.17	4.73 Male	Yes	Fri	Dinner	4
73	17.07	3 Female	No	Sat	Dinner	3	98	27.28	4 Male	Yes	Fri	Dinner	3
74	26.86	3.14 Female	Yes	Sat	Dinner	2	99	12.03	1.5 Male	Yes	Fri	Dinner	3
75	25.28	5 Female	Yes	Sat	Dinner	2	100	21.01	3 Male	Yes	Fri	Dinner	3

OUTPUT:

0.9666666666667



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.

Let us consider the case of locally weighted regression in which the target function f is approximated near x , using a linear function of the form.

1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) = \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training examples by some decreasing function K of its distance from x_q :

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

3. Combine 1 and 2 :

$$E_3(x_q) = \frac{1}{2} \sum_{x \in K \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

PROGRAM:

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

```
def kernel(point, xmat, k):
    m, n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
```

```
    for j in range(m):
        diff = point - x[j]
        weights[j, j] = np.exp(diff * diff.T / (-2.0 * k * k))
    return weights
```

```
def localWeight(point, xmat, ymat, k):
    wei = kernel(point, xmat, k)
    w = (x.T * (wei * x)) * 1 * (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,
                                     xmat, k)
return ypred.
```

```
def graphPlot(X, ypred):
    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();
```

```
data = pd.read_csv('10 data - tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)
mbill = np.array(bill)
mtip = np.array(tip)
m = np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T, mbill.T))
ypred = localWeightRegression(X, mtip, 8)
graphPlot(X, ypred)
```

