**K.S. RANGASAMY COLLEGE OF TECHNOLOGY, TIRUCHENGODE-637215**

**(Autonomous Institution)**


# 50 CB 7P1 &- MACHINE LEARNING LABORATORY

**Semester-VII, -Year-IV, Batch-2020-2024**

**BACHELOR OF TECHNOLOGY**

**DEPARTMENT OF COMPUTER SCIENCE AND BUSINESS SYSTEMS**

## K.S. RANGASAMY COLLEGE OF TECHNOLOGY

(An Autonomous institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

**TIRUCHENGODE - 637 215**

**K.S. RANGASAMY COLLEGE OF TECHNOLOGY, TIRUCHENGODE-637215**

**(Autonomous Institution)**


**CERTIFICATE**

| | |
|---|---|
| **Register Number** | **73772027105** |

Certified that this is the bonafide record of work done by **Ganeshbabu K** of the Seventh Semester B.Tech., Computer Science and Business Systems branch during the academic year 2022-2023 in the **50 CB 7P1 & - Machine Learning Laboratory**

**Faculty in-charge** **Head of the Department**

**Program: B. Tech – Computer Science and Business Systems**

## VISION AND MISSION OF THE DEPARTMENT

### VISION

To produce skilled professionals to the dynamic needs of the industry with innovative computer science professionals associate with managerial services

### MISSION

- To promote student's ability through innovative teaching in computer science to compete globally as an engineer
- To inculcate management skills to meet the industry standards and augment human values and life skills to serve the society

### PROGRAMME OUTCOMES (POs)

Engineering Graduates will be able to:

**PO1: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

**PO2: Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**PO3: Design /development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**PO4: Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO5: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations

**PO6: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**PO7: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**PO8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**PO9: Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**PO11: Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one‘s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO12: Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**PROGRAMME SPECIFIC OUTCOMES (PSOs):**

**Engineering Graduates will be able to:**

**PSO1:** Apply analytical and technical skill of computer science to provide justifiable solution for real World Applications

**PSO2:** Analyze various managerial skills and business disciplines to improve the industry growth and development

# CONTENT

| Ex: 01 | Implementation and demonstrate the 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 ai in h

If the constraint ai is satisfied by x

Then do nothing

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

3. Output hypothesis h

**Training Examples:**

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

**Program: -**

```
import csv
a = []
with open('enjoysport.csv', 'r') as csvfile:
 for row in csv.reader(csvfile):
 a.append(row)
 print(a)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)
for i in range(0, len(a)):
 if a[i][num_attribute] == 'yes':
 for j in range(0, num_attribute):
 if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
 hypothesis[j] = a[i][j]
 else:
 hypothesis[j] = '?'
print("\n The hypothesis for the training instance {} is : \n" .format(i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```

**Output:**

The Given Training Data Set

['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']

['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']

['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']

['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

The total number of training instances are: 4

The initial hypothesis is :

['0', '0', '0', '0', '0', '0']

The hypothesis for the training instance 1 is:

['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

The hypothesis for the training instance 2 is:

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

The hypothesis for the training instance 3 is:

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

The hypothesis for the training instance 4 is:

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

The Maximally specific hypothesis for the training instance is

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

| K.S. Rangasamy College of Technology | | |
|---|---|---|
| **Mark Allocation** | | |
| **Details** | **Mark Allotted** | **Mark Awarded** |
| Preparation | 40 | |
| Compilation and Execution | 30 | |
| Result | 10 | |
| Viva – Voce | 10 | |
| Record | 10 | |
| Total | 100 | |

**Result: -**

| Ex: 02 | Implementation of Candidate-Elimination algorithm |
|---|---|

**Aim: -**

Implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training example

**Candidate - Learning Algorithm: -**

Initialize G to the set of maximally general hypotheses in H

Initialize S to the set of maximally specific hypotheses in H

For each training example d, do

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 any hypothesis that is more general than another hypothesis in S

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

**Training Examples:**

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

## Program: -

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)
def learn(concepts, target):
 specific_h = concepts[0].copy()
 print("initialization of specific_h and general_h")
 print(specific_h)
 general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
 print(general_h)
 for i, h in enumerate(concepts):
 if target[i] == "yes":
 for x in range(len(specific_h)):
 if h[x]!= specific_h[x]:
 specific_h[x] ='?'
```

```
 general_h[x][x] ='?'
 print(specific_h)
 print(specific_h)
 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] = '?'
 print(" steps of Candidate Elimination Algorithm",i+1)
 print(specific_h)
 print(general_h)
 indices = [i for i, val in enumerate(general_h) if val ==
['?', '?', '?', '?', '?', '?']]
 for i in indices:
 general_h.remove(['?', '?', '?', '?', '?', '?'])
 return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

**Output:**

