Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

Department of Artificial Intelligence and Data Science

Course Name: MACHINE LEARNING LABORATORY

Course Code : 21AIL66

(Academic year 2023 -2024)

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MACHINE LEARNING

LABORATORY MANUAL

21AIL66

B.E - VI semester

(Effective from the academic year 2023 -2024)

Prepared by:
Ms. Vindya B J
Ms.Dhanushree c

Reviewed by:
Dr. Gavisiddappa

Approved by:
Dr. Gavisiddappa, HOD,
AD Department

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SYLLABUS

MACHINE LEARNING LABORATORY
SEMESTER – VI

Course Code: 21AIL66	CIE Marks: 50
Teaching Hours/Week(L:T:P:S): 0:0:2:0	SEE Marks: 50
Total hours of pedogogy: 24 hours	Total Marks:100
Credits – 01	Exam Hours: 03

Course Learning Objectives:

CLO 1: To learn and understand the importance machine learning algorithms
CLO 2: Compare and contrast the learning techniques like ANN approach, Bayesian learning and reinforcement learning.

CLO 3: Able to solve and analyze the problems on ANN, instance based learning and reinforcement learning techniques.

CLO 4: To impact the knowledge of clustering and classification algorithms for predictions and evaluating hypothesis.

Prerequisite:

- Students should be familiarized about Python installation and setting Python environment
- Usage and installation of Anaconda should be introduced https://www.anaconda.com/products/individual
- Should have the knowledge about Probability theory, Statistics theory and linear Algebra.
- Should have the knowledge of numpy,pandas,scikit-learn and scipy library packages.

Programs List: PART A

List of problems for which student should develop program and execute in the laboratory.

1. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

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

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

5. Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

6. 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 Python ML library classes/API.

7. 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 Python ML library classes/API in the program.

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

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

10. Implement and demonstrate the working of SVM algorithm for classification.

PART –B

A problem statement for each batch is to be generated in consultation with the co-examiner and student should develop an algorithm, program and execute the Program for the given problem with appropriate outputs.

Course Outcomes: At the end of the course the student will be able to:

CO 1. Understand the Importance of different classification and clustering algorithms.
CO 2. Demonstrate the working of various algorithms with respect to training and test data sets.
CO 3. Illustrate and analyze the principles of Instance based and Reinforcement learning techniques.
CO 4. Elicit the importance and Applications of Supervised and unsupervised machine learning.
CO 5. Compare and contrast the Bayes theorem principles and Q learning approach.

Text Books:

1. Tom M Mitchell, “Machine Lerning”,1st Edition, McGraw Hill Education, 2017.
2. Nello Cristianini, John Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, 2013
3. Allen B. Downey, “Think Python: How to Think Like a Computer Scientist”, 2nd Edition, Green Tea Press, 2015. (Available under CC-BY-NC license at http://greenteapress.com/thinkpython2/thinkpython2.pdf)

PROGRAM 1:

Aim: Illustrate and Demonstrate the working model and principle of Find-S algorithm.

Program: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
import csv
hypo = ['%','%','%','%','%','%'];
with open('trainingdata.csv') as csv_file:
        readcsv = csv.reader(csv_file, delimiter=',')
        print(readcsv)
        data = []
        print("\nThe given training examples are:")
        for row in readcsv:
            print(row)
            if row[len(row)-1].upper() == "YES":
                data.append(row)
print("\nThe positive examples are:");
for x in data:
    print(x);
print("\n");
TotalExamples = len(data);
i=0;
j=0;
k=0;
print("The steps of the Find-s algorithm are:\n",hypo);
list=[];
p=0;
d=len(data[p])-1;
for j in range(d):
    list.append(data[i][j]);
```

```
hypo=list;
i=1;
for i in range(TotalExamples):
    for k in range(d):
        if hypo[k]!=data[i][k]:
            hypo[k]='?';
            k=k+1;
        else:
            hypo[k];
    print(hypo);
i=i+1;
print("\nThe maximally specific find-s hypothesis for the given training examples is :");
list=[];
for i in range(d):
    list.append(hypo[i]);
print(list);
```

OUTPUT

```
<_csv.reader object at 0x0000010D0BC55820>

The given training examples are:
['sky', 'airTemp', 'humidity', 'wind', 'water', 'forecast', 'enjoySport']
['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 positive examples are:
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

The steps of the Find-s algorithm are:
 ['%', '%', '%', '%', '%', '%']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', '?', '?']

