SRI BALAJI CHOCKALINGAM ENGINEERING COLLEGE

A.C.S NAGAR(IRUMBEDU), ARNI,

T.V.MALAI DT.-632 317.

Department

Of

Information Technology

CS3491-ARTIFICIAL INTELLIGENT & MACHINE LEARNING

SRI BALAJI CHOCKALINGAM ENGINEERING COLLEGE

A.C.S NAGAR(IRUMBEDU), ARNI, T.V.MALAI DT.-632 317.

Department

of

Information Technology

BONAFIDE CERTIFICATE

Certified that this is a bonafide record of work done by___________________Of Second Year / IV Semester ***B.Tech Information Technology*** *in the Anna University Practical Examination during the year* ***2022 - 2023*** *in* ***CS3491-ARTIFICIAL INTELLIGENT & MACHINE LEARNING***

Register No. :

Staff In-Charge   Head of the Department

Submitted for Practical Examination held on

Internal Examiner   External Examiner

EX NO: 1A

DATE:

IMPLEMENTATION OF UNINFORMED SEARCH ALGORITHMS(BFS,DFS)

AIM:

To write a python program for Breadth-First Search

Algorithm:

Step1: Start the program.

Step2: Declaring the variables.

Step3: Declaring the function.

Step4: Call the function and perform the operations within loop statements.

Step5: Using the print() the output will be generated.

Step6: Stop the program.

PROGRAM:

```
graph = {'5' : ['3','7'],'3' : ['2', '4'],'7' : ['8'],'2' : [],'4' : ['8'],'8' : [] }
visited = []
queue = []
def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)
  while queue:
    m = queue.pop(0)
    print (m, end = " ")
    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)
```

```
print("Following is the Breadth-First Search: ")
bfs(visited, graph, '5')
```

OUTPUT:

Following is the Breadth-First Search:

5 3 7 2 4 8

EX NO: 1B

DATE:

IMPLEMENTATION OF UNINFORMED SEARCH ALGORITHMS(BFS,DFS)

AIM:

To write a python program for Depth-First Search

Algorithm:

Step1: Start the program.

Step2: Declaring the variables.

Step3: Declaring the function.

Step4: Call the function and perform the operations within loop statements.

Step5: Using the print() the output will be generated.

Step6: Stop the program.

PROGRAM:

```
graph = {'5' : ['3','7'],'3' : ['2', '4'],'7' : ['8'],'2' : [],'4' : ['8'],'8' : []}

visited = set()

def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

print("Following is the Depth-First Search:")
dfs(visited, graph, '5')
```

OUTPUT:

Following is the Depth-First Search:

5

3

2

4

8

7

EX NO: 2A

DATE:

IMPLEMENTATION OF INFORMED SEARCH ALGORITHMS(A*,BFS)

_AIM:

1. To write a python program for A* SEARCH ALGORITHM

Algorithm:

Step1: Start the program.

Step2: Declaring the variables.

Step3: Declaring the function.

Step4: Call the function and perform the operations within loop statements.

Step5: Using the print() the output will be generated.

Step6: Stop the program.

PROGRAM

```
from collections import deque
class Graph:
    def __init__(self, adjacency_list):
        self.adjacency_list = adjacency_list
    def get_neighbors(self, v):
        return self.adjacency_list[v]
    def h(self, n):
        H = {  'A': 1, 'B': 1,'C': 1, 'D': 1 }
        return H[n]
    def a_star_algorithm(self, start_node, stop_node):
        open_list = set([start_node])
        closed_list = set([])
        g = {}
```

```
g[start_node] = 0
parents = {}
parents[start_node] = start_node
while len(open_list) > 0:
    n = None
    for v in open_list:
        if n == None or g[v] + self.h(v) < g[n] + self.h(n):
            n = v;
    if n == None:
        print('Path does not exist!')
        return None
    if n == stop_node:
        reconst_path = []
        while parents[n] != n:
            reconst_path.append(n)
            n = parents[n]
        reconst_path.append(start_node)
        reconst_path.reverse()
        print('Path found: {}'.format(reconst_path))
        return reconst_path
    for (m, weight) in self.get_neighbors(n):
        if m not in open_list and m not in closed_list:
            open_list.add(m)
            parents[m] = n
            g[m] = g[n] + weight
        else:
            if g[m] > g[n] + weight:
```

```
                g[m] = g[n] + weight
                parents[m] = n
                if m in closed_list:
                    closed_list.remove(m)
                    open_list.add(m)
            open_list.remove(n)
            closed_list.add(n)
        print('Path does not exist!')
        return None
adjacency_list = {'A': [('B', 1), ('C', 3), ('D', 7)],'B': [('D', 5)],'C': [('D', 12)]}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

OUTPUT:

Path found: ['A', 'B', 'D']

EX NO: 2B

DATE:

IMPLEMENTATION OF INFORMED SEARCH ALGORITHMS(A*,BFS)

AIM:

To write a python program for BEST_FIRST_SEARCH

Algorithm:

Step1: Start the program.

Step2: Declaring the variables.

Step3: Declaring the function.

Step4: Call the function and perform the operations within loop statements.

Step5: Using the print() the output will be generated.

Step6: Stop the program.

PROGRAM:

```
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]
def best_first_search(actual_Src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_Src] = True
    print("best_first_search: ")
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        if u == target:
            break
        for v, c in graph[u]:
```