```
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'],
['?', 'warm', '?', '?', '?', '?']]
```

| K.S. Rangasamy College of Technology | | |
|---|---|---|
| **Mark Allocation** | | |
| **Details** | **Mark Allotted** | **Mark Awarded** |
| Preparation | 40 | |
| Compilation and Execution | 30 | |
| Result | 10 | |
| Viva – Voce | 10 | |
| Record | 10 | |
| Total | 100 | |

**Result: -**

| Ex: 03 | Implementation of Decision Tree |
|---|---|

**Aim: -**

Implement and demonstrate the Decision algorithm

**Algorithm: -**

- Create a Root node for the 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 ← 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, Targe_tattribute, Attributes – {A}))
- End
- Return Root

**Entropy: -**

Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

Were,

p+ is the proportion of positive examples in S

p- is the proportion of negative examples in S

**Information Gain: -**

- Information gain, is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

**Training DataSets:-**

| Day | *Outlook* | *Temperature* | *Humidity* | *Wind* | *PlayTennis* |
|---|---|---|---|---|---|
| **D1** | Sunny | Hot | High | Weak | No |
| **D2** | Sunny | Hot | High | Strong | No |
| **D3** | Overcast | Hot | High | Weak | Yes |
| **D4** | Rain | Mild | High | Weak | Yes |
| **D5** | Rain | Cool | Normal | Weak | Yes |
| **D6** | Rain | Cool | Normal | Strong | No |
| **D7** | Overcast | Cool | Normal | Strong | Yes |
| **D8** | Sunny | Mild | High | Weak | No |
| **D9** | Sunny | Cool | Normal | Weak | Yes |
| **D10** | Rain | Mild | Normal | Weak | Yes |
| **D11** | Sunny | Mild | Normal | Strong | Yes |
| **D12** | Overcast | Mild | High | Strong | Yes |
| **D13** | Overcast | Hot | Normal | Weak | Yes |
| **D14** | Rain | Mild | High | Strong | No |

**Test DataSet:-**

| Day | *Outlook* | *Temperature* | *Humidity* | *Wind* |
|---|---|---|---|---|
| **T1** | Rain | Cool | Normal | Strong |
| **T2** | Sunny | Mild | Normal | Strong |

**Program: -**

```
import math
 import csv
def load_csv(filename):
lines=csv.reader(open(filename,"r"));
dataset = list(lines)
headers = dataset.pop(0)
return dataset,headers
class Node:
def _init_(self,attribute):
self.attribute=attribute
self.children=[]
self.answer=""
def subtables(data,col,delete):
 dic={}
coldata=[row[col] for row in data]
attr=list(set(coldata))
counts=[0]*len(attr)
r=len(data)
 c=len(data[0])
for x in range(len(attr)):
for y in range(r):
if data[y][col]==attr[x]:
 counts[x]+=1
for x in range(len(attr)):
dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
pos=0
 for y in range(r):
 if data[y][col]==attr[x]:
if delete:
del data[y][col]
dic[attr[x]][pos]=data[y]
pos+=1
 return attr,dic
def entropy (S):
attr-list (set (S))
if len(attr)==1:
return 0
counts=[0,0]
for i in range (2):
counts[i]=sum([1 for x in S if attr[i]==x])/(len (S)*1.0)
sums=0
for cnt in counts:
```