The maximally specific find-s hypothesis for the given training examples is
:
```

```
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

PROGRAM 2:

Aim: Demonstrate the working model and principle of candidate elimination algorithm.

Program: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
import csv
with open('EnjoySport.csv') as csvFile:
 examples = [tuple(line) for line in csv.reader(csvFile)]
print(examples)
def get_domains(examples):
 d = [set() for i in examples[0]]

 for x in examples:
 for i, xi in enumerate(x):
 d[i].add(xi)
 return [list(sorted(x)) for x in d]
get_domains(examples)
def g_0(n):
 return ('?',)*n
def s_0(n):
 return ('0',)*n
def more_general(h1, h2):
 more_general_parts = []
 for x, y in zip(h1, h2):
 mg = x == '?' or (x != '0' and (x == y or y == '0'))
 more_general_parts.append(mg)
 return all(more_general_parts)
def consistent(hypothesis,example):
 return more_general(hypothesis, example)
def min_generalizations(h, x):
 h_new = list(h)
 for i in range(len(h)):
 if not consistent(h[i:i+1],x[i:i+1]):
 if h[i] != '0':
 h_new[i] = '?'
 else:
 h_new[i] = x[i]
 return [tuple(h_new)]
def generalize_S(x, G, S):
 S_prev = list(S)
 for s in S_prev:
 if s not in S:
 continue
 if not consistent(s,x):
 S.remove(s)
 Splus = min_generalizations(s, x)
 S.update([h for h in Splus if any([more_general(g,h) for g in G])])
 S.difference_update([h for h in S if any([more_general(h, h1) for h1 in S if h != h1])])
 return S
def min_specializations(h, domains, x):
 results = []
 for i in range(len(h)):
 if h[i] == '?':
 for val in domains[i]:
 if x[i] != val:
 h_new = h[:i] + (val,) + h[i+1:]
 results.append(h_new)
 elif h[i] != '0':
 h_new = h[:i] + ('0',) + h[i+1:]
 results.append(h_new)
 return results
def specialize_G(x, domains, G, S):
 G_prev = list(G)
 for g in G_prev:
 if g not in G:
 continue
 if consistent(g,x):
 G.remove(g)
 Gminus = min_specializations(g, domains, x)
 G.update([h for h in Gminus if any([more_general(h, s) for s in S])])
 G.difference_update([h for h in G if any([more_general(g1, h) for g1 in G if h != g1])])
 return G
def candidate_elimination(examples):
 domains = get_domains(examples)[:-1]

 G = set([g_0(len(domains))])
 S = set([s_0(len(domains))])
 i=0
 print('All the hypotheses in General and Specific boundary are:\n')
 print('\n G[{0}]:'.format(i),G)
 print('\n S[{0}]:'.format(i),S)
 for xcx in examples:
 i=i+1
 x, cx = xcx[:-1], xcx[-1]
 if cx=='Yes':
 G = {g for g in G if consistent(g,x)}
 S = generalize_S(x, G, S)
 else:
 S = {s for s in S if not consistent(s,x)}
 G = specialize_G(x, domains, G, S)
 print('\n G[{0}]:'.format(i),G)
 print('\n S[{0}]:'.format(i),S)
return
candidate_elimination(example
```

OUTPUT

