POORNIMA

3/3/23

Experiment - 1

Aim -: Implement and demonstrate the FIND S algorithm for finding the most specific hypothesis based on a given set of training data samples.

Code -:

```
import pandas as pd
import numpy as np

df = pd.read_csv("Exp1.csv")
df

df.columns.values

data = np.array(df)[:, :-1]
data

target = np.array(df)[:, -1]
target

for i in range(len(data)):
    if target[i] == 'y':
        h = data[i].copy()
        break
h
```

Output →

	sky	temp	humidity	wind	play
	Sunny	warm	normal	strong	Y
	Sunny	warm	high	strong	Y
	rainy	cold	high	strong	n
	Sunny	warm	high	weak	Y

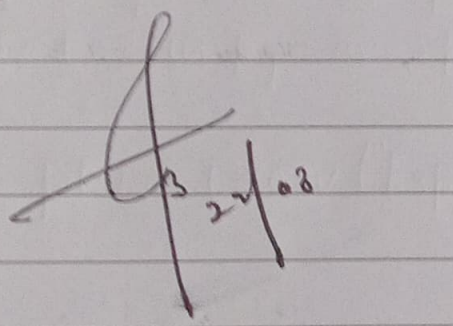
array (['y', 'y', 'n', 'y'], dtype = object)

array (['sunny', 'warm', 'normal', 'strong'], dtype = object)

array (['sunny', 'warm', '?', '?'], dtype = object)

```
for i in range(len(data)):
    if target[i] == 'y':
        for j in range(len(data[0])):
            if h[j] == data[i][j]:
                continue
            else:
                h[j] = "?"
```

h ←



POORNIMA

Experiment - 2

Aim -: Write a program to demonstrate the working of decision tree based ID3 algorithm.

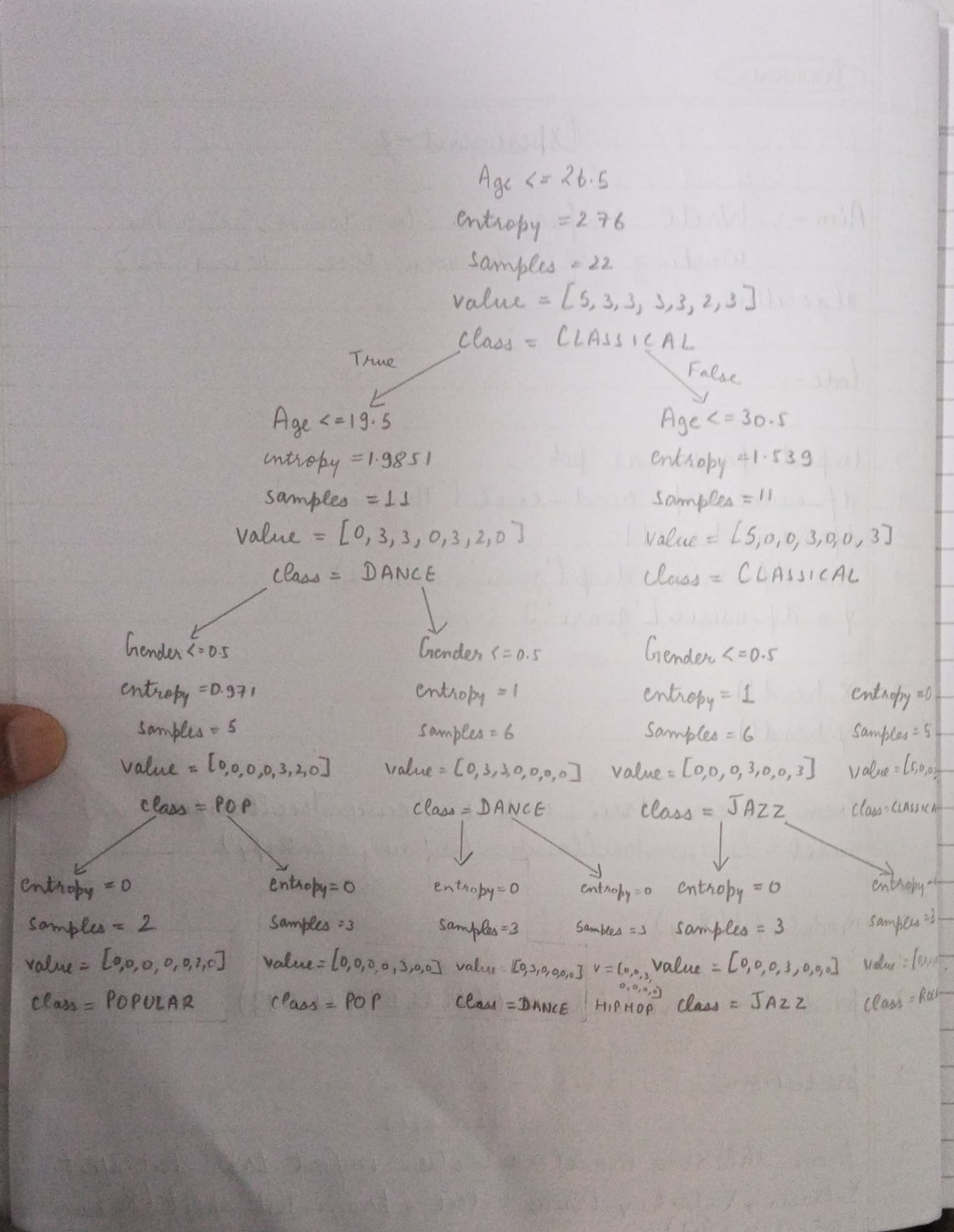
Code -:

```
import pandas as pd
df_music = pd.read_excel('Music.xlsx')
df_music.head()
X = df_music.drop('genre', axis = 1)
y = df_music['genre']

X.head()
y.head()

from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion = 'entropy')

model.fit(X, y)

prediction = model.predict([[23,1],[31,0]])

prediction

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2)
```