```
sums+=-1*cnt*math.log(cnt,2)
return sums
def compute_gain (data, col):
attr, dic subtables (data, col, delete=False) =
total_size=len (data)
entropies=[0]*len (attr)
ratio=[0]*len (attr)
total_entropy-entropy ( [row[-1] for row in data])
for x in range (len (attr)):
ratio [x]=len (dic [attr[x]])/(total_size*1.0) entropies [x]=entropy ( [row[-1] for row in
dic[attr[x]11)
return total entropy
total entropy--ratio [x] *entropies [x]
def build_tree (data, features):
lastcol=[row[-1] for row in data]
if (len (set (lastcol)))==1:
node-Node ("")
node.answer-lastcol [0]
return node
n-len (data[0])-1
gains = [0] *n
for col in range(n):
gains [col]=compute_gain (data, col)
split-gains.index (max (gains))
node=Node (features [split))
fea = features[:split)+features [split+1:]
attr,dic subtables (data, split, delete=True)
for x in range (len (attr)): child-build_tree (dic[attr[x]], fea)
node.children.append((attr[x], child))
return node
def print tree (node, level):
if node.answer!="":
print (" "level, node.answer)
return
print(" "level, node.attribute)
for value,n in node.children:
print(""* (level+1), value)
print_tree (n, level+2)
def classify (node, x_test, features): if node.answer!="":
print (node, answer) return
pos features.index (node.attribute)
for value, n in node.children:
if x test [pos]==value:
```

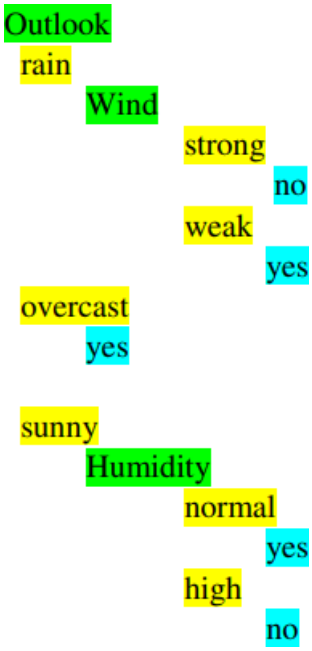
classify (n,x_test, features)

**Main program**

dataset, features-load_csv("data3.csv") nodel-build_tree (dataset, features)
print("The decision tree for the dataset using 103 algorithm18")
print tree (node1,0)
testdata, features-load_csv("data_test.csv") for xtest in testdata:
print("The test instance:",xtest) print ("The label for test instance:", end="
classify (nodel, xtest, features)

**Output: -**

The Decision tree for the datasaet using ID3 algorithms is




The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance: no

The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance: yes

| K.S. Rangasamy College of Technology | | |
|---|---|---|
| **Mark Allocation** | | |
| **Details** | **Mark Allotted** | **Mark Awarded** |
| Preparation | 40 | |
| Compilation and Execution | 30 | |
| Result | 10 | |
| Viva – Voce | 10 | |
| Record | 10 | |
| Total | 100 | |

**Result: -**

| Ex: 04 | Implementation of Back Propagation |
|---|---|

**Aim: -**
To Implment the Back Propagation

**Algorithm: -**
Each training example is a pair of the form $(x, \vec{t})$, where $(x)$ is the vector of network input values, $(t)$ and is the vector of target network output values.
η is the learning rate (e.g., .05). ni, is the number of network inputs, nhidden the number of units in the hidden layer, and nout the number of output units.
The input from unit i into unit j is denoted xji, and the weight from unit i to unit j is denoted wji

- Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
- For each $(\vec{x}, t)$, in training examples, Do

*Propagate the input forward through the network:*

