RAJALAKSHMI INSTITUTE OF TECHNOLOGY

BELIEVE IN THE POSSIBILITIES

II Year/IV Semester

CS3491 – Artificial Intelligence & Machine Learning

Laboratory Manual

Centre of Excellence

COURSE OBJECTIVES:

1. Study about uninformed and Heuristic search techniques.
2. Learn techniques for reasoning under uncertainty
3. Introduce Machine Learning and supervised learning algorithms
4. Study about ensembling and unsupervised learning algorithms
5. Learn the basics of deep learning using neural networks

EXPERIMENTS LIST

1. Implementation of Uninformed search algorithms (BFS, DFS)
2. Implementation of Informed search algorithms (A*, memory-bounded A*)
3. Implement naïve Bayes models
4. Implement Bayesian Networks
5. Build Regression models
6. Build decision trees and random forests
7. Build SVM models
8. Implement ensembling techniques
9. Implement clustering algorithms
10. Implement EM for Bayesian networks
11. Build simple NN models
12. Build deep learning NN models

COURSE OUTCOMES:

On completion of the course, students will be able to:

CO1: Use appropriate search algorithms for problem solving

CO2: Apply reasoning under uncertainty

CO3: Build supervised learning models

CO4: Build ensembling and unsupervised models

CO5: Build deep learning neural network models

TEXT BOOKS:

1. Stuart Russell and Peter Norvig, “Artificial Intelligence – A Modern Approach”, Fourth Edition, Pearson Education, 2021.

2. Ethem Alpaydin, “Introduction to Machine Learning”, MIT Press, Fourth Edition, 2020.

REFERENCES:

1. Dan W. Patterson, “Introduction to Artificial Intelligence and Expert Systems”, Pearson Education,2007
2. Kevin Night, Elaine Rich, and Nair B., “Artificial Intelligence”, McGraw Hill, 2008
3. Patrick H. Winston, "Artificial Intelligence", Third Edition, Pearson Education, 2006
4. Deepak Khemani, “Artificial Intelligence”, Tata McGraw Hill Education, 2013 (http://nptel.ac.in/)
5. Christopher M. Bishop, “Pattern Recognition and Machine Learning”, Springer, 2006.
6. Tom Mitchell, “Machine Learning”, McGraw Hill, 3rd Edition,1997.
7. Charu C. Aggarwal, “Data Classification Algorithms and Applications”, CRC Press, 2014
8. Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar, “Foundations of Machine Learning”, MIT Press, 2012.
9. Ian Goodfellow, Yoshua Bengio, Aaron Courville, “Deep Learning”, MIT Press, 2016

CO-PO - Mapping - Laboratory Experiments

Exercises	CO	PO
1	CO1	PO1,PO2,PO3,PO4,PO9,PO10,PO11,PO12
2	CO1	PO1,PO2,PO3,PO4,PO9,PO10,PO11,PO12
3	CO2	PO1,PO2,PO3,PO4,PO5,PO9,PO10,PO11,PO12
4	CO2	PO1,PO2,PO3,PO4,PO5,PO9,PO10,PO11,PO12
5	CO3	PO1,PO2,PO3,PO4,PO5,PO9,PO10,PO11,PO12
6	CO3	PO1,PO2,PO3,PO4,PO5,PO9,PO10,PO11,PO12
7	CO3	PO1,PO2,PO3,PO4,PO5,PO9,PO10,PO11,PO12
8	CO3	PO1,PO2,PO3,PO4,PO5,PO9,PO10,PO11,PO12
9	CO4	PO1,PO2,PO3,PO4,PO9,PO10,PO11,PO12
10	CO4	PO1,PO2,PO3,PO4,PO9,PO10,PO11,PO12
11	CO5	PO1,PO2,PO3,PO4,PO5,PO9,PO10,PO11,PO12
12	CO5	PO1,PO2,PO3,PO4,PO5,PO9,PO10,PO11,PO12

INDEX

Expt No	Date	Title	Page No	Marks	Signature
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					

1. Implementation of Uninformed search algorithms (BFS, DFS)

Aim:

To implement uninformed search algorithms such as BFS and DFS.

Algorithm(BFS):

Step 1: SET STATUS = 1 (ready state) for each node in G
Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)
Step 3: Repeat Steps 4 and 5 until QUEUE is empty
Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).
Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set
their STATUS = 2
(waiting state)
[END OF LOOP]
Step 6: EXIT

Algorithm(DFS):

Step 1: SET STATUS = 1 (ready state) for each node in G
Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)
Step 3: Repeat Steps 4 and 5 until STACK is empty
Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)
Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)
[END OF LOOP]
Step 6: EXIT

Program(BFS):

```
from collections import defaultdict
class Graph:
    def __init__(self):
         self.graph = defaultdict(list)
     def addEdge(self,u,v):
        self.graph[u].append(v)
    def BFS(self, s):
        visited = [False] * (len(self.graph))
        queue = []
        queue.append(s)
        visited[s] = True
        while queue:
            s = queue.pop(0)
            print (s, end = " ")
            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True
```

```
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print ("Following is Breadth First Traversal"
            " (starting from vertex 2)")
g.BFS(2)
```

Output(BFS):

Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1

Program(DFS):

```
from collections import defaultdict
class Graph:
   def __init__(self):
      self.graph = defaultdict(list)
   def addEdge(self, u, v):
      self.graph[u].append(v)
   def DFSUtil(self, v, visited):
      visited.add(v)
      print(v, end=' ')
      for neighbour in self.graph[v]:
         if neighbour not in visited:
            self.DFSUtil(neighbour, visited)
   def DFS(self, v):
      visited = set()
      self.DFSUtil(v, visited)
if __name__ == "__main__":
   g = Graph()
   g.addEdge(0, 1)
   g.addEdge(0, 2)
   g.addEdge(1, 2)
   g.addEdge(2, 0)
   g.addEdge(2, 3)
   g.addEdge(3, 3)
   print("Following is DFS from (starting from vertex 2)")
   g.DFS(2)
```

Output(DFS):

Following is Depth First Traversal (starting from vertex 2)
2 0 1 3

Result:

Thus the uninformed search algorithms such as BFS and DFS have been executed successfully and the output got verified.

2. Implementation of Informed search algorithm (A*)

Aim:

To implement the informed search algorithm A*.

Algorithm(A*):

1. Initialize the open list
2. Initialize the closed list
put the starting node on the open
list (you can leave its f at zero)
3. while the open list is not empty
a) find the node with the least f on
the open list, call it "q"
b) pop q off the open list
c) generate q's 8 successors and set their
parents to q
d) for each successor
i) if successor is the goal, stop search
ii) else, compute both g and h for successor
successor.g = q.g + distance between successor and q
successor.h = distance from goal to successor (This can be done using many ways, we
will discuss three heuristics-
Manhattan, Diagonal and Euclidean Heuristics)
successor.f = successor.g + successor.h
iii) if a node with the same position as
successor is in the OPEN list which has a
lower f than successor, skip this successor
iv) if a node with the same position as
successor is in the CLOSED list which has
a lower f than successor, skip this successor
otherwise, add the node to the open list
end (for loop)
e) push q on the closed list
end (while loop)

Program(A*):

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
    parents = {}
    g[start_node] = 0
    parents[start_node] = start_node
    while len(open_set) > 0:
```

```
        n = None
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                if m not in open_set and m not in closed_set:
                    open_set.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_set:
                            closed_set.remove(m)
                            open_set.add(m)
        if n == None:
            print('Path does not exist!')
            return None
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
```

```
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
    'F': [('A', 3), ('G', 1), ('H', 7)],
    'G': [('F', 1), ('I', 3)],
    'H': [('F', 7), ('I', 2)],
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
}

aStarAlgo('A', 'J')
```

Output(A*):

Path found: ['A', 'F', 'G', 'I', 'J']

Result:

Thus the program to implement informed search algorithm have been executed successfully and output got verified.

3. Implement Naïve Bayes models.

Aim:

To diagnose heart patients and predict disease using heart disease dataset with Naïve Bayes Classifier Algorithm.

Algorithm:

Steps in Naïve Bayes Classifier Algorithm:

1. Read the training dataset T;
2. Calculate the mean and standard deviation of the predictor variables in each class;
3. Repeat Calculate the probability of fi using the gauss density equation in each class; Until the probability of all predictor variables (f1, f2, f3, .. , fn) has been calculated.
4. Calculate the likelihood for each class;
5. Get the greatest likelihood;

Program:

NB_from_scratch.py

```
import csv
import numpy as np
from sklearn.metrics import confusion_matrix, f1_score, roc_curve, auc
import matplotlib.pyplot as plt
from itertools import cycle
from scipy import interp
import warnings
import random
import math