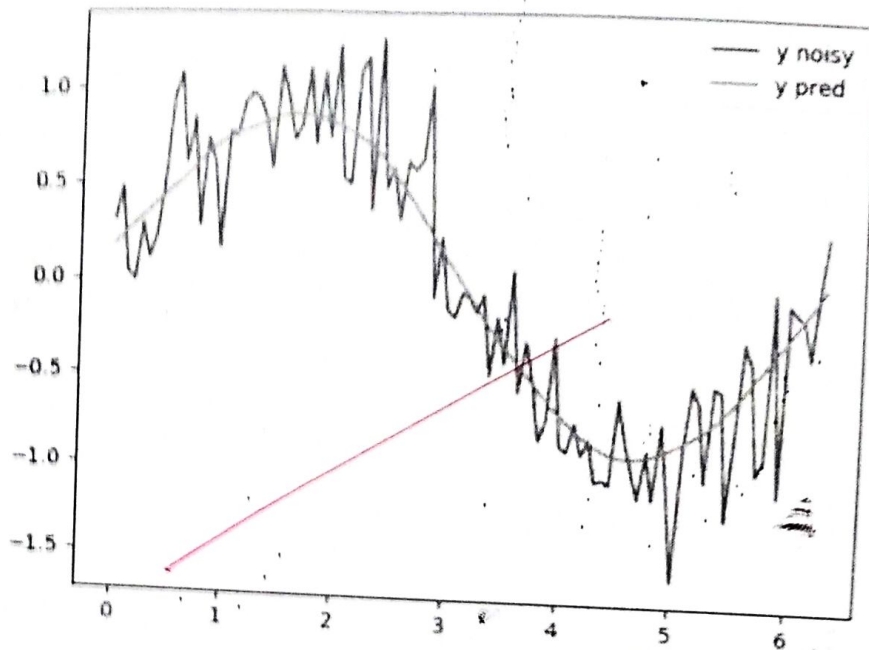
848 15/01/2021

DATASET 1

001	12.46	1.5	Male	No	Fri	Dinner	2
002	11.35	2.5	Female	Yes	Fri	Dinner	2
003	15.38	3	Female	Yes	Fri	Dinner	2
004	44.3	2.5	Female	Yes	Sat	Dinner	3
005	44.44	3.48	Female	Yes	Sat	Dinner	2
006	20.04	4.08	Female	No	Sat	Dinner	2
007	15.38	1.84	Male	Yes	Sat	Dinner	2
008	20.49	4.06	Male	Yes	Sat	Dinner	2
009	25.21	4.29	Male	Yes	Sat	Dinner	2
010	18.24	3.76	Male	No	Sat	Dinner	2
011	14.31	4	Female	Yes	Sat	Dinner	2
012	14	3	Male	No	Sat	Dinner	2
013	7.25	1	Female	No	Sat	Dinner	2
014	38.07	4	Male	No	Sun	Dinner	1
015	23.95	2.55	Male	No	Sun	Dinner	3
016	25.71	4	Female	No	Sun	Dinner	2
017	17.31	3.5	Female	No	Sun	Dinner	3
018	29.93	8.07	Male	No	Sun	Dinner	2
019	10.65	1.5	Female	No	Thur	Lunch	4
020	12.43	1.8	Female	No	Thur	Lunch	2
021	24.08	2.92	Female	No	Thur	Lunch	2
022	11.69	2.31	Male	No	Thur	Lunch	4
023	13.42	1.88	Female	No	Thur	Lunch	2
024	14.26	2.5	Male	No	Thur	Lunch	2
025	15.95	3	Male	No	Thur	Lunch	2

OUTPUT:

Output



OBSERVATION	10
RECORD	10
VIVA	10
TOTAL	30
SIGNATURE	S. Rayll



KSIT
K.S. INSTITUTE OF TECHNOLOGY

Kammavari Sangham (R)-1952

K.S.GROUP OF INSTITUTIONS

K.S. INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Affiliated to VTU)

KANAKAPURA ROAD, BANGALORE - 560 062

IA, PRACTICAL TEST /PROJECT VIVA- VOCE

Course/Branch: BE / CSE

1	K	S	1	7	C	S	0	1	1
University Seat Number									

Subject: MACHINE LEARNING LABORATORY Subject Code:

17CSL76

Total number of supplements tied together : 1 + _____ = _____

Date

15 01 20 21

D D M M Y Y Y Y

(3)

S. Raj

O/p seen

Andrew J

Signature of the Candidate

Entries to be made by the examiners

SCHEME OF AWARDING MARKS

Practical Examination					Project viva -voce				
a. Procedure & write up	:		of Max. Marks		a. Project Work	:	40% of Max .Marks		
b. Conducting the practicals, calculations, graphs, results, etc.,	:		of Max. Marks		b. Presentation	:	30% of Max .Marks		
c. Viva-voce	:		of Max. Marks		c. Viva-voce	:	30% of Max .Marks		

Question Number	Marks awarded			
	a	b	c	Total
1	15	70	15	100
2				
3				
4				

Title of the Project				

Total Max Marks	Marks awarded			
	a	b	c	Total

Total Maximum Marks:	<u>100</u>	Total Marks obtained:	<u>100</u>
Marks awarded in words:	<u>one</u>		<u>zero zero</u>

Total Marks obtained:	
Marks awarded in words:	

Examiner 1

Examiner 2

Name

Dr. Rekha B Venkatapur.

Raglavendrarao S

Signature with Date :

Dr. Rekha B Venkatapur 15/1/21

S. Raj 15/01/2021

③ Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

→

```
import csv
import math
import random
```

```
def majorClass(attribute, data, target):
```

```
    freq = { }
```

```
    index = attribute.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 key major
```



```
def entropy (attributes, data, targetAttr):
```

```
    freq = {}
```

```
    dataEntropy = 0.0
```

```
    i = 0
```

```
    for entry in attribute:
```

```
        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
```

```
def info-gain (attributes, data, attr, targetAttr):
```

```
    freq = {}
```

```
    i = index attribute.index(attr)
```

```
    subsetEntropy = 0.0
```

```
    for entry in data:
```

```
        if 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(attribute,  
                                         dataSubset, targetAttr))
```

```
return (entropy(attribute, data, targetAttr) - subsetEntropy)
```

```
def attr_choose(data, attributes, target):
```

```
    best = attributes[0]
```

```
    maxGain = 0
```

```
    for attr in attributes:
```

```
        newGain = info_gain(attribute, data, attr, target)
```

```
        if newGain > maxGain:
```

```
            maxGain = newGain
```

```
            best = attr
```

```
    return best
```

```
def get_values(data, attribute, attr):
```

```
    values = []  
    → index = attribute.index(attr)
```

```
    for entry in data:
```

```
        values = entry[index]
```

```
    if (values not in entry[index]):
```

```
        values.append(entry[index])
```

```
    return values
```