```
[('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'), ('Sunny'
, 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'), ('Rainy', 'Cold', '
High', 'Strong', 'Warm', 'Change', 'No'), ('Sunny', 'Warm', 'High', 'St
rong', 'Cool', 'Change', 'Yes')]
All the hypotheses in General and Specific boundary are:
G[0]: {('?', '?', '?', '?', '?', '?')}
S[0]: {('0', '0', '0', '0', '0', '0')}
G[1]: {('?', '?', '?', '?', '?', '?')}
S[1]: {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}
G[2]: {('?', '?', '?', '?', '?', '?')}
S[2]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}
G[3]: {('?', '?', '?', '?', '?', 'Same'), ('Sunny', '?', '?', '?', '?'
, '?'), ('?', 'Warm', '?', '?', '?', '?')}
S[3]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}
G[4]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?'
, '?')}
S[4]: {('Sunny', 'Warm', '?', 'Strong', '?', '?')} [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', 'Normal',
'?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
```

PROGRAM 3:

Aim: To construct the Decision tree using the training data sets under supervised learning concept.

Program: 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 numpy as np
import math
import csv
def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)
class Node:
    def _init_(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

    def _str_(self):
        return self.attribute
def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")
        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
        if delete:
            dict[items[x]] = np.delete(dict[items[x]], col, 1)

    return items, dict
def entropy(S):
    items = np.unique(S)

    if items.size == 1:
        return 0

    counts = np.zeros((items.shape[0], 1))
    sums = 0

    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)

    for count in counts:
        sums += -1 * count * math.log(count, 2)
    return sums
def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)

    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))

    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)

    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)

    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

    return total_entropy / iv
def create_node(data, metadata):
    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node

    gains = np.zeros((data.shape[1] - 1, 1))

    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    split = np.argmax(gains)
    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)

    items, dict = subtables(data, split, delete=True)

    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))

    return node
def empty(size):
    s = ""
    for x in range(size):
        s += "   "
    return s

def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return
    print(empty(level), node.attribute)
    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)
metadata, traindata = read_data("tennisdata.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)
```

OUTPUT:

```
Outlook
   Overcast
     b'Yes'
   Rainy
     Windy
       b'False'
         b'Yes'
       b'True'
         b'No'
   Sunny
     Humidity
       b'High'
         b'No'
       b'Normal'
         b'Yes'
```

PROGRAM 4:

Aim: To understand the working principle of Artificial Neural network with feed forward and feed backward principle.

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

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)     # X = (hours sleeping, hours studying)
y = np.array(([92], [86], [89]), dtype=float)           # y = score on test

# scale units
X = X/np.amax(X, axis=0)       # maximum of X array
y = y/100                      # max test score is 100
class Neural_Network(object):
    def _init_(self):
                        # Parameters
        self.inputSize = 2
        self.outputSize = 1
        self.hiddenSize = 3
                        # Weights
        self.W1 = np.random.randn(self.inputSize, self.hiddenSize)           # (3x2) weight matrix from input to
hidden layer
        self.W2 = np.random.randn(self.hiddenSize, self.outputSize)        # (3x1) weight matrix from hidden to
output layer

    def forward(self, X):
                        #forward propagation through our network
        self.z = np.dot(X, self.W1)             # dot product of X (input) and first set of 3x2 weights
        self.z2 = self.sigmoid(self.z)          # activation function
        self.z3 = np.dot(self.z2, self.W2)       # dot product of hidden layer (z2) and second set of 3x1 weights
        o = self.sigmoid(self.z3)                # final activation function
        return o

    def sigmoid(self, s):
        return 1/(1+np.exp(-s))     # activation function

    def sigmoidPrime(self, s):
        return s * (1 - s)         # derivative of sigmoid

    def backward(self, X, y, o):
                                  # backward propgate through the network
        self.o_error = y - o        # error in output
        self.o_delta = self.o_error*self.sigmoidPrime(o) # applying derivative of sigmoid to
        self.z2_error = self.o_delta.dot(self.W2.T)    # z2 error: how much our hidden layer weights contributed
to output error
        self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2) # applying derivative of sigmoid to z2 error
        self.W1 += X.T.dot(self.z2_delta)       # adjusting first set (input --> hidden) weights
        self.W2 += self.z2.T.dot(self.o_delta)  # adjusting second set (hidden --> output) weights