# convert txt file to csv
with open('heartdisease.txt', 'r') as in_file:
   stripped = (line.strip() for line in in_file)
   lines = (line.split(",") for line in stripped if line)
   with open('heartdisease.csv', 'w', newline='') as out_file:
      writer = csv.writer(out_file)
      writer.writerow(('age', 'sex', 'cp', 'restbp', 'chol', 'fbs', 'restecg',
                   'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num'))
      writer.writerows(lines)

warnings.filterwarnings("ignore")
```

```python
# Example of Naive Bayes implemented from Scratch in Python

# calculating mean of column values belonging to one class
def mean(columnvalues):
    s = 0
    n = float(len(columnvalues))
    for i in range(len(columnvalues)):
        s = s + float(columnvalues[i])
    return s / n


# calculating standard deviation of column values belonging to one class
def stdev(columnvalues):
    avg = mean(columnvalues)
    s = 0.0
    num = len(columnvalues)
    for i in range(num):
        s = s + pow(float(columnvalues[i]) - avg, 2)
    variance = s / (float(num - 1))
    return math.sqrt(variance)


# Reading CSV file
filename = 'heartdisease.csv'
lines = csv.reader(open(filename, "r"))
dataset = list(lines)
for i in range(len(dataset) - 1):
    dataset[i] = [float(x) for x in dataset[i + 1]]

for z in range(5):
    print("\n\n\nTest Train Split no. ", z + 1, "\n\n\n")
    trainsize = int(len(dataset) * 0.75)
    trainset = []
    testset = list(dataset)
    for i in range(trainsize):
        index = random.randrange(len(testset))
        trainset.append(testset.pop(index))

    # separate list according to class
    classlist = {}
    for i in range(len(dataset)):
        class_num = float(dataset[i][-1])
        row = dataset[i]
        if (class_num not in classlist):
            classlist[class_num] = []
        classlist[class_num].append(row)
```

```
    # preparing data class wise
    class_data = {}
    for class_num, row in classlist.items():
        class_datarow = [(mean(columnvalues), stdev(columnvalues)) for columnvalues in
zip(*row)]
        class_datrow = class_datarow[0:13]
        class_data[class_num] = class_datarow

        # Getting test vector
    y_test = []
    for j in range(len(testset)):
        y_test.append(testset[j][-1])

        # Getting prediction vector
    y_pred = []
    for i in range(len(testset)):
        class_probability = {}
        for class_num, row in class_data.items():
            class_probability[class_num] = 1
            for j in range(len(row)):
                calculated_mean, calculated_dev = row[j]
                x = float(testset[i][j])
                if (calculated_dev != 0):
                    power = math.exp(-(math.pow(x - calculated_mean, 2) / (2 *
math.pow(calculated_dev, 2))))
                    probability = (1 / (math.sqrt(2 * math.pi) * calculated_dev)) * power
                class_probability[class_num] *= probability

        resultant_class, max_prob = -1, -1
        for class_num, probability in class_probability.items():
            if resultant_class == -1 or probability > max_prob:
                max_prob = probability
                resultant_class = class_num

        y_pred.append(resultant_class)

    # Getting Accuracy
    count = 0
    for i in range(len(testset)):
        if testset[i][-1] == y_pred[i]:
            count += 1
    accuracy = (count / float(len(testset))) * 100.0
    print("\n\n Accuracy: ", accuracy, "%")

    y1 = [float(k) for k in y_test]
    y_pred1 = [float(k) for k in y_pred]
```

```
print("\n\n\n\nConfusion Matrix")
cf_matrix = confusion_matrix(y1, y_pred1)
print(cf_matrix)

print("\n\n\n\nF1 Score")
f_score = f1_score(y1, y_pred1, average='weighted')
print(f_score)

# Matrix from 1D array
y2 = np.zeros(shape=(len(y1), 5))
y3 = np.zeros(shape=(len(y_pred1), 5))
for i in range(len(y1)):
    y2[i][int(y1[i])] = 1

for i in range(len(y_pred1)):
    y3[i][int(y_pred1[i])] = 1

# ROC Curve generation
n_classes = 5

fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y2[:, i], y3[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y2.ravel(), y3.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Compute macro-average ROC curve and ROC area
print("\n\n\n\nROC Curve")
# First aggregate all false positive rates
lw = 2
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
```

```
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    # Plot all ROC curves
    plt.figure()
    plt.plot(fpr["micro"], tpr["micro"],
             label='micro-average (area = {0:0.2f})'
                   ''.format(roc_auc["micro"]),
             color='deeppink', linestyle=':', linewidth=4)

    plt.plot(fpr["macro"], tpr["macro"],
             label='macro-average (area = {0:0.2f})'
                   ''.format(roc_auc["macro"]),
             color='navy', linestyle=':', linewidth=4)

    colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'red', 'black'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=lw,
                 label='ROC of class {0} (area = {1:0.2f})'
                       ''.format(i, roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--', lw=lw)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic for multi-class')
    plt.legend(loc="lower right")
    plt.savefig('Exp-8')
    plt.show()
```

NB_from_Gaussian_Sklearn.py

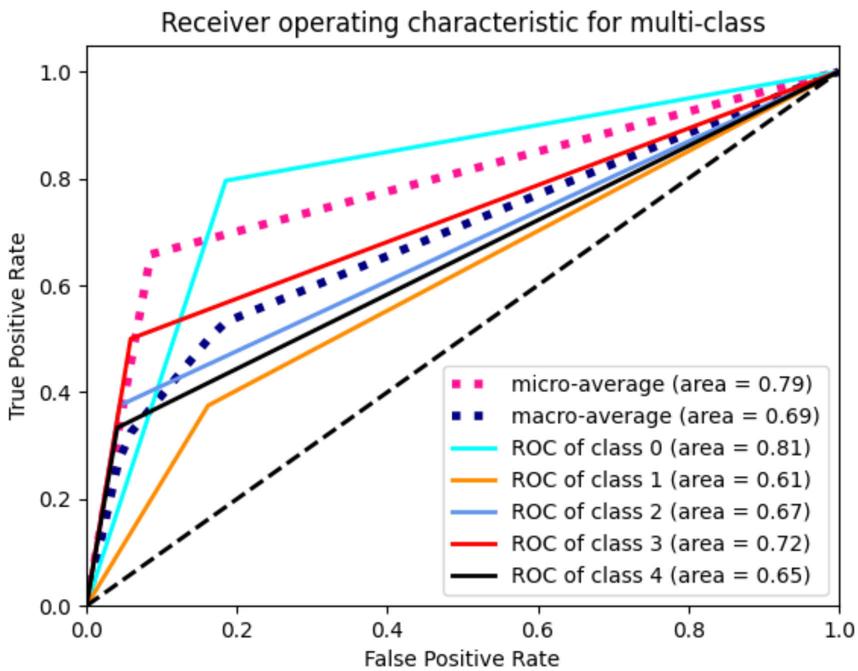
```
import csv
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix, f1_score, roc_curve, auc
import matplotlib.pyplot as plt
from itertools import cycle
from scipy import interp

# converting txt file to csv file
with open('heartdisease.txt', 'r') as in_file:
    stripped = (line.strip() for line in in_file)
    lines = (line.split(",") for line in stripped if line)
```

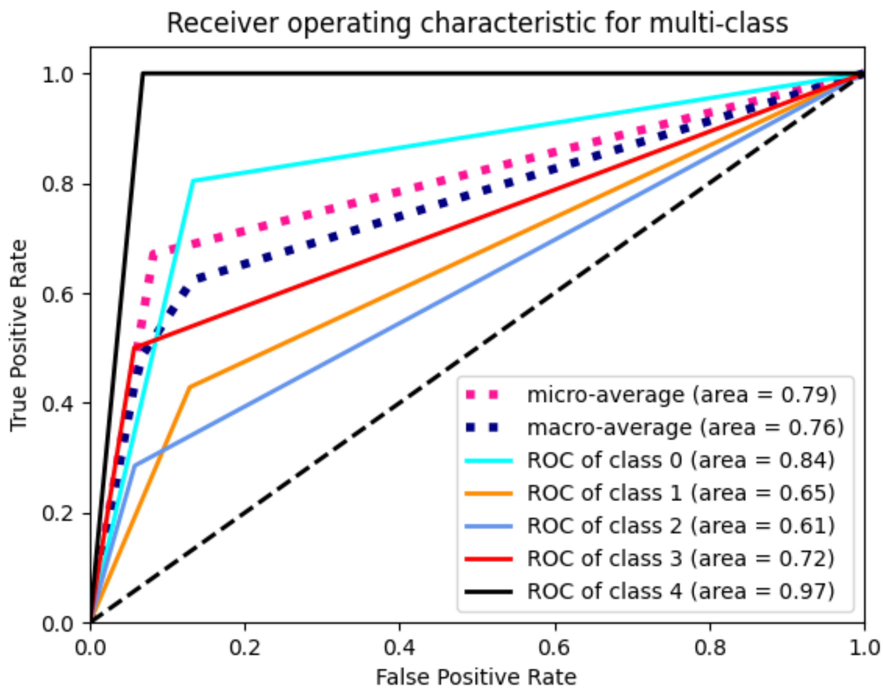
```
    with open('heartdisease.csv', 'w') as out_file:
        writer = csv.writer(out_file)
        writer.writerow(('age', 'sex', 'cp', 'restbp', 'chol', 'fbs', 'restecg',
                         'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num'))
        writer.writerows(lines)