	Age	Grender	Genre
0	20	1	HIP HOP
1	24	1	HIP HOP
2	26	1	HIP HOP
3	27	1	ROCK
4	29	1	ROCK

	Age	Grender
0	20	1
1	24	1
2	26	1
3	27	1
4	29	1

0	HIP HOP
1	HIP HOP
2	HIP HOP
3	ROCK
4	ROCK


Age <= 26.5
entropy = 2.76
samples = 22
value = [5, 3, 3, 3, 3, 2, 3]
class = CLASSICAL
True
False
Age <= 19.5
entropy = 1.9851
samples = 11
value = [0, 3, 3, 0, 3, 2, 0]
class = DANCE
Age <= 30.5
entropy = 1.539
samples = 11
value = [5, 0, 0, 3, 0, 0, 3]
class = CLASSICAL
Gender <= 0.5
entropy = 0.971
samples = 5
value = [0, 0, 0, 0, 3, 2, 0]
class = POP
Gender <= 0.5
entropy = 1
samples = 6
value = [0, 3, 3, 0, 0, 0, 0]
class = DANCE
Gender <= 0.5
entropy = 1
Samples = 6
value = [0, 0, 0, 3, 0, 0, 3]
class = JAZZ
entropy = 0
Samples = 5
entropy = 0
samples = 2
value = [0, 0, 0, 0, 0, 2, 0]
class = POPULAR
entropy = 0
Samples = 3
value = [0, 0, 0, 0, 3, 0, 0]
class = POP
entropy = 0
samples = 3
value = [0, 3, 0, 0, 0, 0]
class = DANCE
entropy = 0
samples = 3
value = [0, 0, 0, 3, 0, 0, 0]
class = HIPHOP
entropy = 0
samples = 3
value = [0, 0, 0, 3, 0, 0, 0]
class = JAZZ

```
→ model.fit(X_train, y_train)

→ prediction = model.predict(X_test)

→ prediction

→ from sklearn.metrics import accuracy_score
  accuracy_score(y_test, prediction)

→ import joblib
→ joblib.dump(model, 'music-recommender')

→ model = joblib.load('music-recommender')
→ prediction = model.predict([[23,1],[31,0]])
→ prediction

→ from sklearn.tree import export_graphviz
→ export_graphviz(model, out_file='music-recommender.dot',
  feature_names=['Age','Gender'], class_names=sorted(y.unique())
  , label='all', rounded=True, filled=True)

→ import pydotplus

→ decision_tree = pydotplus.graph_from_dot_file('music-
  recommender.dot')

→ from IPython.display import Image
  Image(decision_tree.create_png())
```