1. Input the instance $\vec{x}$, to the network and compute the output $o_u$ of every unit u in the network.

*Propagate the errors backward through the network:*

2. For each network output unit k, calculate its error term $\delta_k$

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit *h*, calculate its error term $\delta_h$

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight $w_{ji}$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

**Training Example:-**

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

Normalize the input

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

## 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=5000 #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)

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

# dotproduct of nextlayererror and currentlayerop
wout += hlayer_act.T.dot(d_output) *lr
 wh += X.T.dot(d_hiddenlayer) *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:**
[[0.89726759]
[0.87196896]
[0.9000671]]

| K.S. Rangasamy College of Technology | | |
|---|---|---|
| **Mark Allocation** | | |
| **Details** | **Mark Allotted** | **Mark Awarded** |
| Preparation | 40 | |
| Compilation and Execution | 30 | |
| Result | 10 | |
| Viva – Voce | 10 | |
| Record | 10 | |
| Total | 100 | |

**Result: -**

| Ex: 05 | Implementation of Naïve Bayes |
|---|---|

**Aim: -**

To Implement the Naïve Bayes Algorithm

**Algorithm: -**

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Were,
P(h|D) is the probability of hypothesis h given the data D. This is called the posterior probability.
P(D|h) is the probability of data d given that the hypothesis h was true.
P(h) is the probability of hypothesis h being true. This is called the prior probability of h.
P(D) is the probability of the data. This is called the prior probability of D
After calculating the posterior probability for a number of different hypotheses h, and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such
maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis.
Bayes theorem to calculate the posterior probability of each candidate hypothesis is hMAP is a
MAP hypothesis provided

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D|h)P(h)$$

(Ignoring P(D) since it is a constant)

**Sample Example: -**

| Examples | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | Diabetic Pedigree Function | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 2 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 3 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 4 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 5 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| 6 | 5 | 116 | 74 | 0 | 0 | 25.6 | 0.201 | 30 | 0 |
| 7 | 3 | 78 | 50 | 32 | 88 | 31 | 0.248 | 26 | 1 |
| 8 | 10 | 115 | 0 | 0 | 0 | 35.3 | 0.134 | 29 | 0 |
| 9 | 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | 1 |
| 10 | 8 | 125 | 96 | 0 | 0 | 0 | 0.232 | 54 | 1 |

**Program: -**