# reading CSV using Pandas and storing in dataframe
df = pd.read_csv('heartdisease.csv', header=None)

training_x = df.iloc[1:df.shape[0], 0:13]
# print(training_set)

training_y = df.iloc[1:df.shape[0], 13:14]
# print(testing_set)


# converting dataframe into arrays
x = np.array(training_x)
y = np.array(training_y)

for z in range(5):
    print("\n\n\nTest Train Split no. ", z + 1, "\n\n\n")
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=None)
    # Gaussian function of sklearn
    gnb = GaussianNB()
    gnb.fit(x_train, y_train.ravel())
    y_pred = gnb.predict(x_test)

    print("\n\nGaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test,
y_pred) * 100)

    # convert 2D array to 1D array
    y1 = y_test.ravel()
    y_pred1 = y_pred.ravel()

    print("\n\n\n\nConfusion Matrix")
    cf_matrix = confusion_matrix(y1, y_pred1)
    print(cf_matrix)

    print("\n\n\n\nF1 Score")
    f_score = f1_score(y1, y_pred1, average='weighted')
    print(f_score)

    # Matrix from 1D array
    y2 = np.zeros(shape=(len(y1), 5))
    y3 = np.zeros(shape=(len(y_pred1), 5))
    for i in range(len(y1)):
```

```
    y2[i][int(y1[i])] = 1

for i in range(len(y_pred1)):
    y3[i][int(y_pred1[i])] = 1

# ROC Curve generation
n_classes = 5

fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y2[:, i], y3[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y2.ravel(), y3.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Compute macro-average ROC curve and ROC area
print("\n\n\n\nROC Curve")
# First aggregate all false positive rates
lw = 2
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
```

```
        color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'red', 'black'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC of class {0} (area = {1:0.2f})'
                   ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic for multi-class')
plt.legend(loc="lower right")
plt.show()
```

Output:





Result:

Thus the program to diagnose heart patients and predict disease using heart disease dataset with Naïve Bayes Classifier Algorithm have been executed successfully and output got verified.

4. Implement Bayesian Networks

Aim:

To construct a Bayesian network, to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Algorithm:

1. Read the training dataset T;
2. Calculate the mean and standard deviation of the predictor variables in each class;
3. Repeat Calculate the probability of fi using the gauss density equation in each class; Until the probability of all predictor variables (f1, f2, f3, .. , fn) has been calculated.
4. Calculate the likelihood for each class;
5. Get the greatest likelihood;

Program:

```
import bayespy as bp
import numpy as np
import csv
from colorama import init
from colorama import Fore, Back, Style
init()

ageEnum = {'SuperSeniorCitizen':0, 'SeniorCitizen':1, 'MiddleAged':2, 'Youth':3, 'Teen':4}
genderEnum = {'Male':0, 'Female':1}
familyHistoryEnum = {'Yes':0, 'No':1}
dietEnum = {'High':0, 'Medium':1, 'Low':2}
lifeStyleEnum = {'Athlete':0, 'Active':1, 'Moderate':2, 'Sedetary':3}
cholesterolEnum = {'High':0, 'BorderLine':1, 'Normal':2}
heartDiseaseEnum = {'Yes':0, 'No':1}

with open('heart_disease_data.csv') as csvfile:
  lines = csv.reader(csvfile)
  dataset = list(lines)
  data = []
  for x in dataset:
       data.append([ageEnum[x[0]],genderEnum[x[1]],familyHistoryEnum[x[2]],dietEnum[x
       [3]],lifeStyleEnum[x[4]],cholesterolEnum[x[5]],heartDiseaseEnum[x[6]]])
data = np.array(data)
N = len(data)

p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:,0])
```

```
p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:,1])

p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:,2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
diet = bp.nodes.Categorical(p_diet, plates=(N,))
diet.observe(data[:,3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:,4])

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

p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.MultiMixture([age, gender, familyhistory, diet, lifestyle,
cholesterol], bp.nodes.Categorical, p_heartdisease)
heartdisease.observe(data[:,6])
p_heartdisease.update()

m = 0
while m == 0:
   print("\n")
   res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter
          Gender: ' + str(genderEnum))), int(input('Enter FamilyHistory: ' +
          str(familyHistoryEnum))), int(input('Enter dietEnum: ' + str(dietEnum))),
          int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' +
          str(cholesterolEnum)))], bp.nodes.Categorical,
          p_heartdisease).get_moments()[0][heartDiseaseEnum['Yes']]
   print("Probability(HeartDisease) = " +  str(res))
   m = int(input("Enter for Continue:0, Exit :1  "))
```

Output:

```
Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}1
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}2
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}2
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}1
```

Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 1

Result:

Thus the program to implement a bayesian networks in the given heart disease dataset have been executed successfully and the output got verified.

5. Build Regression models

Aim:

To build regression models such as locally weighted linear regression and plot the necessary graphs.

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 :

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

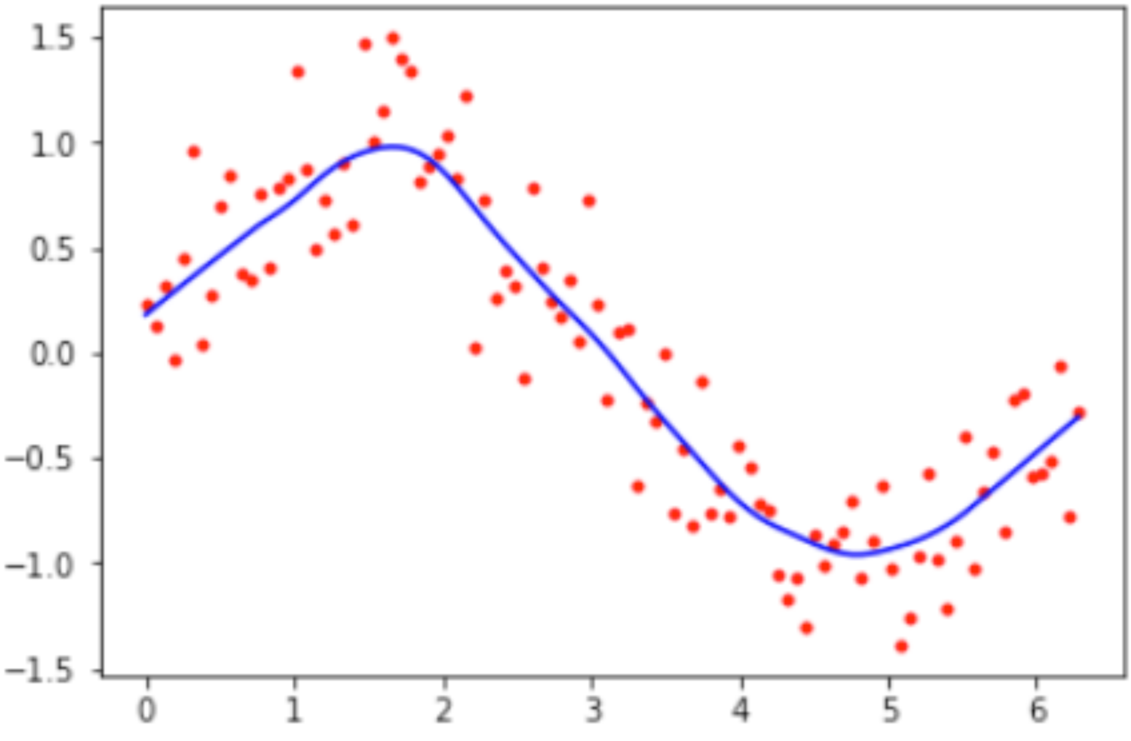
6. Prediction = x0*β.

Program:

```
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:



Result:

Thus the program to implement non-parametric Locally Weighted Regression algorithm in order to fit data points with a graph visualization have been executed successfully.

6. Build decision trees and random forests.

Aim:

To implement the concept of decision trees with suitable dataset from real world problems using CART algorithm.