POORNIMA

Exp-2

Aim:- for a given set of training data examples store in data.csv file, implement and demonstrate the candidate elimination algorithm to output a description of the set of all hypothesis consistent with examples

Candidate-Elimination Algorithm:-

- Load dataset
- Initialize general and specific hypothesis
- for each training example, if example is positive:
  - if attribute value == hypothesis value:
    - do nothing
  - else:
    - replace attribute-value with '?'
- If example is negative:
  - make general hypothesis more specific

Source code

```
import pandas as pd
import numpy as pd

data = Pd.read_csv("data.csv")
Print(data)
d = np.array(data)[:, :-1]
target = np.array(data)[:, -1]
```

O/P

	Time	weather	Temp	humidity	wind	Plays
0	Morning	sunny	Warm	mild	Normal	Yes
1	morning	Rainy	Cold	mild	strong	No
2	morning	sunny	moderate	high	Normal	Yes
3	Evening	sunny	Cold	High	Normal	Yes

The final specific hypothesis is: ['?' 'sunny' '?' '?' 'Normal']

The final generated hypothesis is: [['?' 'sunny' '?' '?' '?' '?'],['?' '?' '?' '?' '?' 'Normal']]

POORNIMA

```
def tecoin(t,f):
    for i, val in enumerate(t):
    if val == "Yes":
        specific_hypothesis = c[i].copy()
        break
    z = len(specific_hypothesis)
    general_hypothesis = [['?' for i in range[z]] for i in range(z)]

for i, val in enumerate(c):
    if t[i] == "Yes":
    for x in range[z]:
        if val[x] != specific_hypothesis[x]:
            specific_hypothesis[x] = '?'
            general_hypothesis[x][x] = '?'
    if t[i] == "No"
    for x in range(z):
        if val[x] != specific_hypothesis[x]:
            general_hypothesis[x][x] = specific_hypothesis[x]
        else:
            general_hypothesis[x][x] = '?'
indices = [i for i, val in enumerate(general_hypothesis) if val == ['?', '?', '?', '?', '?', '?']]
for i in indices:
    general_hypothesis.remove(['?', '?', '?', '?', '?', '?'])
Print("The final specific hypothesis is:", specific_hypothesis)
Print("The final general hypothesis is:", general_hypothesis)

tecoin(d, target)
```

Page No.....

POORNIMA

Exp-3

Aim:- write to program to demonstrate the working of the decision tree based ID3 Algorithm