```
import csv
import random
import math
def loadcsv(filename):
lines = csv.reader(open(filename, "r"));
dataset = list(lines)
for i in range(len(dataset)):

 #converting strings into numbers for processing
dataset[i] = [float(x) for x in dataset[i]]

return dataset
def splitdataset(dataset, splitratio):

 #67% training size
trainsize = int(len(dataset) * splitratio);
trainset = []
copy = list(dataset);
while len(trainset) < trainsize:

#generate indices for the dataset list randomly to pick ele for
training data
```

```
index = random.randrange(len(copy));
trainset.append(copy.pop(index))
return [trainset, copy]
def separatebyclass(dataset):
separated = {} #dictionary of classes 1 and 0

#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
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): #creates a dictionary of classes
summaries = [(mean(attribute), stdev(attribute)) for
attribute in zip(*dataset)];
del summaries[-1] #excluding labels +ve or -ve
return summaries
def summarizebyclass(dataset):
separated = separatebyclass(dataset);

 #print(separated)
summaries = {}
for classvalue, instances in separated.items():

#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
summaries[classvalue] = summarize(instances)

#summarize is used to cal to mean and std
```

```
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 contains the all prob of all class of test data
probabilities = {}
for classvalue, classsummaries in summaries.items():

#class and attribute information as mean and sd
probabilities[classvalue] = 1
for i in range(len(classsummaries)):
mean, stdev = classsummaries[i] #take mean and
sd of every attribute for class 0 and 1 seperaely
x = inputvector[i] #testvector's first attribute
probabilities[classvalue] *=
calculateprobability(x, mean, stdev);#use normal dist
return probabilities
def predict(summaries, inputvector): #training and test data
is passed
probabilities = calculateclassprobabilities(summaries,
inputvector)
bestLabel, bestProb = None, -1
for classvalue, probability in probabilities.items():
#assigns that class which has the highest prob
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 = 'naivedata.csv'
splitratio = 0.67
dataset = loadcsv(filename);
trainingset, testset = splitdataset(dataset, splitratio)
print('Split {0} rows into train={1} and test={2}
rows'.format(len(dataset), len(trainingset), len(testset)))
# prepare model
summaries = summarizebyclass(trainingset);
#print(summaries)
# test model
predictions = getpredictions(summaries, testset) #find the
predictions of test data with the training data
accuracy = getaccuracy(testset, predictions)
print('Accuracy of the classifier is :
{0}%'.format(accuracy))
main()
```

**Output:-**

```
Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is : 71.65354330708661%
```

‘

| K.S. Rangasamy College of Technology | | |
|---|---|---|
| **Mark Allocation** | | |
| **Details** | **Mark Allotted** | **Mark Awarded** |
| Preparation | 40 | |
| Compilation and Execution | 30 | |
| Result | 10 | |
| Viva – Voce | 10 | |
| Record | 10 | |
| Total | 100 | |

**Result: -**

| Ex: 06 | Implementation of k-Nearest Neighbour |
|---|---|

**Aim: -**

To Implement the k-Nearest Neighbour

**K-Nearest Neighbour Algorithm: -**

Training algorithm:

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

Classification algorithm:

- Given a query instance xq to be classified,
- Let x1 . . .xk denote the k instances from training examples that are nearest to xq
- Return

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

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

**Data Set: -**

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

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

**Program: -**

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

iris=datasets.load_iris()

x = iris.data
y = iris.target

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

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

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

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

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

## Output: -

sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 . . . . .
 . . . . .
 [6.2 3.4 5.4 2.3]
 [5.9 3. 5.1 1.8]]

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

**Confusion Matrix**

[[20 0 0]
 [ 0 10 0]
 [ 0 1 14]]

**Accuracy Metrics**

| | Precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 20 |
| 1 | 0.91 | 1.00 | 0.95 | 10 |
| 2 | 1.00 | 0.93 | 0.97 | 15 |
| avg / total | 0.98 | 0.98 | 0.98 | 45 |

| K.S. Rangasamy College of Technology | | |
|---|---|---|
| **Mark Allocation** | | |
| **Details** | **Mark Allotted** | **Mark Awarded** |
| Preparation | 40 | |
| Compilation and Execution | 30 | |
| Result | 10 | |
| Viva – Voce | 10 | |
| Record | 10 | |
| Total | 100 | |

**Result: -**

| Ex: 07 | Implementation of Weighted Regression |
|---|---|

**Aim: -**

Implementation of Weighted Regression

**Weighted Regression Algorithm: -**

**Regression:**

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

a given input vector x.

- y is called the dependent variable.
- x is called the independent variable.

**Loess/Lowess Regression:**

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

**Lowess Algorithm:**

- Locally weighted regression is a very powerful nonparametric model used in statistical

learning.

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

**Algorithm**

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

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

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

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

**Program: -**

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

def local_regression(x0, X, Y, tau):# add bias term
 x0 = np.r_[1, x0] # Add one to avoid the loss in information
 X = np.c_[np.ones(len(X)), X]

 # fit model: normal equations with kernel
 xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W

 beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product


 # predict value
 return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
def radial_kernel(x0, X, tau):
 return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernal Bias Function

n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])
def plot_lwr(tau):
 # prediction through regression
 prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
 plot = figure(plot_width=400, plot_height=400)
```

```
 plot.title.text='tau=%g' % tau
 plot.scatter(X, Y, alpha=.3)
 plot.line(domain, prediction, line_width=2, color='red')
 return plot

show(gridplot([
 [plot_lwr(10.), plot_lwr(1.)],
 [plot_lwr(0.1), plot_lwr(0.01)]]))

/*
Spyder Editor
This is a temporary script file
*/

from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr

def kernel(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np1.shape(xmat)
```