Algorithm:

Steps in CART algorithm:

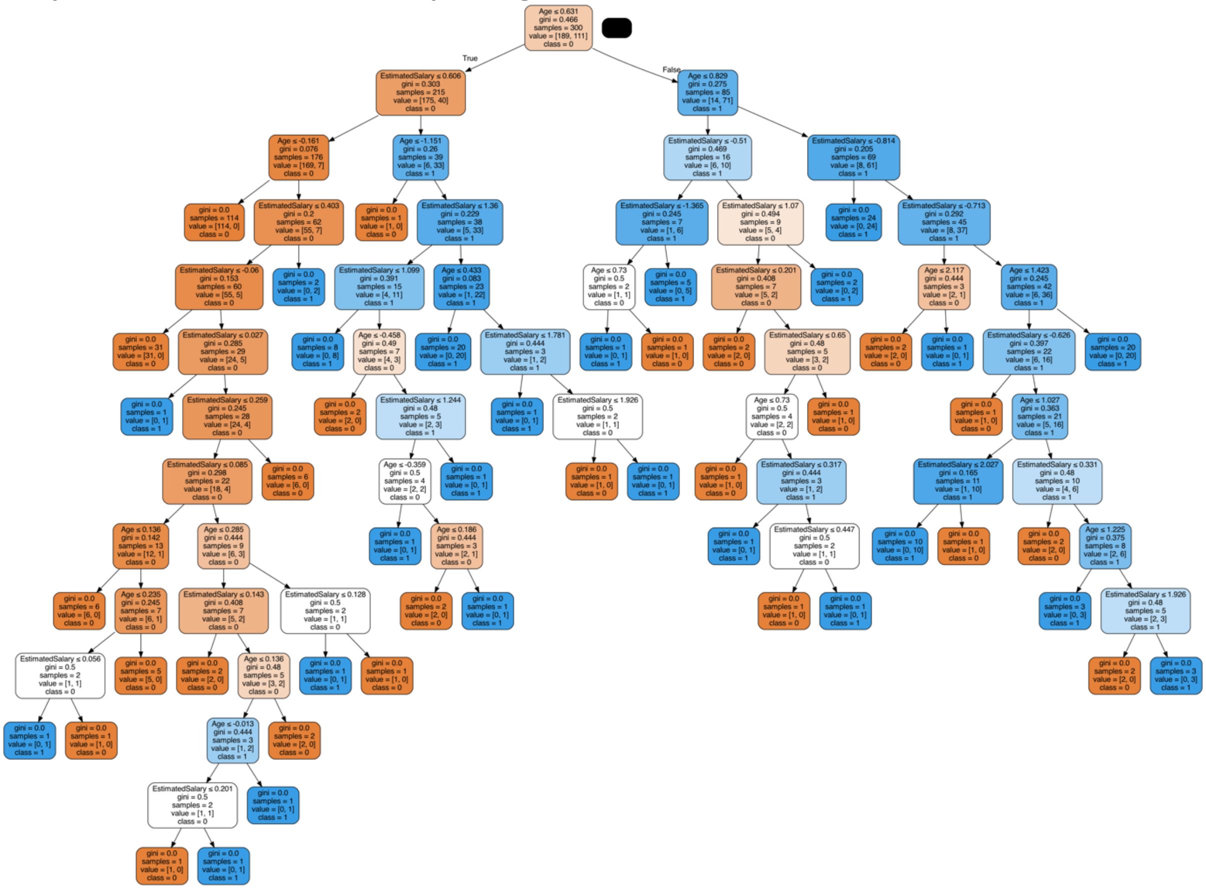
1. It begins with the original set S as the root node.
2. On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates Gini index of this attribute.
3. Gini Index works with the categorical target variable “Success” or “Failure”. It performs only Binary splits.
4. The set S is then split by the selected attribute to produce a subset of the data.
5. The algorithm continues to recur on each subset, considering only attributes never selected before.

Program:

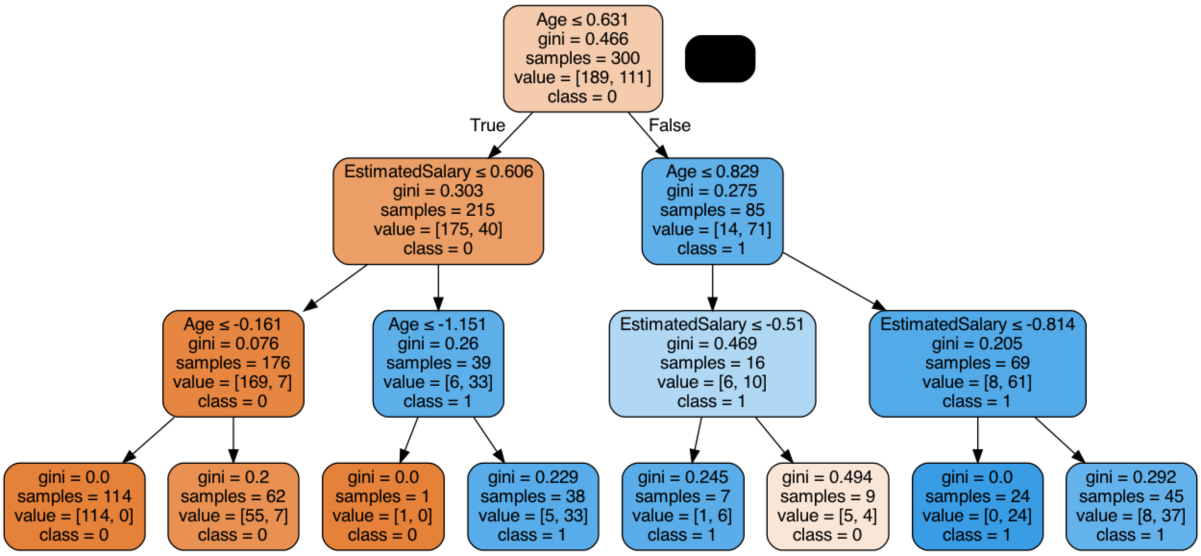
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

data =
pd.read_csv('/Users/ganesh/PycharmProjects/DecisionTree/Social_Network_Ads.csv')
data.head()

feature_cols = ['Age', 'EstimatedSalary']
x = data.iloc[:, [2, 3]].values
y = data.iloc[:, 4].values

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)

from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
x_train = sc_x.fit_transform(x_train)
x_test = sc_x.transform(x_test)

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier = classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)
```

```
from sklearn import metrics
print('Accuracy Score:', metrics.accuracy_score(y_test, y_pred))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test

x1, x2 = np.meshgrid(np.arange(start=x_set[:, 0].min()-1, stop=x_set[:, 0].max()+1,
step=0.01), np.arange(start=x_set[:, 1].min()-1, stop=x_set[:, 1].max()+1, step=0.01))
plt.contourf(x1,x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha=0.75, cmap=ListedColormap(("red", "green")))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c=ListedColormap(("red", "green"))(i),
label=j)

plt.title("Decision Tree(Test set)")
plt.xlabel("Age")
plt.ylabel("Estimated Salary")
plt.legend()
plt.show()

from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(classifier, out_file=dot_data, filled=True, rounded=True,
special_characters=True, feature_names=feature_cols, class_names=['0', '1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.write_png('decisiontree.png'))

classifier = DecisionTreeClassifier(criterion="gini", max_depth=3)
classifier = classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))