```
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print(" ")
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)
source = 0
target = 15
best_first_search(source, target, v)
```

<u>OUTPUT:</u>

best_first_search:

0 1 3 2 8 9 11 13 10 5 4 12 6 7

EX NO: 3

IMPLEMENT NAIVE BAYES MODELS

DATE:

AIM:

To write a python program for NAIVE BAYES MODELS

Algorithm:

Step1: Start the program.

Step2: Import the modules related to it.

Step3: Using the pandas module we have the capability of reading the csv file – datasets.

Step4: Declare the variables and functions.

Step5: Using the print() function the respective output statements will be printed.

Step6: Using the extensions the required map and table will be generated.

Step7: Stop the program.

PROGRAM:

```
import pandas as pd
class NaiveBayesClassifier:
    def __init__(self, X, y):
        self.X, self.y = X, y
        self.N = len(self.X)
        self.dim = len(self.X[0])
        self.attrs = [[] for _ in range(self.dim)]
        self.output_dom = {}
        self.data = []
        for i in range(len(self.X)):
            for j in range(self.dim):
```

```
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])
            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1
            else:
                self.output_dom[self.y[i]] += 1
            self.data.append([self.X[i], self.y[i]])
    def classify(self, entry):
        solve = None
        max_arg = -1
        for y in self.output_dom.keys():
            prob = self.output_dom[y]/self.N
            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                n = len(cases)
                prob *= n/self.N
            if prob > max_arg:
                max_arg = prob
                solve = y
        return solve
data = pd.read_csv("E:\\el\\train.csv")
print(data.head())
y = list(map(lambda v: 'yes' if v == 1 else 'no', data['Survived'].values))
X = data[['Pclass', 'Sex', 'Age', ]].values
print(len(y))
y_train = y[:600]
y_val = y[600:]
```

```
X_train = X[:600]
X_val = X[600:]
nbc = NaiveBayesClassifier(X_train, y_train)
total_cases = len(y_val)
good = 0
bad = 0
for i in range(total_cases):
    predict = nbc.classify(X_val[i])
    if y_val[i] == predict:
        good += 1
    else:
        bad += 1
print('TOTAL EXAMPLES:', total_cases)
print('RIGHT:', good)
print('WRONG:', bad)
print('NAIVE BAYES ACCURACY: ', good/total_cases)
```

OUTPUT:

```
PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0       3  ...   7.2500   NaN        S
1            2         1       1  ...  71.2833   C85        C
2            3         1       3  ...   7.9250   NaN        S
3            4         1       1  ...  53.1000  C123        S
4            5         0       3  ...   8.0500   NaN        S

[5 rows x 12 columns]
891
TOTAL EXAMPLES: 291
RIGHT: 217
WRONG: 74
NAIVE BAYES ACCURACY:  0.7457044673539519
```

EX NO: 4

IMPLEMENT BAYESIAN NETWORKS

DATE:

AIM:

To write a python program for Bayesian Networks

Algorithm:

Step1: Start the program.

Step2: Import the modules related to it.

Step3: Using the pandas module we have the capability of reading the csv file – datasets.

Step4: Declare the variables and functions.

Step5: Using the print() function the respective output statements will be printed.

Step6: Using the extensions the required map and table will be generated.

Step7: Stop the program.

PROGRAM:

```
import bnlearn

edges = [('task', 'size'),('lat var', 'size'), ('task', 'fill level'), ('task', 'object shape'),
('task', 'side graspable'), ('size', 'GrasPose'), ('task', 'GrasPose'), ('fill level',
'GrasPose'), ('object shape', 'GrasPose'), ('side graspable', 'GrasPose'), ('GrasPose',
'latvar'),]


DAG = bnlearn.make_DAG(edges)

print(DAG['adjmat'])

bnlearn.print_CPD(DAG)

bnlearn.plot(DAG)


df = pd.read_csv('C:\\Users\\Elango\\OneDrive\\Desktop\\lab\\Heart.csv')
```

```
DAG = bnlearn.parameter_learning.fit(DAG,df,methodtype='maximumlikelihood')

bnlearn.print_CPD(DAG)

q1 = bnlearn.inference.fit(DAG, variables=['lat var'], evidence={'fill level':1,
'size':0, 'task':1})

df = bnlearn.import_example('sprinkler')

print(df)

edges = [('Cloudy', 'Sprinkler'),('Cloudy', 'Rain'),('Sprinkler', 'Wet_Grass'),('Rain',
'Wet_Grass')]


DAG = bnlearn.make_DAG(edges)

bnlearn.print_CPD(DAG)

bnlearn.plot(DAG)

DAG = bnlearn.parameter_learning.fit(DAG, df)

bnlearn.print_CPD(DAG)


q1 = bnlearn.inference.fit(DAG, variables=['Wet_Grass'], evidence={'Rain':1,
'Sprinkler':0, 'Cloudy':1})


print(q1.values)
```

<u>OUTPUT:</u>

```
[bnlearn] >bayes DAG created.

target          task   size  lat var  ...  side graspable  GrasPose  latvar

source                               ...