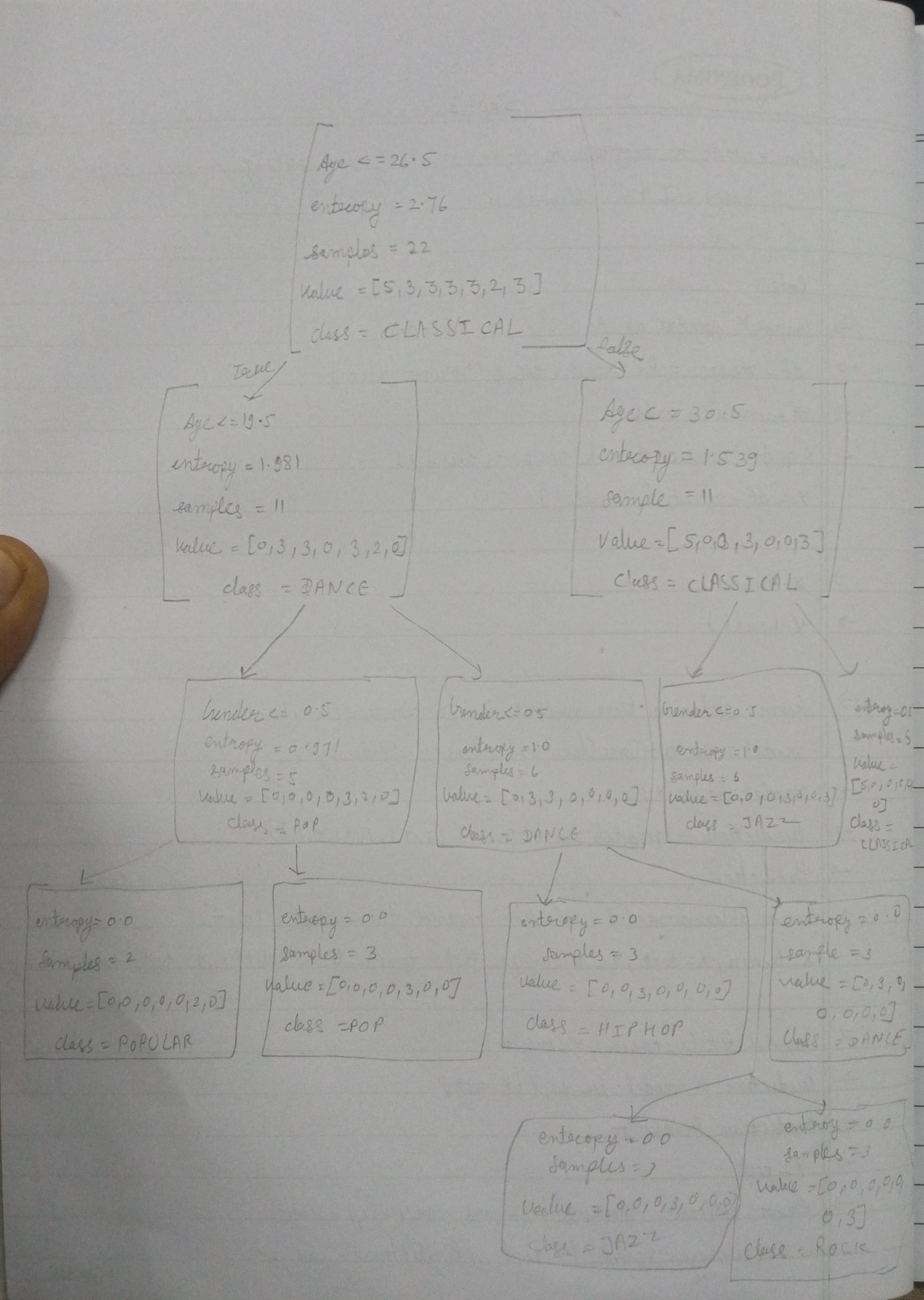
Code

```
→ import pandas as Pd
→ df_music = Pd.read_excel('music.xlsx')
→ df_music.head()
→ X = df_music.drop('genre', axis=1)
  Y = df_music['genre']

→ X.head()
→ Y.head()

→ from sklearn.tree import DecisionTreeClassifier
  model = DecisionTreeClassifier(criterion='entropy')
→ model.fit(X,Y)
→ Prediction = model.Predict([[23,1],[31,0]])
→ Prediction
→ from sklearn.model_selection import train_test_split
  X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2)

→ model.fit(X_train, Y_train)
→ Prediction = model.Predict(X_test)
→ Prediction Prediction
→ Y_test
→ from sklearn.metrics import accuracy_score
  accuracy_score(Y_test, Prediction)
```

	Age	Gender	Genre
0	20	1	HIP HOP
1	24	1	HIP HOP
2	26	1	HIP HOP
3	27	1	ROCK
4	29	1	ROCK

	Age	Gender
0	20	1
1	24	1
2	26	1
3	27	1
4	29	1

0	HIP HOP
1	HIP HOP
2	HIP HOP
3	ROCK
4	ROCK


Age <= 26.5
entropy = 2.76
samples = 22
value = [5, 3, 3, 3, 3, 2, 3]
class = CLASSICAL
True
False
Age <= 19.5
entropy = 1.981
samples = 11
value = [0, 3, 3, 0, 3, 2, 0]
class = DANCE
Age <= 30.5
entropy = 1.539
sample = 11
Value = [5, 0, 0, 3, 0, 0, 3]
Class = CLASSICAL
Gender <= 0.5
entropy = 0.971
samples = 5
value = [0, 0, 0, 0, 3, 2, 0]
class = POP
Gender <= 0.5
entropy = 1.0
samples = 6
value = [0, 3, 3, 0, 0, 0, 0]
class = DANCE
Gender <= 0.5
entropy = 1.0
samples = 6
value = [0, 0, 0, 3, 0, 0, 3]
class = JAZZ
entropy = 0.0
samples = 5
value = [5, 0, 0, 0, 0]
Class = CLASSICAL
entropy = 0.0
samples = 2
value = [0, 0, 0, 0, 0, 2, 0]
class = POPULAR
entropy = 0.0
samples = 3
value = [0, 0, 0, 0, 3, 0, 0]
class = POP
entropy = 0.0
samples = 3
value = [0, 0, 3, 0, 0, 0, 0]
class = HIPHOP
entropy = 0.0
sample = 3
value = [0, 3, 0, 0, 0, 0, 0]
Class = DANCE
entropy = 0.0
samples = 3
value = [0, 0, 0, 3, 0, 0, 0]
class = JAZZ
entropy = 0.0
samples = 3
value = [0, 0, 0, 0, 0, 0, 3]
class = ROCK