dot_data = StringIO()
export_graphviz(classifier, out_file=dot_data, filled=True, rounded=True,
special_characters=True, feature_names=feature_cols, class_names=['0', '1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.write_png('opt_decisiontree_gini.png'))
```

Output of decision tree without pruning:



Optimized output of decision tree using Gini Index (CART):



Result:

Thus the program to implement the concept of decision trees with suitable dataset from real world problems using CART algorithm have been executed successfully.

7. Build SVM models.

Aim:

To create a machine learning model which classifies the Spam and Ham E-Mails from a given dataset using Support Vector Machine algorithm.

Algorithm:

1. Import all the necessary libraries.
2. Read the given csv file which contains the emails which are both spam and ham.
3. Gather all the words given in that dataset and Identify the stop words with a mean distribution.
4. Create an ML model using the Support Vector Classifier after splitting the dataset into training and test set.
5. Display the accuracy and f1 score and print the confusion matrix for the classification of spam and ham.

Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
from nltk.corpus import stopwords
import os
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from PIL import Image
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_curve, auc
from sklearn import metrics
from sklearn import model_selection
from sklearn import svm
from nltk import word_tokenize
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
from sklearn.metrics import plot_confusion_matrix

class data_read_write(object):
    def __init__(self):
        pass
    def __init__(self, file_link):
        self.data_frame =  pd.read_csv(file_link)
```

```
    def read_csv_file(self, file_link):
        return self.data_frame
    def write_to_csvfile(self, file_link):
        self.data_frame.to_csv(file_link, encoding='utf-8', index=False, header=True)
        return

class generate_word_cloud(data_read_write):
    def __init__(self):
        pass
    def variance_column(self, data):
        return np.variance(data)
    def word_cloud(self, data_frame_column, output_image_file):
        text = " ".join(review for review in data_frame_column)
        stopwords = set(STOPWORDS)
        stopwords.update(["subject"])
        wordcloud = WordCloud(width = 1200, height = 800, stopwords=stopwords,
                              max_font_size = 50, margin=0,
                              background_color = "white").generate(text)
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis("off")
        plt.savefig("Distribution.png")
        plt.show()
        wordcloud.to_file(output_image_file)
        return


class data_cleaning(data_read_write):
    def __init__(self):
        pass
    def message_cleaning(self, message):
        Test_punc_removed = [char for char in message if char not in string.punctuation]
        Test_punc_removed_join = ''.join(Test_punc_removed)
        Test_punc_removed_join_clean = [word for word in Test_punc_removed_join.split()
                                        if word.lower() not in stopwords.words('english')]
        final_join = ' '.join(Test_punc_removed_join_clean)
        return final_join

    def apply_to_column(self, data_column_text):
        data_processed = data_column_text.apply(self.message_cleaning)
        return data_processed


class apply_embeddding_and_model(data_read_write):
    def __init__(self):
        pass

    def apply_count_vector(self, v_data_column):
```

```
        vectorizer = CountVectorizer(min_df=2, analyzer="word", tokenizer=None,
preprocessor=None, stop_words=None)
        return vectorizer.fit_transform(v_data_column)



    def apply_svm(self, X, y):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        params = {'kernel': 'linear', 'C': 2, 'gamma': 1}
        svm_cv = svm.SVC(C=params['C'], kernel=params['kernel'], gamma=params['gamma'],
probability=True)
        svm_cv.fit(X_train, y_train)
        y_predict_test = svm_cv.predict(X_test)
        cm = confusion_matrix(y_test, y_predict_test)
        sns.heatmap(cm, annot=True)
        print(classification_report(y_test, y_predict_test))
        print("test set")

        print("\nAccuracy Score: " + str(metrics.accuracy_score(y_test, y_predict_test)))
        print("F1 Score: " + str(metrics.f1_score(y_test, y_predict_test)))
        print("Recall: " + str(metrics.recall_score(y_test, y_predict_test)))
        print("Precision: " + str(metrics.precision_score(y_test, y_predict_test)))

        class_names = ['ham', 'spam']
        titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
        for title, normalize in titles_options:
            disp = plot_confusion_matrix(svm_cv, X_test, y_test,
                               display_labels=class_names,
                               cmap=plt.cm.Blues,
                               normalize=normalize)
            disp.ax_.set_title(title)
            print(title)
            print(disp.confusion_matrix)
        plt.savefig("SVM.png")
        plt.show()

        ns_probs = [0 for _ in range(len(y_test))]
        lr_probs = svm_cv.predict_proba(X_test)
        lr_probs = lr_probs[:, 1]
        ns_auc = roc_auc_score(y_test, ns_probs)
        lr_auc = roc_auc_score(y_test, lr_probs)
        print('No Skill: ROC AUC=%.3f' % (ns_auc))
        print('SVM: ROC AUC=%.3f' % (lr_auc))
        ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
        lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
        pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
```

```
    pyplot.plot(lr_fpr, lr_tpr, marker='.', label='SVM')
    pyplot.xlabel('False Positive Rate')
    pyplot.ylabel('True Positive Rate')
    pyplot.legend()
    pyplot.savefig("SVMMat.png")
    pyplot.show()
    return

data_obj = data_read_write("emails.csv")

data_frame = data_obj.read_csv_file("processed.csv")
data_frame.head()
data_frame.tail()
data_frame.describe()
data_frame.info()

data_frame.head()

data_frame.groupby('spam').describe()

data_frame['length'] = data_frame['text'].apply(len)
data_frame['length'].max()

sns.set(rc={'figure.figsize':(11.7,8.27)})
ham_messages_length =  data_frame[data_frame['spam']==0]
spam_messages_length =  data_frame[data_frame['spam']==1]

ham_messages_length['length'].plot(bins=100, kind='hist',label = 'Ham')
spam_messages_length['length'].plot(bins=100, kind='hist',label = 'Spam')
plt.title('Distribution of Length of Email Text')
plt.xlabel('Length of Email Text')
plt.legend()

data_frame[data_frame['spam']==0].text.values

ham_words_length = [len(word_tokenize(title)) for title in
data_frame[data_frame['spam']==0].text.values]
spam_words_length = [len(word_tokenize(title)) for title in
data_frame[data_frame['spam']==1].text.values]
print(max(ham_words_length))
print(max(spam_words_length))

sns.set(rc={'figure.figsize':(11.7,8.27)})
ax = sns.distplot(ham_words_length, norm_hist = True, bins = 30, label = 'Ham')
ax = sns.distplot(spam_words_length, norm_hist = True, bins = 30, label = 'Spam')
plt.title('Distribution of Number of Words')
plt.xlabel('Number of Words')
```

```
plt.legend()
plt.savefig("SVMGraph.png")
plt.show()

def mean_word_length(x):
    word_lengths = np.array([])
    for word in word_tokenize(x):
        word_lengths = np.append(word_lengths, len(word))
    return word_lengths.mean()

ham_meanword_length =
data_frame[data_frame['spam']==0].text.apply(mean_word_length)
spam_meanword_length =
data_frame[data_frame['spam']==1].text.apply(mean_word_length)


sns.distplot(ham_meanword_length, norm_hist = True, bins = 30, label = 'Ham')
sns.distplot(spam_meanword_length , norm_hist = True, bins = 30, label = 'Spam')
plt.title('Distribution of Mean Word Length')
plt.xlabel('Mean Word Length')
plt.legend()
plt.savefig("Graph.png")
plt.show()

from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))


def stop_words_ratio(x):
    num_total_words = 0
    num_stop_words = 0
    for word in word_tokenize(x):
        if word in stop_words:
            num_stop_words += 1
        num_total_words += 1
    return num_stop_words / num_total_words


ham_stopwords = data_frame[data_frame['spam'] == 0].text.apply(stop_words_ratio)
spam_stopwords = data_frame[data_frame['spam'] == 1].text.apply(stop_words_ratio)

sns.distplot(ham_stopwords, norm_hist=True, label='Ham')
sns.distplot(spam_stopwords, label='Spam')

print('Ham Mean: {:.3f}'.format(ham_stopwords.values.mean()))
print('Spam Mean: {:.3f}'.format(spam_stopwords.values.mean()))
plt.title('Distribution of Stop-word Ratio')
```

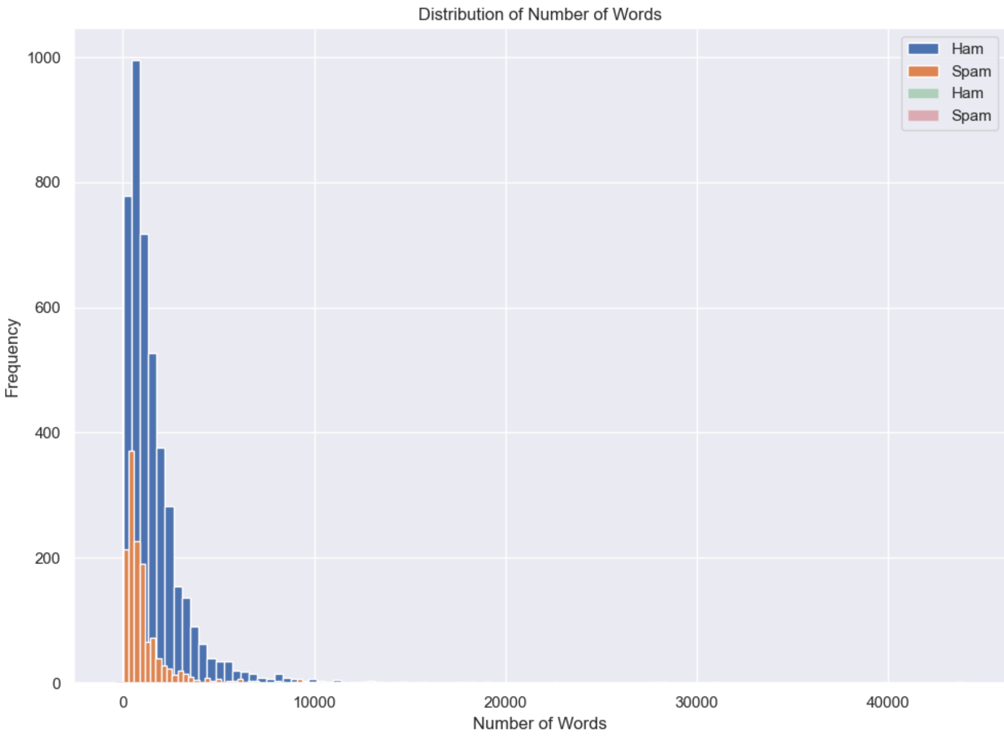
```
plt.xlabel('Stop Word Ratio')
plt.legend()

ham = data_frame[data_frame['spam']==0]
spam = data_frame[data_frame['spam']==1]
spam['length'].plot(bins=60, kind='hist')
ham['length'].plot(bins=60, kind='hist')
data_frame['Ham(0) and Spam(1)'] = data_frame['spam']
print( 'Spam percentage =', (len(spam) / len(data_frame) )*100,"%")
print( 'Ham percentage =', (len(ham) / len(data_frame) )*100,"%")
sns.countplot(data_frame['Ham(0) and Spam(1)'], label = "Count")

data_clean_obj = data_cleaning()
data_frame['clean_text'] = data_clean_obj.apply_to_column(data_frame['text'])

data_frame.head()

data_obj.data_frame.head()

data_obj.write_to_csvfile("processed_file.csv")

cv_object = apply_embeddding_and_model()
spamham_countvectorizer = cv_object.apply_count_vector(data_frame['clean_text'])
X = spamham_countvectorizer
label = data_frame['spam'].values
y = label
cv_object.apply_svm(X,y)
```