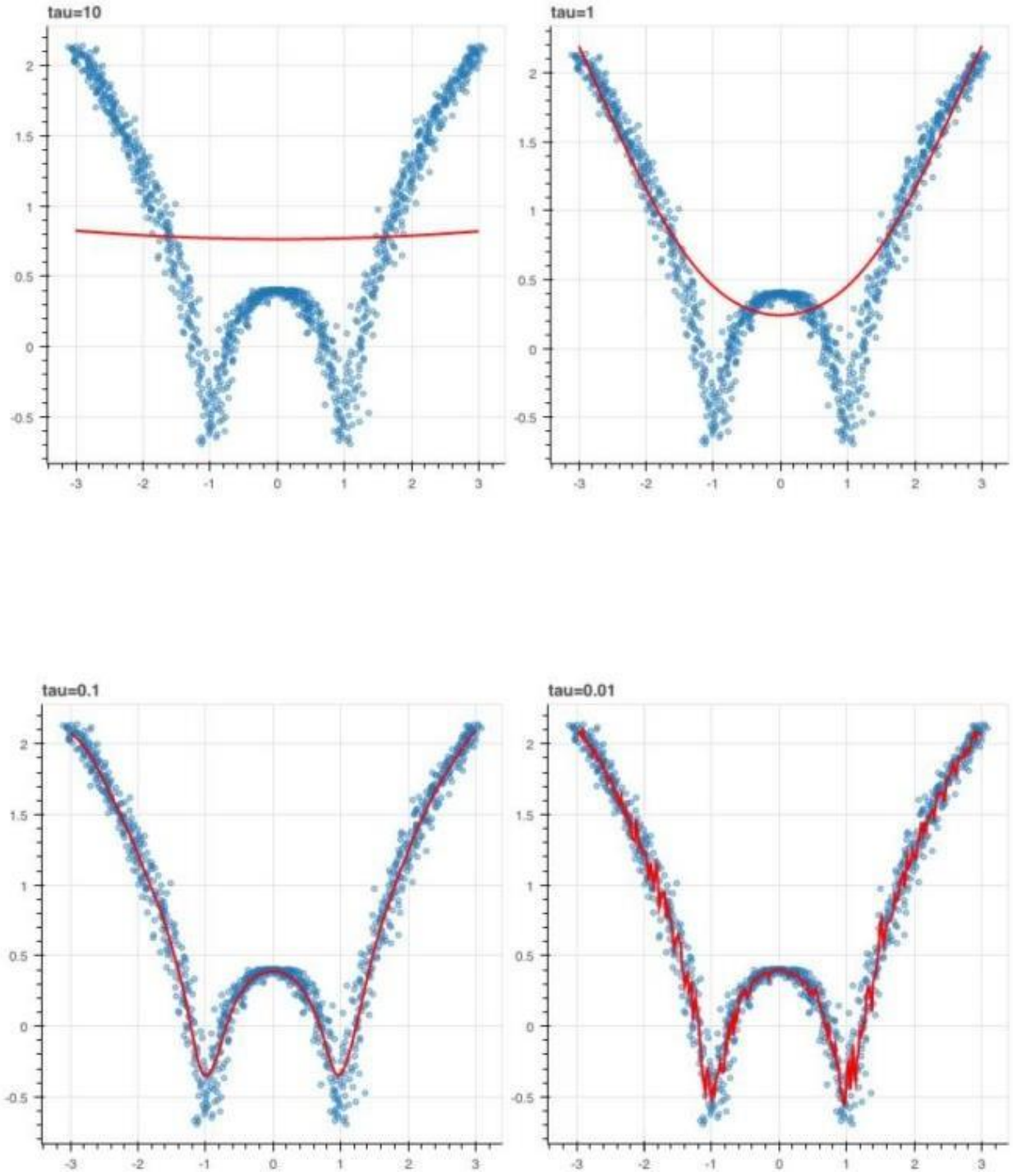
```
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

# load data points
data = pd.read_csv('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
#print(X)
#set k here
ypred = localWeightRegression(X,mtip,0.3)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

**Output: -**



| K.S. Rangasamy College of Technology | | |
|---|---|---|
| **Mark Allocation** | | |
| **Details** | **Mark Allotted** | **Mark Awarded** |
| Preparation | 40 | |
| Compilation and Execution | 30 | |
| Result | 10 | |
| Viva – Voce | 10 | |
| Record | 10 | |
| Total | 100 | |

**Result: -**

| Ex: 08 | Implementation of Support Vector Machine |
|---|---|

Aim:-
The aim of this implementation is to create a basic Support Vector Machine (SVM) classifier using the linear kernel and demonstrate its usage on a sample dataset.

**Algorithm:**

1. Import necessary libraries.
2. Load a sample dataset.
3. Preprocess the data if needed (e.g., scaling).
4. Split the dataset into training and testing sets.
5. Create an SVM model with a linear kernel.
6. Train the SVM model using the training data.
7. Evaluate the model's performance on the testing data.
8. Visualize the decision boundary and support vectors.

**Program:**