task          False   True    False  ...          True      True   False
```

size	False	False	False	...	False	True	False
lat var	False	True	False	...	False	False	False
fill level	False	False	False	...	False	True	False
object shape	False	False	False	...	False	True	False
side graspable	False	False	False	...	False	True	False
GrasPose	False	False	False	...	False	False	True
latvar	False	False	False	...	False	False	False

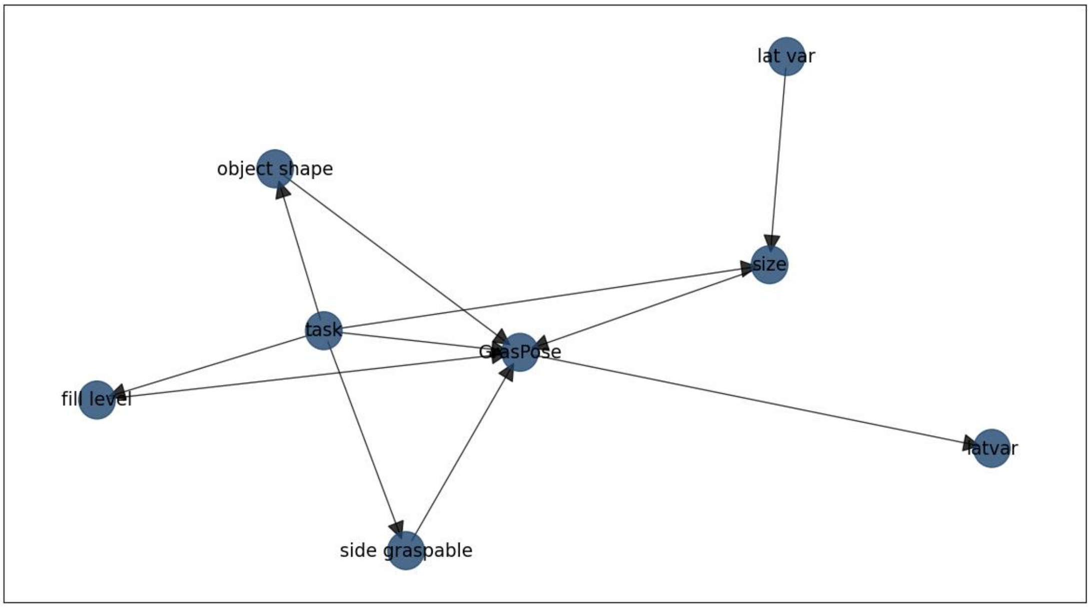
[8 rows x 8 columns]

[bnlearn] >No CPDs to print. Hint: Add CPDs as following: <bn.make_DAG(DAG, CPD=[cpd_A, cpd_B, etc])> and use bnlearn.plot(DAG) to make a plot.

[bnlearn] >Set node properties.

[bnlearn] >Set edge properties.

[bnlearn] >Plot based on Bayesian model



EX NO: 5

DATE:

BUILD REGRESSION MODELS

AIM:

To write a python program for Regression Models

Algorithm:

Step1: Start the program.

Step2: Import the modules related to it.

Step3: Using the pandas module we have the capability of reading the csv file – datasets.

Step4: Declare the variables and functions.

Step5: Using the print() function the respective output statements will be printed.

Step6: Using the extensions the required map and table will be generated.

Step7: Stop the program.

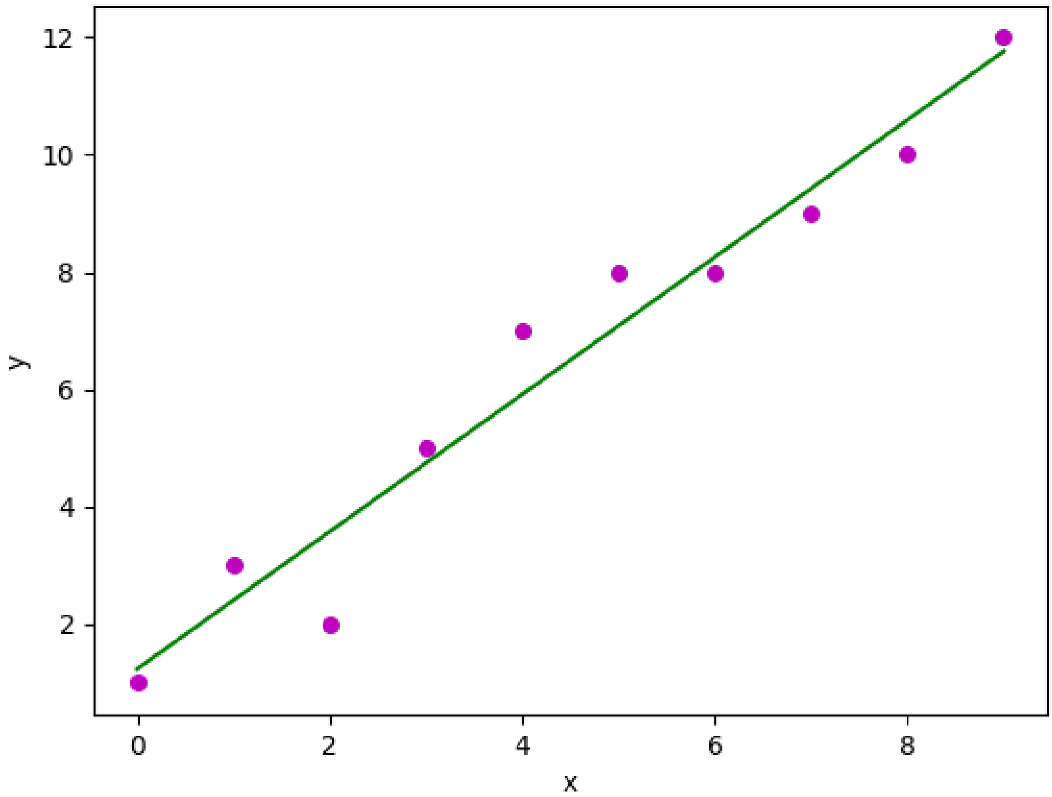
PROGRAM:

```
import numpy as np

import matplotlib.pyplot as plt


def estimate_coef(x, y):

        n = np.size(x)

        m_x = np.mean(x)

        m_y = np.mean(y)

        SS_xy = np.sum(y*x) - n*m_y*m_x

        SS_xx = np.sum(x*x) - n*m_x*m_x

        b_1 = SS_xy / SS_xx

        b_0 = m_y - b_1*m_x
```

```
    return (b_0, b_1)

def plot_regression_line(x, y, b):
    plt.scatter(x, y, color = "m",marker = "o", s = 30)
    y_pred = b[0] + b[1]*x
    plt.plot(x, y_pred, color = "g")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

def main():
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {} \nb_1 = {}".format(b[0], b[1]))
    plot_regression_line(x, y, b)

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

OUTPUT:

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```



12
10
8
6
4
2
y
0
2
4
6
8
x

EX NO: 6

DATE:

BUILD DECISION TREES AND RANDOM FORESTS

AIM:

To write a python program for Decision Trees And Random Forests

Algorithm:

Step1: Start the program.

Step2: Import the modules related to it.

Step3: Using the pandas module we have the capability of reading the csv file – datasets.

Step4: Declare the variables and functions.

Step5: Using the print() function the respective output statements will be printed.

Step6: Using the extensions the required map and table will be generated.

Step7: Stop the program.

PROGRAM:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import seaborn as sns

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report

from sklearn.metrics import confusion_matrix
```

```python
raw_data =
pd.read_csv('C:\\Users\\Elango\\OneDrive\\Desktop\\lab\\kyphosis.csv')
raw_data.columns
raw_data.info()

sns.pairplot(raw_data, hue = 'Kyphosis')
x = raw_data.drop('Kyphosis', axis = 1)

y = raw_data['Kyphosis']

x_training_data, x_test_data, y_training_data, y_test_data = train_test_split(x, y,
test_size = 0.3)
model = DecisionTreeClassifier()
model.fit(x_training_data, y_training_data)
predictions = model.predict(x_test_data)
print("_____________________________________________________________")
print("DecisionTree: \n")
print(classification_report(y_test_data, predictions))
print(confusion_matrix(y_test_data, predictions))
print("_____________________________________________________________")
random_forest_model = RandomForestClassifier()
random_forest_model.fit(x_training_data, y_training_data)

random_forest_predictions = random_forest_model.predict(x_test_data)
print("RandomForest: \n")
print(classification_report(y_test_data, random_forest_predictions))
print(confusion_matrix(y_test_data, random_forest_predictions))
print("_____________________________________________________________")
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Kyphosis  81 non-null     object
 1   Age       81 non-null     int64
 2   Number    81 non-null     int64
 3   Start     81 non-null     int64
dtypes: int64(3), object(1)
memory usage: 2.7+ KB
```

DecisionTree:

```
              precision    recall  f1-score   support

      absent       0.82      0.90      0.86        20
     present       0.33      0.20      0.25         5

    accuracy                           0.76        25
   macro avg       0.58      0.55      0.55        25
weighted avg       0.72      0.76      0.74        25

[[18  2]
 [ 4  1]]
```

RandomForest:

```
              precision    recall  f1-score   support

      absent       0.79      0.95      0.86        20
     present       0.00      0.00      0.00         5

    accuracy                           0.76        25
   macro avg       0.40      0.47      0.43        25
weighted avg       0.63      0.76      0.69        25

[[19  1]
 [ 5  0]]
```

EX NO: 7

DATE:

BUILD SVM MODELS

AIM:

To write a python program for support vector machine.

ALGORTHIM:

STEP1: Start the Program

STEP2: Import the python library packages like pandas,numpy,sklearn.

STEP3: Download the datasets fish

STEP4: Declaring the variables and function

STEP5: Print the results

STEP6: Stop

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

fish = pd.read_csv("C:\\Users\\Elango\\OneDrive\\Desktop\\lab\\fish.csv")
X = fish.drop(['Species'], axis = 'columns')
y = fish.Species

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2)

model = SVC(kernel = 'linear', C = 1)
```

```
model.fit(X_train, y_train)

svm_pred = model.predict(X_test)
accuracy = model.score(X_test, y_test)
print("prediction of svm: ",svm_pred)
print("\nAccuracy of svm:",accuracy)
```

OUTPUT:

```
prediction of svm:  ['Perch' 'Perch' 'Parkki' 'Roach' 'Perch' 'Smelt' 'Bream' 'Roach' 'Smelt'
 'Bream' 'Bream' 'Pike' 'Bream' 'Perch' 'Bream' 'Perch' 'Bream' 'Perch'
 'Pike' 'Perch' 'Perch' 'Roach' 'Smelt' 'Perch' 'Roach' 'Whitefish'
 'Perch' 'Perch' 'Perch' 'Perch' 'Perch' 'Perch']

Accuracy of svm: 1.0
```

EX NO: 8

IMPLEMENT ENSEMBLING TECHNIQUES

DATE:

AIM:

To write a python program for ensemble techniques.