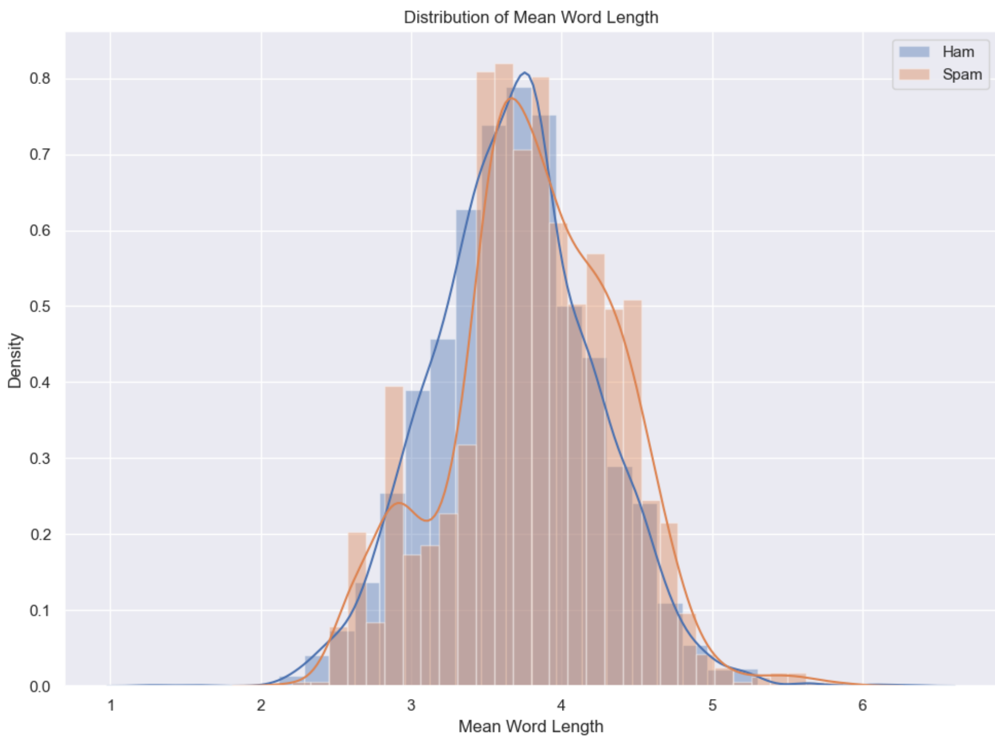
Output:

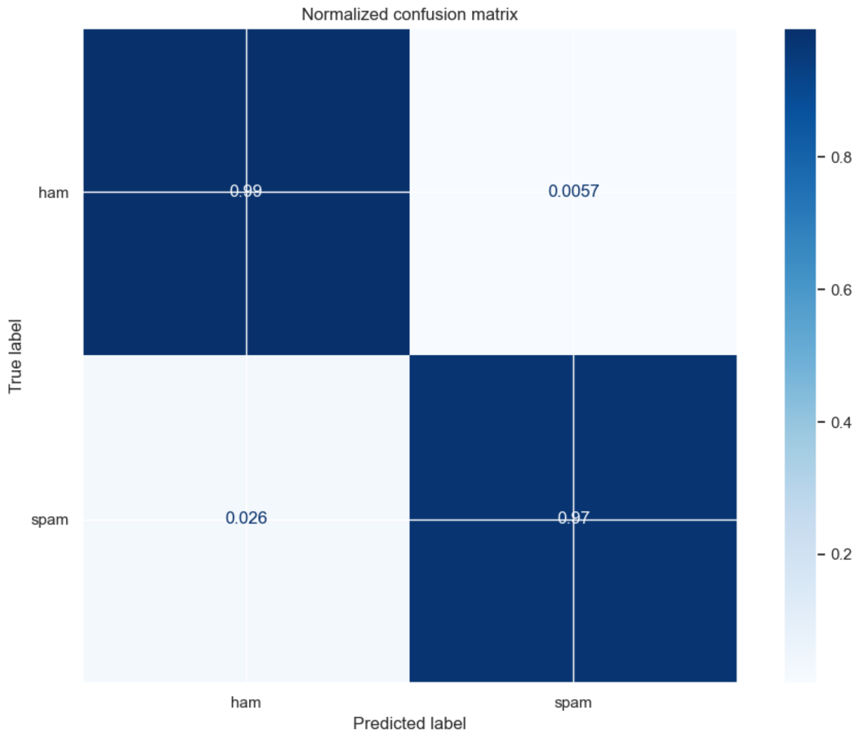
	precision	**recall**	**f1-score**	**support**
0	0.99	0.99	0.99	877
1	0.98	0.97	0.98	269
accuracy			0.99	1146
macro avg	0.99	0.98	0.99	1146
weighted avg	0.99	0.99	0.99	1146

test set
Accuracy Score: 0.9895287958115183
F1 Score: 0.9776119402985075
Recall: 0.9739776951672863
Precision: 0.9812734082397003

Normalized confusion matrix
[[0.99429875 0.00570125]
 [0.0260223 0.9739777]]







Result:

Thus the program to create a machine learning model which classifies the Spam and Ham E-Mails from a given dataset using Support Vector Machine algorithm have been successfully executed.

8. Implement ensembling techniques.

Aim:

To implement the ensembling technique of Blending with the given Alcohol QCM Dataset.

Algorithm:

1. Split the training dataset into train, test and validation dataset.
2. Fit all the base models using train dataset.
3. Make predictions on validation and test dataset.
4. These predictions are used as features to build a second level model
5. This model is used to make predictions on test and meta-features.

Program:

```
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
df = pd.read_csv("train_data.csv")
target = df["target"]
train = df.drop("target")
X_train, X_test, y_train, y_test = train_test_split(train, target, test_size=0.20)
train_ratio = 0.70
validation_ratio = 0.20
test_ratio = 0.10
x_train, x_test, y_train, y_test = train_test_split(
   train, target, test_size=1 - train_ratio)
x_val, x_test, y_val, y_test = train_test_split(
   x_test, y_test, test_size=test_ratio/(test_ratio + validation_ratio))
model_1 = LinearRegression()
model_2 = xgb.XGBRegressor()
model_3 = RandomForestRegressor()
model_1.fit(x_train, y_train)
val_pred_1 = model_1.predict(x_val)
test_pred_1 = model_1.predict(x_test)
val_pred_1 = pd.DataFrame(val_pred_1)
test_pred_1 = pd.DataFrame(test_pred_1)
model_2.fit(x_train, y_train)
val_pred_2 = model_2.predict(x_val)
test_pred_2 = model_2.predict(x_test)
val_pred_2 = pd.DataFrame(val_pred_2)
test_pred_2 = pd.DataFrame(test_pred_2)
```

```
model_3.fit(x_train, y_train)
val_pred_3 = model_1.predict(x_val)
test_pred_3 = model_1.predict(x_test)
val_pred_3 = pd.DataFrame(val_pred_3)
test_pred_3 = pd.DataFrame(test_pred_3)
df_val = pd.concat([x_val, val_pred_1, val_pred_2, val_pred_3], axis=1)
df_test = pd.concat([x_test, test_pred_1, test_pred_2, test_pred_3], axis=1)
final_model = LinearRegression()
final_model.fit(df_val, y_val)
final_pred = final_model.predict(df_test)
print(mean_squared_error(y_test, pred_final))
```

Output:

4790

Result:

Thus the program to implement ensembling technique of Blending with the given Alcohol QCM Dataset have been executed successfully and the output got verfied.

9. Implement clustering algorithms

Aim:

To implment k-Nearest Neighbour algorithm to classify the Iris Dataset.

Algorithm:

Step-1: Select the number K of the neighbors
Step-2: Calculate the Euclidean distance of K number of neighbors
Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
Step-4: Among these k neighbors, count the number of the data points in each category.
Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
Step-6: Our model is ready.