```
# Step 1: Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Step 2: Load a sample dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # Using only the first two features for visualization
y = iris.target

# Step 3: Preprocess the data if needed (scaling)
# In this case, we'll skip scaling for simplicity

# Step 4: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Step 5: Create an SVM model with a linear kernel
svm_model = SVC(kernel='linear')

# Step 6: Train the SVM model using the training data
svm_model.fit(X_train, y_train)

# Step 7: Evaluate the model's performance on the testing data
y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Step 8: Visualize the decision boundary and support vectors
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)

# Plot the decision boundary
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# Create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = svm_model.decision_function(xy).reshape(XX.shape)

# Plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])

# Plot support vectors
ax.scatter(svm_model.support_vectors_[:, 0], svm_model.support_vectors_[:, 1], s=100,
linewidth=1, facecolors='none', edgecolors='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('SVM Decision Boundary and Support Vectors')
plt.show()
```

**Input: -**

```
X = [
    [5.1, 3.5],
    [4.9, 3.0],
]

y = [0, 1, 2, ...]
```

**Output: -**

Accuracy: 0.82

| K.S. Rangasamy College of Technology | | |
|---|---|---|
| **Mark Allocation** | | |
| **Details** | **Mark Allotted** | **Mark Awarded** |
| Preparation | 40 | |
| Compilation and Execution | 30 | |
| Result | 10 | |
| Viva – Voce | 10 | |
| Record | 10 | |
| Total | 100 | |

**Result: -**

| Ex: 09 | Implementation of Clustering |
|---|---|

**Aim: -**

The aim of this clustering implementation is to group a set of data points into clusters based on their similarity, without prior knowledge of the number of clusters. We will use the K-means clustering algorithm to achieve this.

**Algorithm: -**

**Algorithm - K-means Clustering:**

1. **Initialization:**
   - Choose the number of clusters, K.
   - Initialize K cluster centroids randomly from the data points.
2. **Assignment Step:**
   - For each data point, calculate its distance to each centroid.
   - Assign the data point to the cluster with the nearest centroid.
3. **Update Step:**
   - Recalculate the centroids for each cluster as the mean of all data points in that cluster.
4. **Convergence:**
   - Repeat the Assignment and Update steps until the centroids no longer change significantly, or a predefined number of iterations is reached.

**Program: -**

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

# Generate synthetic data for demonstration
data, _ = make_blobs(n_samples=300, centers=4, cluster_std=1.0, random_state=42)

# Implement K-means Clustering
def kmeans_clustering(data, n_clusters, max_iterations=100):
    # Randomly initialize cluster centroids
    centroids = data[np.random.choice(len(data), size=n_clusters, replace=False)]

    for _ in range(max_iterations):
        # Assignment Step
        distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
```

```
        labels = np.argmin(distances, axis=1)

        # Update Step
        new_centroids = np.array([data[labels == i].mean(axis=0) for i in range(n_clusters)])

        # Check for convergence
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids

    return labels, centroids

# Number of clusters
n_clusters = 4
# Apply K-means clustering
labels, centroids = kmeans_clustering(data, n_clusters)
# Plot the clustered data
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=100, label='Centroids')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-means Clustering')
plt.legend()
plt.show()
```