    def train (self, X, y):
        o = self.forward(X)
        self.backward(X, y, o)
NN = Neural_Network()
for i in range(1000): # trains the NN 1,000 times
    print ("\nInput: \n" + str(X))
    print ("\nActual Output: \n" + str(y))
    print ("\nPredicted Output: \n" + str(NN.forward(X)))
    print ("\nLoss: \n" + str(np.mean(np.square(y - NN.forward(X)))))     # mean sum squared loss)
    NN.train(X, y)
```

OUTPUT:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

Predicted Output:
[[0.48117308]
 [0.47630198]
 [0.4828714 ]]

Loss:
0.16851564433468483

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

Predicted Output:
[[0.5407191 ]
 [0.53327742]
 [0.53775366]]

Loss:
0.12489304209583975

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

Predicted Output:
[[0.59141058]
 [0.58169703]
 [0.58469738]]

Loss:
0.09287774592933 43

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]
```

PROGRAM 5:

Aim: Demonstrate the text classifier using Naïve bayes classifier algorithm.

Program: Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
# import necessary libarities
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# load data from CSV
data = pd.read_csv('tennisdata.csv')
print("THe first 5 values of data is :\n",data.head())
# obtain Train data and Train output
X = data.iloc[:,:-1]
print("\nThe First 5 values of train data is\n",X.head())
# Convert then in numbers
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)
```

```
print("\nNow the Train data is :\n",X.head())
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)

classifier = GaussianNB()
classifier.fit(X_train,y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

OUTPUT:

```
THe first 5 values of data is :
    Outlook Temperature Humidity  Windy PlayTennis
0     Sunny         Hot     High  False         No
1     Sunny         Hot     High   True         No
2  Overcast         Hot     High  False        Yes
3     Rainy        Mild     High  False        Yes
4     Rainy        Cool   Normal  False        Yes

The First 5 values of train data is
    Outlook Temperature Humidity  Windy
0     Sunny         Hot     High  False
1     Sunny         Hot     High   True
2  Overcast         Hot     High  False
3     Rainy        Mild     High  False
4     Rainy        Cool   Normal  False

The first 5 values of Train output is
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object

Now the Train data is :
   Outlook  Temperature  Humidity  Windy
0        2            1         0      0
1        2            1         0      1
2        0            1         0      0
3        1            2         0      0
4        1            0         1      0
Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is: 1.0
```

PROGRAM 6:

Aim: Demonstrate and Analyse the results sets obtained from Bayesian belief network Principle.

Program:- 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 Python ML library classes/API.

```
import numpy as np
#from matplotlib import pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn import metrics
'''
The dataset contains cases from a study that was
conducted between 1958 and 1970 at the University
of Chicago's Billings Hospital on the survival of
patients who had undergone surgery for cancer
1. Age of patient at time of operation (numerical)
2. Patient's year of operation (year - 1900, numerical)
3. Number of positive axillary nodes detected (numerical)
4. Survival status (class attribute)
-- 1 = the patient survived 5 years or longer
-- 2 = the patient died within 5 year
'''
c1,c2,c3,c4= np.loadtxt('data.csv',unpack=True,delimiter = ',')
print("The age of patients are :",c1[0:5])
print("The year of operation are:",c2[0:5])
print("Number of positive axillary nodes detected are:",c3[0:5])
print("The labes (survived in 5 years[1-yes,2-no])",c4[0:5])
x=np.column_stack((c1,c3))
y=c4
clf = GaussianNB()
clf.fit(x,y)
predictions=clf.predict(x)
print("\n")
print('Accuracy of the classifer is:',accuracy_score(y,predictions))
print("\n")
print(metrics.classification_report(y,predictions))
```