ALGORTHIM:

STEP1: Start the Program

STEP2: Import the python library packages like pandas,numpy,sklearn.

STEP3: From sklearn import classification and regression for weight and voting accuracy.

STEP4: Declaring the variables and function

STEP5: Print the results

STEP6: Stop

PROGRAM:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier



def get_models():
 models = list()
```

```
 models.append(('lr', LogisticRegression()))
 models.append(('cart', DecisionTreeClassifier()))
 models.append(('bayes', GaussianNB()))
 return models

def evaluate_models(models, X_train, X_val, y_train, y_val):
 scores = list()
 for name, model in models:
  model.fit(X_train, y_train)
  yhat = model.predict(X_val)
  acc = accuracy_score(y_val, yhat)
  scores.append(acc)
 return scores

X, y = make_classification(n_samples=10000, n_features=20, n_informative=15,
n_redundant=5, random_state=7)
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.50,
random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full,
test_size=0.33, random_state=1)
models = get_models()
scores = evaluate_models(models, X_train, X_val, y_train, y_val)
print(scores)
ensemble = VotingClassifier(estimators=models, voting='soft', weights=scores)
ensemble.fit(X_train_full, y_train_full)
yhat = ensemble.predict(X_test)

score = accuracy_score(y_test, yhat)
```

```
print('Weighted Avg Accuracy: %.3f' % (score*100))
scores = evaluate_models(models, X_train_full, X_test, y_train_full, y_test)
for i in range(len(models)):
 print('>%s: %.3f' % (models[i][0], scores[i]*100))
ensemble = VotingClassifier(estimators=models, voting='soft')
ensemble.fit(X_train_full, y_train_full)
yhat = ensemble.predict(X_test)
score = accuracy_score(y_test, yhat)
print('Voting Accuracy: %.3f' % (score*100))
```

<u>OUTPUT:</u>