Program:

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

import pandas as pd
import numpy as np
from sklearn import datasets

iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target

x_train, x_test, y_train, y_test=(train_test_split(iris_data, iris_labels, test_size=0.20))
classifier=KNeighborsClassifier(n_neighbors=6)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)

print("accuracy is")
print(classification_report(y_test, y_pred))
```

Output:

accuracy is

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9
1	1.00	0.93	0.96	14
2	0.88	1.00	0.93	7
accuracy			0.97	30
macro avg	0.96	0.98	0.97	30
weighted avg	0.97	0.97	0.97	30

Result:

Thus the program to implement k-Nearest Neighbour Algorithm for clustering Iris dataset have been executed successfully and output got verified.

10. Implement EM for Bayesian networks.

Aim:

To implement the EM algorithm for clustering networks using the given dataset.

Algorithm:

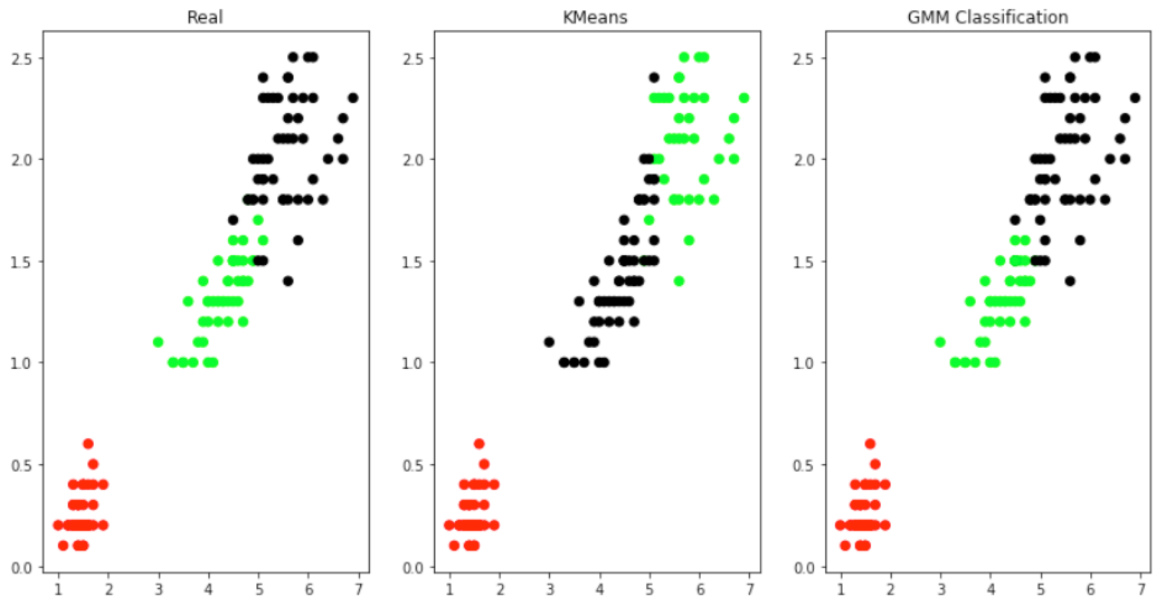
Initialize θ randomly Repeat until convergence:
E-step:
Compute q(h) = P(H = h | E = e; θ) for each h (probabilistic inference)
Create fully-observed weighted examples: (h, e) with weight q(h)
M-step:
Maximum likelihood (count and normalize) on weighted examples to get θ

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:



Result:

Thus the program to implement EM Algorithm for clustering networks using the given dataset have been executed successfully and the output got verified.

11. Build simple NN models.

Aim:

To implement the neural network model for the given dataset.

Algorithm:

1. Image Acquisition: The first step is to acquire images of paper documents with the help of optical scanners. This way, an original image can be captured and stored.
2. Pre-processing: The noise level on an image should be optimized and areas outside the text removed. Pre-processing is especially vital for recognizing handwritten documents that are more sensitive to noise.
3. Segmentation: The process of segmentation is aimed at grouping characters into meaningful chunks. There can be predefined classes for characters. So, images can be scanned for patterns that match the classes.
4. Feature Extraction: This step means splitting the input data into a set of features, that is, to find essential characteristics that make one or another pattern recognizable.
5. Training an MLP neural network using the following steps:
    1. Starting with the input layer, propagate data forward to the output layer. This step is the forward propagation.
    2. Based on the output, calculate the error (the difference between the predicted and known outcome). The error needs to be minimized.
    3. Backpropagate the error. Find its derivative with respect to each weight in the network, and update the model.
6. Post processing: This stage is the process of refinement as an OCR model can require some corrections. However, it isn't possible to achieve 100% recognition accuracy. The identification of characters heavily depends on the context.

Program:

```
from __future__ import print_function
import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers import Conv2D, MaxPooling2D, Flatten
from keras.optimizers import RMSprop, SGD
from keras.optimizers import Adam
from keras.utils import np_utils
from emnist import list_datasets
from emnist import extract_training_samples
from emnist import extract_test_samples
import matplotlib
matplotlib.use('TkAgg')
```

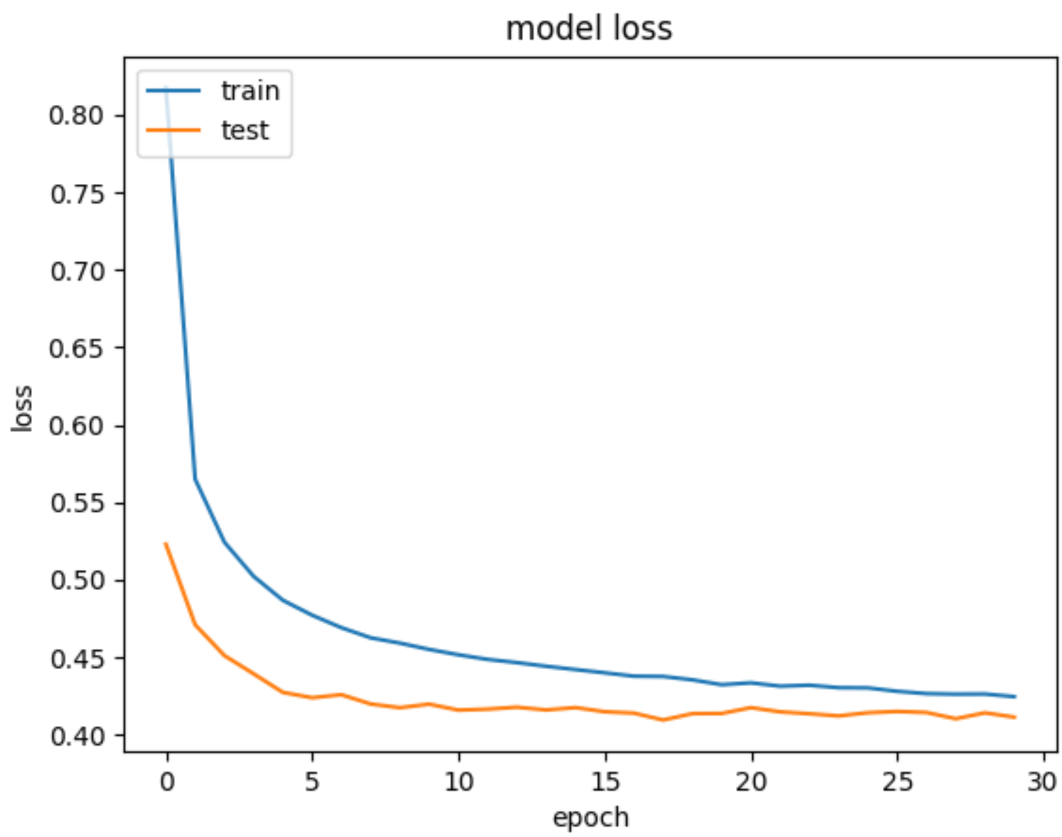
```
import matplotlib.pyplot as plt
np.random.seed(1671) # for reproducibility
# network and training
NB_EPOCH = 30
BATCH_SIZE = 256
VERBOSE = 2
NB_CLASSES = 256 # number of outputs = number of classes
OPTIMIZER = Adam()
N_HIDDEN = 512
VALIDATION_SPLIT=0.2 # how much TRAIN is reserved for VALIDATION
DROPOUT = 0.20
print(list_datasets())
X_train, y_train = extract_training_samples('byclass')
print("train shape: ", X_train.shape)
print("train labels: ",y_train.shape)
X_test, y_test = extract_test_samples('byclass')
print("test shape: ",X_test.shape)
print("test labels: ",y_test.shape)
#for indexing from 0
y_train = y_train-1
y_test = y_test-1
RESHAPED = len(X_train[0])*len(X_train[1])
X_train = X_train.reshape(len(X_train), RESHAPED)
X_test = X_test.reshape(len(X_test), RESHAPED)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalize
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)
# M_HIDDEN hidden layers
# 35 outputs
# final stage is softmax
model = Sequential()
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
```

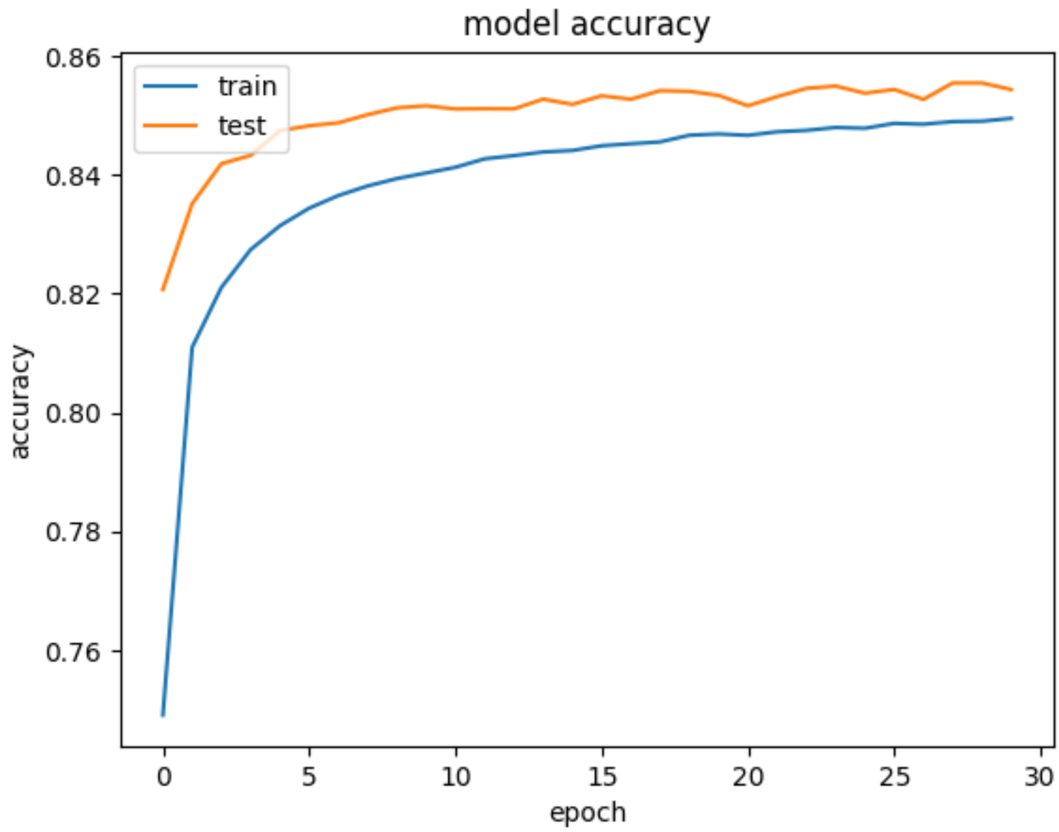
```
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()