<u>OUTPUT:</u>

```
The age of patients are : [ 30. 30. 30. 31. 31.]
The year of operation are: [ 64. 62. 65. 59. 65.]
Number of positive axillary nodes detected are: [ 1. 3. 0. 2. 4.]
The labes (survived in 5 years[1-yes,2-no]) [ 1. 1. 1. 1. 1.]
Accuracy of the classifer is: 0.748366013072
precision recall f1-score support
1.0 0.77 0.94 0.85 225
2.0 0.57 0.21 0.31 81
avg / total 0.71 0.75 0.70 306
```

PROGRAM 7:

Aim: Implement and demonstrate the working model of K-means clustering algorithm with Expectation Maximization Concept.

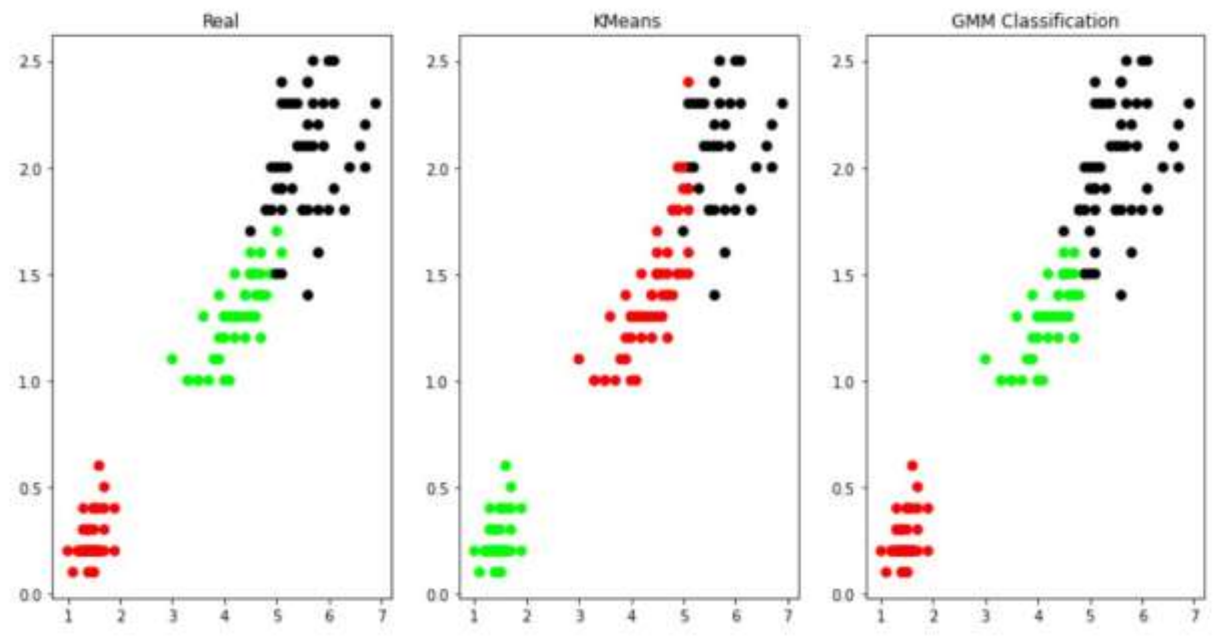
Program: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/API in the program.

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset=load_iris()
# print(dataset)

X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X)

plt.figure(figsize=(14,7))
```

```
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')

# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)

y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

OUTPUT:

```
Out[6]: Text(0.5, 1.0, 'GMM Classification')
```



PROGRAM 8:

Aim: Demonstrate and analyse the results of classification based on KNN Algorithm.