```
[0.8896969696969697, 0.8624242424242424, 0.8812121212121212]
Weighted Avg Accuracy: 90.940
>lr: 87.800
>cart: 88.420
>bayes: 87.300
Voting Accuracy: 90.720
```

EX NO: 9

IMPLEMENT CLUSTERING ALGORITHMS

DATE:

AIM:

To write a python program for clustering algorthims.

ALGORTHIM:

STEP1: Start the Program

STEP2: Import the python library packages like pandas,numpy,sklearn,matplotlib.

STEP3: From sklearn import kmeans .

STEP4: Declaring the variables and function

STEP5: Print the results

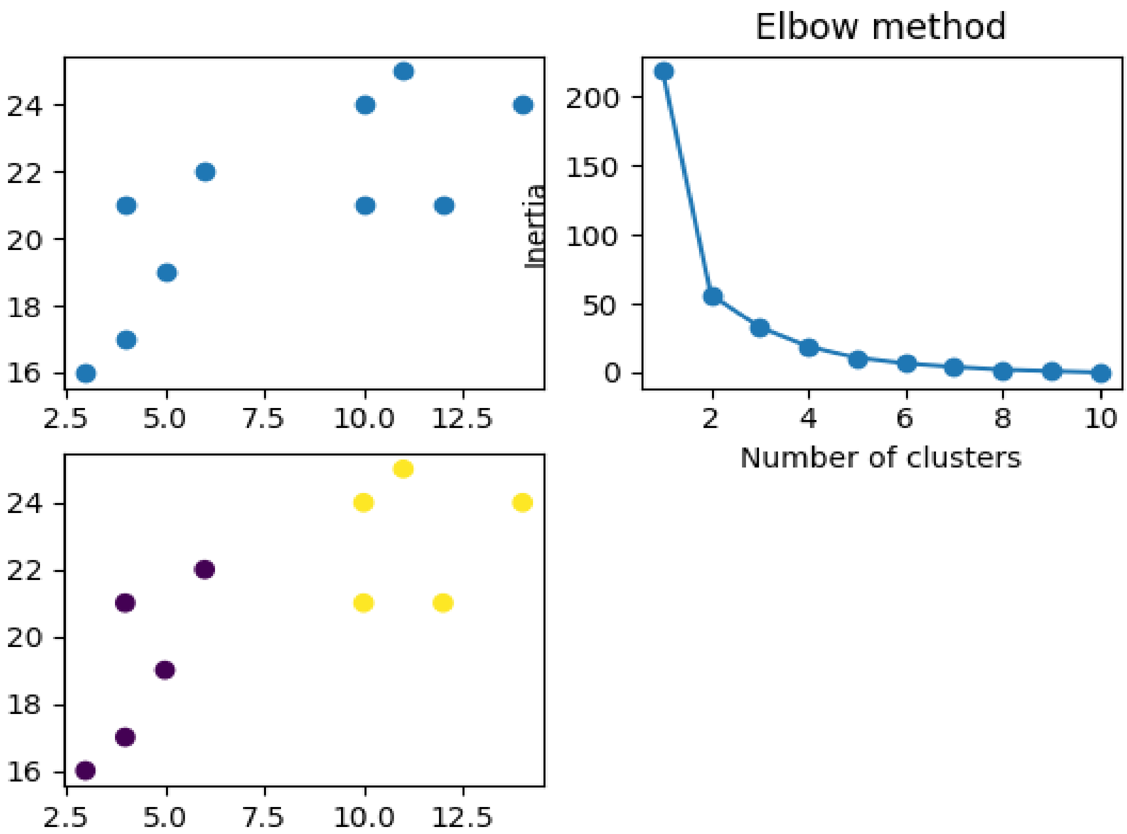
STEP6: Stop

PROGRAM:

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
plt.subplot(2,2,1)
plt.scatter(x, y)
data = list(zip(x, y))
inertias = []
plt.subplot(2,2,2)
for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)
plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
```

```
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.subplot(2,2,3)
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)
plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```

<u>OUTPUT:</u>



EX NO: 10

DATE:

IMPLEMENT EM FOR BAYESIAN NETWORK

AIM:

To write a python program to implement expectation maximization for bayesian network.

ALGORTHIM:

STEP1: Start the Program.

STEP2: Import the python library packages like pandas,numpy,sklearn,matplotlib,warnings.

STEP3: From sklearn import linear model.

STEP4: Declaring the variables and function.

STEP5: Print the results.

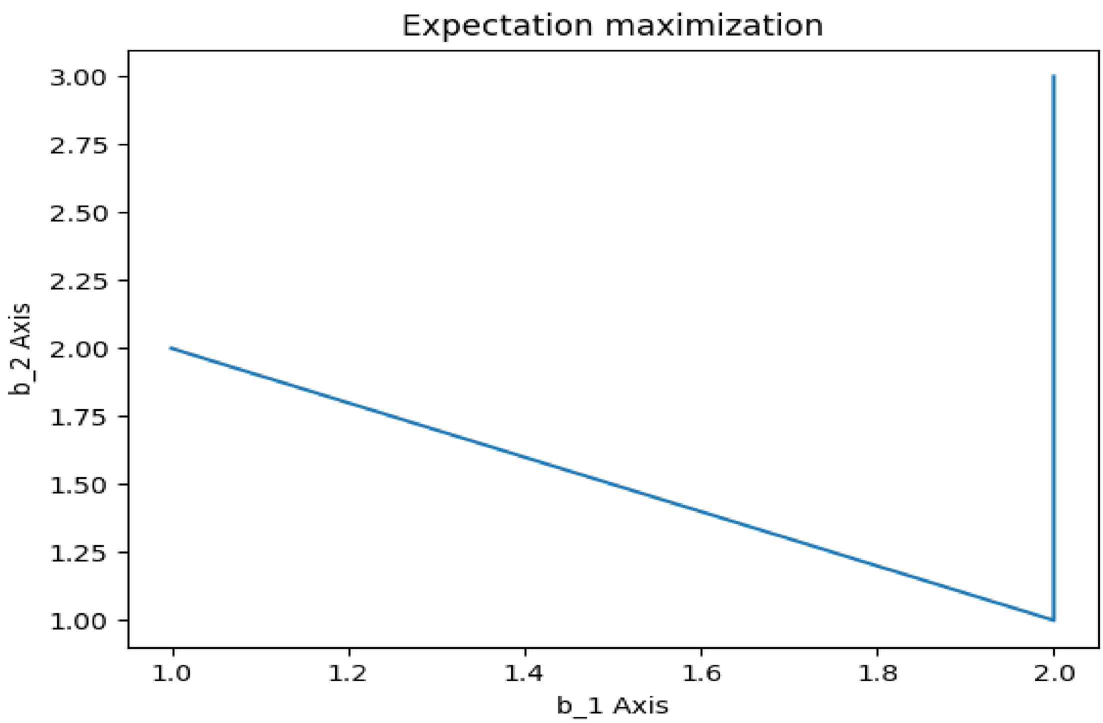
STEP6: Stop.

PROGRAM:

```
import numpy as np
from sklearn import datasets, linear_model
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
b_1 = np.array([1,1,2,2,2])
b_2 = np.array([2,2,1,1,3])
iteration = 500
data_y = np.random.random_integers(500, size=(300, 1))
data_x = np.random.random_integers(500, size=(300, 5))
def em(b_1,b_2,iteration, data_y, data_x):
    for t in range(1, iteration+1):
        J_1=[]
        J_2=[]
```

```
        for i in range(len(data_y)):
            if abs(data_y[i]-(np.dot(data_x[i],b_1))) < abs(data_y[i]-
(np.dot(data_x[i],b_2))):
                J_1.append(i)
            else:
                J_2.append(i)
        b_1 = np.argmin((np.sum(J_1[data_y] - (J_1[data_x])*b_1)**2)**.5)
        b_2 = np.argmin((np.sum(J_2[data_y] - (J_2[data_x])*b_2)**2)**.5)
    return b_1, b_2
plt.xlabel("b_1 Axis")
plt.ylabel("b_2 Axis")
plt.title("Expectation maximization")
plt.plot(b_1, b_2)
plt.show()
```

<u>OUTPUT:</u>



EX NO: 11

BUILD SIMPLE NN MODELS

DATE:

AIM:

To write a python program to implement simple neural networks models.

ALGORTHIM:

STEP1:Start the Program.

STEP2:Import the python library packages like pandas,numpy,tensorflow.

STEP3:Download the datasets winequality-red.

STEP4:Declaring the variables and function.

STEP5:Print the results.

STEP6:Stop.

PROGRAM:

```
import tensorflow as tf
import numpy as np
import pandas as pd
df = pd.read_csv('C:\\Users\\Elango\\OneDrive\\Desktop\\lab\\winequality-red.csv')
df.head()
train_df = df.sample(frac=0.75, random_state=4)
val_df = df.drop(train_df.index)
max_val = train_df.max(axis= 0)
min_val = train_df.min(axis= 0)
range = max_val - min_val
train_df = (train_df - min_val)/(range)
val_df =  (val_df- min_val)/range
X_train = train_df.drop('quality',axis=1)
X_val = val_df.drop('quality',axis=1)
```

```
y_train = train_df['quality']

y_val = val_df['quality']

input_shape = [X_train.shape[1]]

input_shape

model = tf.keras.Sequential([tf.keras.layers.Dense(units=64,
activation='relu',input_shape=input_shape),tf.keras.layers.Dense(units=64,
activation='relu'),tf.keras.layers.Dense(units=1)])

model.summary()
```

<u>OUTPUT:</u>

```
Model: "sequential"
_________________________________________________________________
 Layer (type)              Output Shape            Param #
=================================================================
 dense (Dense)              (None, 64)             768

 dense_1 (Dense)            (None, 64)              4160

 dense_2 (Dense)            (None, 1)              65

=================================================================
Total params: 4,993
Trainable params: 4,993
Non-trainable params: 0
_________________________________________________________________
```

EX NO: 12

BUILD DEEP LEARNING NN MODELS

DATE:

AIM:

To write a python program to implement deep learning neural network models.

ALGORTHIM:

STEP1: Start the Program.

STEP2: Import the python library packages like pandas,numpy,tensorflow.

STEP3: Download the datasets pima-indians-diabetes.data

STEP4: Declaring the variables and function.

STEP5: Print the results.

STEP6: Stop.

PROGRAM:

```
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
dataset = loadtxt('C:\\Users\\Elango\\OneDrive\\Desktop\\lab\\pima-indians-diabetes.data.csv', delimiter=',')
X = dataset[:,0:8]
y = dataset[:,8]
model = Sequential()
model.add(Dense(12, input_shape=(8,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=20, batch_size=10)
_, accuracy = model.evaluate(X, y)
print('\nAccuracy: %.2f' % (accuracy*100))
```

OUTPUT:

Epoch 1/20

1/77 [..............................] - ETA: 51s - loss: 12.2142 - accuracy: 0.3000
38/77 [=============>................] - ETA: 0s - loss: 7.3044 - accuracy: 0.5000

77/77 [==============================] - 1s 1ms/step - loss: 5.1459 - accuracy: 0.5117

Epoch 2/20

1/77 [..............................] - ETA: 0s - loss: 2.2128 - accuracy: 0.4000
50/77 [==================>...........] - ETA: 0s - loss: 1.4614 - accuracy: 0.5840
51/77 [==================>...........] - ETA: 0s - loss: 1.4540 - accuracy: 0.5843
77/77 [==============================] - 0s 2ms/step - loss: 1.3755 - accuracy: 0.5794

Epoch 3/20

1/77 [..............................] - ETA: 0s - loss: 0.4842 - accuracy: 0.8000
47/77 [=================>............] - ETA: 0s - loss: 1.2925 - accuracy: 0.5809
48/77 [=================>............] - ETA: 0s - loss: 1.2968 - accuracy: 0.5771
77/77 [==============================] - 0s 2ms/step - loss: 1.1937 - accuracy: 0.6107

Epoch 4/20

1/77 [..............................] - ETA: 0s - loss: 1.0799 - accuracy: 0.5000
48/77 [=================>............] - ETA: 0s - loss: 0.9900 - accuracy: 0.6583
49/77 [==================>...........] - ETA: 0s - loss: 0.9851 - accuracy: 0.6612
77/77 [==============================] - 0s 2ms/step - loss: 1.0293 - accuracy: 0.6406

Epoch 5/20

 1/77 [..............................] - ETA: 1s - loss: 0.4911 - accuracy: 0.8000
58/77 [=====================>........] - ETA: 0s - loss: 1.0135 - accuracy:
0.6224
59/77 [=====================>........] - ETA: 0s - loss: 1.0182 - accuracy:
0.6237
77/77 [==============================] - 0s 2ms/step - loss: 0.9720 -
accuracy: 0.6302

Epoch 6/20

 1/77 [..............................] - ETA: 1s - loss: 2.1297 - accuracy: 0.4000
55/77 [====================>.........] - ETA: 0s - loss: 0.9519 - accuracy:
0.5964
56/77 [====================>.........] - ETA: 0s - loss: 0.9544 - accuracy:
0.5946
77/77 [==============================] - 0s 2ms/step - loss: 0.9069 -
accuracy: 0.6224

Epoch 7/20

 1/77 [..............................] - ETA: 0s - loss: 1.0358 - accuracy: 0.5000
45/77 [================>.............] - ETA: 0s - loss: 0.9617 - accuracy:
0.6244
46/77 [================>.............] - ETA: 0s - loss: 0.9590 - accuracy:
0.6239
77/77 [==============================] - 0s 2ms/step - loss: 0.9292 -
accuracy: 0.6237

Epoch 8/20

 1/77 [..............................] - ETA: 0s - loss: 0.7088 - accuracy: 0.7000
43/77 [===============>..............] - ETA: 0s - loss: 0.7955 - accuracy: 0.6163
44/77 [================>.............] - ETA: 0s - loss: 0.7904 - accuracy:
0.6205
73/77 [===========================>..] - ETA: 0s - loss: 0.8354 -
accuracy: 0.6233

77/77 [==============================] - 0s 3ms/step - loss: 0.8314 -
accuracy: 0.6250

Epoch 9/20