model.compile(loss='categorical_crossentropy',
optimizer=OPTIMIZER,
metrics=['accuracy'])

history = model.fit(X_train, Y_train,
batch_size=BATCH_SIZE, epochs=NB_EPOCH,
verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("\nTest score:", score[0])
print('Test accuracy:', score[1])

# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Output:

```
['balanced', 'byclass', 'bymerge', 'digits', 'letters', 'mnist']
train shape: (697932, 28, 28)
train labels: (697932,)
test shape: (116323, 28, 28)
test labels: (116323,)
697932 train samples
116323 test samples
Model: "sequential"
```

Layer (type) Output Shape Param #
===
dense (Dense) (None, 512) 401920
activation (Activation) (None, 512) 0
dropout (Dropout) (None, 512) 0
dense_1 (Dense) (None, 256) 131328
activation_1 (Activation) (None, 256) 0
dropout_1 (Dropout) (None, 256) 0
dense_2 (Dense) (None, 256) 65792
activation_2 (Activation) (None, 256) 0
dropout_2 (Dropout) (None, 256) 0
dense_3 (Dense) (None, 256) 65792
activation_3 (Activation) (None, 256) 0
dropout_3 (Dropout) (None, 256) 0
dense_4 (Dense) (None, 256) 65792
activation_4 (Activation) (None, 256) 0
===
Total params: 730,624
Trainable params: 730,624
Non-trainable params: 0





Result:

Thus the program to implement the neural network model for the given dataset.

12. Build deep learning NN models.

Aim:

To implement and build a Convolutional neural network model which predicts the age and gender of a person using the given pre-trained models.

Algorithm:

Steps in CNN Algorithm:

Step-1: Choose the Dataset.
Step-2: Prepare the Dataset for training.
Step-3: Create training Data.
Step-4: Shuffle the Dataset.
Step-5: Assigning Labels and Features.
Step-6: Normalising X and converting labels to categorical data.
Step-7: Split X and Y for use in CNN.
Step-8: Define, compile and train the CNN Model.
Step-9: Accuracy and Score of the model.

Program:

```
import cv2 as cv
import math
import time
from google.colab.patches import cv2_imshow

def getFaceBox(net, frame, conf_threshold=0.7):
  frameOpencvDnn = frame.copy()
  frameHeight = frameOpencvDnn.shape[0]
  frameWidth = frameOpencvDnn.shape[1]
  blob = cv.dnn.blobFromImage(frameOpencvDnn, 1.0, (300, 300), [104, 117, 123], True,
False)   net.setInput(blob)
  detections = net.forward()
  bboxes = []
  for i in range(detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > conf_threshold:
      x1 = int(detections[0, 0, i, 3] * frameWidth)
      y1 = int(detections[0, 0, i, 4] * frameHeight)
      x2 = int(detections[0, 0, i, 5] * frameWidth)
      y2 = int(detections[0, 0, i, 6] * frameHeight)
      bboxes.append([x1, y1, x2, y2])
      cv.rectangle(frameOpencvDnn, (x1, y1), (x2, y2), (0, 255, 0),
int(round(frameHeight/150)), 8)
  return frameOpencvDnn, bboxes
```

```
faceProto = "/content/opencv_face_detector.pbtxt"
faceModel = "/content/opencv_face_detector_uint8.pb"
ageProto = "/content/age_deploy.prototxt"
ageModel = "/content/age_net.caffemodel"
genderProto = "/content/gender_deploy.prototxt"
genderModel = "/content/gender_net.caffemodel"

MODEL_MEAN_VALUES = (78.4263377603, 87.7689143744, 114.895847746)
ageList = ['(0-2)', '(4-6)', '(8-12)', '(15-20)', '(25-32)', '(38-43)', '(48-53)', '(60-100)']
genderList = ['Male', 'Female']

ageNet = cv.dnn.readNet(ageModel, ageProto)
genderNet = cv.dnn.readNet(genderModel, genderProto)
faceNet = cv.dnn.readNet(faceModel, faceProto)

def age_gender_detector(frame):
# Read frame
t = time.time()
frameFace, bboxes = getFaceBox(faceNet, frame)
for bbox in bboxes:
# print(bbox)
face = frame[max(0,bbox[1]-padding):min(bbox[3]+padding,frame.shape[0]-
1),max(0,bbox[0]-padding):min(bbox[2]+padding, frame.shape[1]-1)]blob =
cv.dnn.blobFromImage(face, 1.0, (227, 227), MODEL_MEAN_VALUES, swapRB=False)
 genderNet.setInput(blob)
 genderPreds = genderNet.forward()
 gender = genderList[genderPreds[0].argmax()]
 # print("Gender Output : {}".format(genderPreds))
 print("Gender : {}, conf = {:.3f}".format(gender,
genderPreds[0].max()))ageNet.setInput(blob)
agePreds = ageNet.forward()
age = ageList[agePreds[0].argmax()]
print("Age Output : {}".format(agePreds))
print("Age : {}, conf = {:.3f}".format(age, agePreds[0].max()))label = "{},{}".format(gender,
age)
cv.putText(frameFace, label, (bbox[0], bbox[1]-10), cv.FONT_HERSHEY_SIMPLEX, 0.8, (0,
255, 255), 2, cv.LINE_AA)
   return frameFace

from google.colab import files
uploaded = files.upload()
input = cv.imread("2.jpg")
output = age_gender_detector(input)
cv2_imshow(output)
```

Output:

gender : Male, conf = 1.000
Age Output : [[2.8247703e-05 8.9249297e-05 3.0017464e-04 8.8183772e-03 9.3055397e-01
5.1735926e-02 7.6946630e-03 7.7927281e-04]]
Age: (25-32), conf = 0.873.

Result:

Thus the program to implement and build a Convolutional neural network model which predicts the age and gender of a person using the given pre-trained models have been executed successfully and the output got verified.