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

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np

dataset=load_iris()

X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=0)

kn=KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train,y_train)

for i in range(len(X_test)):
    x=X_test[i]
    x_new=np.array([x])
    prediction=kn.predict(x_new)

print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,dataset["targe
t_names"][prediction])
```

```
print(kn.score(X_test,y_test))
```

OUTPUT:

```
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
0.9736842105263158
```

PROGRAM 9:

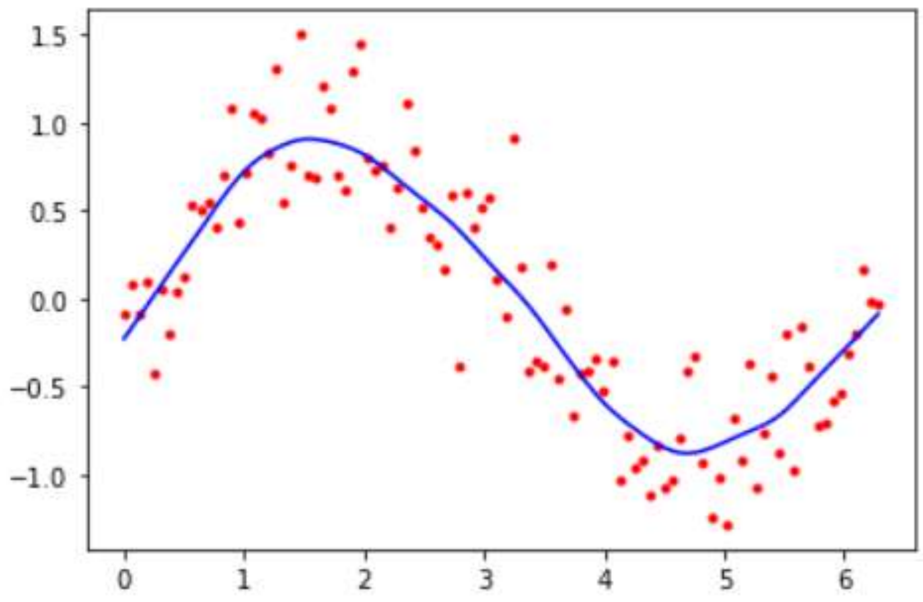
Aim: Understand and analyse the concept of Regression algorithm techniques.

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

```
from math import ceil
import numpy as np
from scipy import linalg

def lowess(x, y, f, iterations):
    n = len(x)
    r = int(ceil(f * n))
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
    w = (1 - w ** 3) ** 3
    yest = np.zeros(n)
    delta = np.ones(n)
    for iteration in range(iterations):
        for i in range(n):
            weights = delta * w[:, i]
            b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
            A = np.array([[np.sum(weights), np.sum(weights * x)],[np.sum(weights * x), np.sum(weights * x * x)]])
            beta = linalg.solve(A, b)
            yest[i] = beta[0] + beta[1] * x[i]

        residuals = y - yest
        s = np.median(np.abs(residuals))
        delta = np.clip(residuals / (6.0 * s), -1, 1)
        delta = (1 - delta ** 2) ** 2

    return yest

import math
n = 100
x = np.linspace(0, 2 * math.pi, n)
y = np.sin(x) + 0.3 * np.random.randn(n)
f =0.25
iterations=3
yest = lowess(x, y, f, iterations)

import matplotlib.pyplot as plt
plt.plot(x,y,"r.")
plt.plot(x,yest,"b-")
```

OUTPUT:

```
Out[1]: [<matplotlib.lines.Line2D at 0x2b57b9bbdf0>]
```



PROGRAM 10:

Aim: Implement and demonstrate classification algorithm using Support vector machine Algorithm.

Program: Implement and demonstrate the working of SVM algorithm for classification.

```
# Importing 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

# Load dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize SVM classifier
svm_classifier = SVC(kernel='linear')
```

```
# Train the SVM classifier
svm_classifier.fit(X_train, y_train)

# Make predictions
y_pred = svm_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Visualization (For 2D data)
if X.shape[1] == 2:
    # Plotting decision boundary
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
    Z = svm_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plotting data points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Support Vector Machine (SVM) Classifier')
    plt.show()
```

OUTPUT:

```
Accuracy: 1.0
```