```
 1/77 [..............................] - ETA: 1s - loss: 1.0295 - accuracy: 0.5000
59/77 [=====================>........] - ETA: 0s - loss: 0.8002 - accuracy: 0.6339
60/77 [======================>.......] - ETA: 0s - loss: 0.8011 - accuracy: 0.6350
77/77 [==============================] - 0s 2ms/step - loss: 0.7969 - accuracy: 0.6341
```

Epoch 10/20

```
 1/77 [..............................] - ETA: 1s - loss: 0.6393 - accuracy: 0.7000
63/77 [=======================>......] - ETA: 0s - loss: 0.8175 - accuracy: 0.6333
64/77 [=======================>......] - ETA: 0s - loss: 0.8225 - accuracy: 0.6313
77/77 [==============================] - 0s 2ms/step - loss: 0.7726 - accuracy: 0.6536
```

Epoch 11/20

```
 1/77 [..............................] - ETA: 1s - loss: 0.5027 - accuracy: 0.7000
56/77 [====================>.........] - ETA: 0s - loss: 0.7737 - accuracy: 0.6679
57/77 [=====================>........] - ETA: 0s - loss: 0.7846 - accuracy: 0.6667
77/77 [==============================] - 0s 2ms/step - loss: 0.7822 - accuracy: 0.6549
```

Epoch 12/20

```
 1/77 [..............................] - ETA: 1s - loss: 1.1196 - accuracy: 0.7000
58/77 [=====================>........] - ETA: 0s - loss: 0.7504 - accuracy: 0.6448
59/77 [=====================>........] - ETA: 0s - loss: 0.7489 - accuracy: 0.6458
77/77 [==============================] - 0s 2ms/step - loss: 0.7496 - accuracy: 0.6562
```

Epoch 13/20

1/77 [..............................] - ETA: 0s - loss: 0.4070 - accuracy: 0.8000
47/77 [=================>............] - ETA: 0s - loss: 0.7324 - accuracy: 0.6319
48/77 [=================>............] - ETA: 0s - loss: 0.7286 - accuracy: 0.6375
77/77 [==============================] - 0s 2ms/step - loss: 0.7349 - accuracy: 0.6419

Epoch 14/20

1/77 [..............................] - ETA: 0s - loss: 0.7780 - accuracy: 0.5000
46/77 [================>.............] - ETA: 0s - loss: 0.7433 - accuracy: 0.6370
47/77 [=================>............] - ETA: 0s - loss: 0.7411 - accuracy: 0.6383
75/77 [============================>.] - ETA: 0s - loss: 0.7248 - accuracy: 0.6507

77/77 [==============================] - 0s 2ms/step - loss: 0.7199 - accuracy: 0.6523

Epoch 15/20

1/77 [..............................] - ETA: 1s - loss: 1.1487 - accuracy: 0.5000
62/77 [=======================>......] - ETA: 0s - loss: 0.7370 - accuracy: 0.6403
63/77 [=======================>......] - ETA: 0s - loss: 0.7369 - accuracy: 0.6397
77/77 [==============================] - 0s 2ms/step - loss: 0.7042 - accuracy: 0.6641

Epoch 16/20

1/77 [..............................] - ETA: 1s - loss: 0.7917 - accuracy: 0.5000
55/77 [====================>.........] - ETA: 0s - loss: 0.6688 - accuracy: 0.6945
56/77 [====================>.........] - ETA: 0s - loss: 0.6685 - accuracy: 0.6964

77/77 [==============================] - 0s 2ms/step - loss: 0.7121 - accuracy: 0.6732

Epoch 17/20

1/77 [..............................] - ETA: 0s - loss: 0.8313 - accuracy: 0.5000
46/77 [================>.............] - ETA: 0s - loss: 0.7169 - accuracy: 0.6739
47/77 [=================>............] - ETA: 0s - loss: 0.7165 - accuracy: 0.6723
77/77 [==============================] - 0s 2ms/step - loss: 0.6944 - accuracy: 0.6784

Epoch 18/20

1/77 [..............................] - ETA: 0s - loss: 0.6247 - accuracy: 0.6000
44/77 [================>.............] - ETA: 0s - loss: 0.6769 - accuracy: 0.6909

77/77 [==============================] - 0s 1ms/step - loss: 0.6802 - accuracy: 0.6758

Epoch 19/20

1/77 [..............................] - ETA: 0s - loss: 0.8535 - accuracy: 0.7000
45/77 [================>.............] - ETA: 0s - loss: 0.7046 - accuracy: 0.6756

77/77 [==============================] - 0s 1ms/step - loss: 0.6755 - accuracy: 0.6758

Epoch 20/20

1/77 [..............................] - ETA: 0s - loss: 0.5593 - accuracy: 0.7000
50/77 [==================>...........] - ETA: 0s - loss: 0.6808 - accuracy: 0.6640
51/77 [==================>...........] - ETA: 0s - loss: 0.6735 - accuracy: 0.6667
77/77 [==============================] - 0s 2ms/step - loss: 0.6669 - accuracy: 0.6810

1/24 [>.............................] - ETA: 2s - loss: 0.5648 - accuracy: 0.6875

24/24 [==============================] - 0s 1ms/step - loss: 0.6773 - accuracy: 0.6602

Accuracy: 66.02