Kammavari Sangham (R)-1952
K.S.GROUP OF INSTITUTIONS
K.S. INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Affiliated to VTU)
KANAKAPURA ROAD, BANGALORE - 560 062

USN No.:

1	K	S	I	T	C	S	O	I
---	---	---	---	---	---	---	---	---

```
def get-data (data, attribute, best, val):  
    new-data = []  
    index = attribute.index (best)  
    for entry in data :  
        if (entry[index] == val)  
            newEntry = []  
            if (i != len(entry) - 1):  
                for i in range (0, len(entry)): :  
                    newEntry.append (entry[i])  
                new-data.append (newEntry)  
            new-data.remove ([])  
    return new-data.
```

```
def build-tree (data, attribute, target):  
    data = data[:]  
    vals = [record [attribute.index (target) for record in data]]  
    default = majorClass (attribute, data, target)  
    if not data or (len(attribute) - 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, attribute, best):
```

```
    new_data = get_data(data, attribute, best, val)
```

```
    newAttr = attribute[:]
```

```
    newAttr.remove(best)
```

```
    subtree = build_tree(new_data, newAttr, target)
```

```
    tree[best][val] = subtree
```

```
    return tree
```

```
def create_decision_tree():
```

```
    data = []
```

```
    with open("weather.csv") as tsv:
```

```
        for lines in tsv.reader(tsv):
```

```
            data.append(tuple(lines))
```

```
    print("Number of records: ", len(data))
```

```
    attributes = ['outlook', 'temperature', 'humidity', 'wind',  
                  'Play']
```

```
    target = attribute[-1]
```



```
acc = []
```

```
training-set = [x for i, x in enumerate(data)]
```

```
tree = build-tree (training-set, attributes, target)
```

```
print (tree)
```

```
results = []
```

```
test-set = ['rainy', 'mild', 'high', 'strong']
```

```
for entry in test-set:
```

```
    tempDict = tree.copy()
```

```
    result = ""
```

```
    while (isinstance(tempDict, dict)):
```

```
        child = []
```

```
        nodeVal = next(iter(tempDict))
```

```
        child = tempDict[next(iter(tempDict))].keys()
```

```
        tempDict = tempDict[next(iter(tempDict))]
```

```
        value = entry[index]
```

```
        index = attribute.index(entry[index])
```

```
        if (value in tempDict.keys()):
```

```
            result = tempDict[value]
```

```
            tempDict = tempDict[value]
```

```
        else:
```

```
            result = "Null"
```

```
            break
```

if (result != null)

results.append(result == entry[-1])

return results

if __name__ == "__main__":

execute_decision_tree()

g.s. ✓

K.S. INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGG.
VII SEM A SEC ML LAB INTERNALS MARKS

SL. NO	USN	NAME OF THE STUDENT	OBSERVATION & RECORD (30)	TEST (10)	TOTAL (40)	SIGNATURE
1	1KS17CS001	AAFREEN HUSSAIN	30	9	39	Aafreen
2	1KS17CS002	ABHISHEK GOWDA.M.V	30	8	38	Abhishek
3	1KS17CS003	AKSHATHA RAMESH	30	10	40	Akshatha
4	1KS17CS004	AKSHITHA.B.S	30	8	38	Akshitha
5	1KS17CS005	AMOGH.R	30	10	40	Amogh
6	1KS17CS006	AMOGHA MANJUNATHA.K	30	9	39	Amoghachem
7	1KS17CS007	AMRUTHA.V.DESHPANDE	30	10	40	Amrutha
8	1KS17CS008	ANOOP.P.S	30	8	38	Anoop
9	1KS17CS010	ANUSHA.A.G	30	8	38	Anusha
10	1KS17CS011	ANUSHREEJ	30	10	40	Anushreej
11	1KS17CS013	ASHISH.K.AMAR	30	10	40	Ashish
12	1KS17CS014	LAKSHMI PRASANNA.B	30	10	40	Lakshmi
13	1KS17CS016	BHAVESH BHANSALI	30	10	40	Bhavesh
14	1KS17CS017	CHAITRA	30	10	40	Chaitra
15	1KS17CS018	CHANDANA.B.R	30	10	40	Chandana.B.R.
16	1KS17CS019	CHENNA KESHAVA.N.T	30	9	39	Chenna NT
17	1KS17CS020	DARSHAN.S	30	10	40	Darshan
18	1KS17CS021	DEEKSHITHA.R	30	09	39	Deekshitha