Input Data:

```
data = np.array([
    [2.0, 3.0],
    [3.5, 5.0],
    [1.5, 1.8],
    [6.0, 5.5],
    [9.0, 8.0],
    [8.0, 7.5],
    [1.0, 2.5],
    [7.0, 6.0]
])
```

Output: -

```
array([[ 2.33333333,  2.76666667],
       [ 6.        ,  5.5       ],
       [ 8.        ,  7.33333333]])
```

| K.S. Rangasamy College of Technology | | |
|---|---|---|
| **Mark Allocation** | | |
| **Details** | **Mark Allotted** | **Mark Awarded** |
| Preparation | 40 | |
| Compilation and Execution | 30 | |
| Result | 10 | |
| Viva – Voce | 10 | |
| Record | 10 | |
| Total | 100 | |

**Result: -**

| Ex: 10 | Implementation of Support Vector Machine |
|---|---|

**Aim: -**

The aim of this rule-based classification program is to classify incoming emails as either "Spam" or "Not Spam" based on a set of predefined rules.

**Algorithm: -**

1. Define a set of rules based on specific keywords, patterns, and characteristics commonly found in spam emails.
2. Read the subject and content of the incoming email.
3. Apply each rule to the email's subject and content.
4. If any rule is satisfied, classify the email as "Spam." If no rules are satisfied, classify it as "Not Spam."
5. Display the classification result to the user.

**Program: -**

```
class RuleBasedClassifier:
    def __init__(self):
        self.rules = {
            'viagra': 'Spam',
            'lottery': 'Spam',
            'inheritance': 'Spam',
            'urgent': 'Spam',
            'free': 'Spam'
        }

    def classify_email(self, subject, content):
        for keyword, label in self.rules.items():
            if keyword in subject.lower() or keyword in content.lower():
                return label
        return 'Not Spam'

def main():
    classifier = RuleBasedClassifier()

    while True:
        subject = input("Enter email subject: ")
```

```
        content = input("Enter email content: ")

        classification = classifier.classify_email(subject, content)
        print(f"Classification: {classification}")

        again = input("Do you want to classify another email? (y/n): ")
        if again.lower() != 'y':
            break

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

**Input 1: -**

Enter email subject: Congratulations, you've won the lottery!
Enter email content: Claim your prize now and get rich!
Do you want to classify another email? (y/n): y

**Output 1: -**
Classification: Spam

**Input 2: -**
Enter email subject: Hello, just checking in
Enter email content: How have you been? Let's catch up soon.
Do you want to classify another email? (y/n): y

**Output 2: -**
Classification: Not Spam

| K.S. Rangasamy College of Technology | | |
|---|---|---|
| **Mark Allocation** | | |
| **Details** | **Mark Allotted** | **Mark Awarded** |
| Preparation | 40 | |
| Compilation and Execution | 30 | |
| Result | 10 | |
| Viva – Voce | 10 | |
| Record | 10 | |
| Total | 100 | |

**Result: -**