SL NO	USN	NAME OF THE STUDENT	OBSERVATION & RECORD (30)	TEST (10)	TOTAL (40)	SIGNATURE
19	1KS17CS022	DEEPIKA.S.H	30	10	40	Deepeeka
20	1KS17CS023	DIVYA YASHASWI KANNEY	30	08	38	Divya
21	1KS17CS024	GANESH.G.B	30	10	40	Ganesh
22	1KS17CS025	GANESH MAUDGHALYA.H.G	30	09	39	Ganesh
23	1KS17CS026	GAUTHAM.C.R	30	08	38	Gautham C.R
24	1KS17CS027	H.PRIYANKA	30	08	38	H.Priyanka
25	1KS17CS028	HANUMESH.V.T	30	08	38	Hanumesh
26	1KS17CS029	HARSHITHA.V	30	08	38	Harshitha..
27	1KS17CS030	INDRASENA KALYANAM	30	10	40	Indrasena
28	1KS17CS032	KARAN RAGHUNATH	30	10	40	Karan
29	1KS17CS033	KARTHIK.T.C	30	10	40	Karthik
30	1KS17CS034	KAVITHA.S	30	08	38	Kavitha S
31	1KS17CS035	KEERTHI.N	30	09	39	Keerthi
32	1KS17CS036	KRITHIKA JAGANNATH	30	10	40	Krithika
33	1KS17CS037	LAVANYA.V	30	10	40	Lavanya V
34	1KS17CS038	LOKESH.B.M	30	10	40	Lokesh B.M
35	1KS17CS040	MANJUNATH.A	30	8	38	Manjunath
36	1KS17CS041	MEGHANA.C.V	30	10	40	Meghana C.V
37	1KS17CS042	MEGHANA.G	30	10	40	Megha
38	1KS17CS043	MEGHANA.G.R	30	10	40	Megha G.R

SL. NO	USN	NAME OF THE STUDENT	OBSERVATION & RECORD (30)	TEST (10)	TOTAL (40)	SIGNATURE
39	1KS17CS044	MOUNIKA.M.K.L	30	09	39	Mounika
40	1KS17CS045	NEHA.K	30	10	40	Neha K
41	1KS17CS046	NIKHIL SUBRAMANYA.K	30	09	39	Nikhil
42	1KS17CS047	NIKITHA KATARI	30	08	38	Nikitha
43	1KS17CS048	NISCHITHA.C	30	8	38	Nishitha.C.
44	1KS17CS049	NITISH KUMAR GUPTA	30	10	40	Nitish
45	1KS17CS050	NYDILE.G.R	30	10	40	Nydhile
46	1KS17CS051	P.KISHORE	30	10	40	P.Kishore
47	1KS17CS102	SHRIRAKSHAS.KANAGO	30	09	39	Shritha
48	1KS18CS401	KRUTHIKA.B.M	30	08	38	Kruthika BM
49	1KS16CS042	MEGHANA.H.S	30	07	37	Meghana HS
50	1KS15CS050	LAXMI K V	15 (15)	02 (5)	17 (20)	Laxmi
51	1KS16CS090	SHASHANK KAVUR	15 (15)	02 (5)	17 (20)	Shashank

S. Rajee
[COURSE IN CHARGE]