POORNIMA

```
→ import joblib
→ joblib.dump(model, 'music-recommender')
→ model = joblib.load('music-recommender')
→ Predictions = model.predict([[23,1],[31,0]])

→ Predictions

→ from sklearn.tree import export_graphviz

→ export_graphviz(model, out_file='music-recommender.dot',
                  feature_names=['Age', 'Gender'])

→ import pydotplus
→ decision_tree = pydotplus.graph_from_dot_file('music-recommender-
                                                 dot')
→ From IPython.display import Image
  Image(decision_tree.Create_Png())
```

POORNIMA

* Experiment-6 *

* Aim :-

ANN with backpropagation.

* Code :-

```
import numpy as np
class NueralNetwork:
    def init_(self, layers):
        self.layers = layers
        self.num_layers = len(layers)
        self.weights = [np.random.randn(y,x) for x,y in
        zip(layers[:-1], layers[1:])]
        self.biases=[np.random.randn (y,1) for y in layers
          [1:])
    def forward-propagation (self, x):
        a=x
        activations=[a]
        zs=[]
for w,b in zip (self.weights, self.biases):
        z = np.dot(w,a)+b
        zs.append (z)
        a = self.Sigmoid (z)
        activations.append(a)
    return activations, zs

def backward_propagations (self, x, y):
```

POORNIMA

```
activations, zs = self.forward_propagation(x)
delta = self.cost_derivative(activations[-1], y) * self.
    sigmoid_prime(zs[-1])
    nabla_b = [delta]
    nabla_w = [np.dot(delta, activation[-2].T)]
    for l in range(2, self.num_layers):
        z = zs[-l]
    sp = self.sigmoid_prime(z)
delta = np.dot(self.weights[-l+1].T, delta) * sp
    nabla_b.insert(0, delta)
    nabla_w.insert(0, np.dot(delta, activations[-l-1].T))
    return nabla_b, nabla_w
def weight_biases(self, mini_batch, eta):
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    for x, y in mini_batch:
delta_nabla_b, delta_nabla_w = self.backward_propagation
(x, y)
    nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_
    nabla_b)]
    nabla_w = [nw+dnw for nw, dnw in zip(nabla_w,
        delta_nabla_w)]
    self.weights = [w-(eta/len(mini_batch))*nw for w, i
        nw in zip(self.weights, nabla_w)
    self.biases = [b-(eta/len(mini_batch))*nb for b, in
        zip(self.biases, nabla_b)
```

```
def train(self, training_data, epochs, mini_batch_size,
    learning_rate):
        n = len(training_data)
        for epoch in range(epochs):
            np.random.shuffle(training_data)
        mini_batches = [training_data[k:k+mini_batch
                _size]
                    for k in range(0, n, mini_batch_size)
                ]
for mini_batch in mini_batches:
    self.update_weights_biases(mini_batch, learning
        rate)
def predict(self, x):
    a = x
    for w, b in zip(self.weights, self.biases):
        z = np.dot(w, a) + b
        a = self.sigmoid(z)
    return a
@ staticmethod
    def sigmoid(z):
        return 1.0/(1.0 + np.exp(-z))
@ staticmethod
        def sigmoid_prime(z):
return NeuralNetwork.sigmoid(z)*(1-NeuralNetwork
    .sigmoid(z))
@staticmethod
```

O/P :-

Prediction for [0,0] : [[0.52397428]]
prediction for [1,1] : [[0.4774217]]

POORNIMA

```
def cost_derivative (output_activations, y):
    return output_activations - y
nn = NeuralNetwork ([2, 3, 1])
    training_data = [
        (np.array [[0], [0]], np.array ([[0]])
        (np.array [[0], [1]], np.array ([[1]])
        (np.array [[1], [0]], np.array ([[1]])
        (np.array [[1], [1]], np.array ([[0]])
    ]
nn.train (training_data, epochs = 1000, mini_batch_size
        = 4, learning_rate = 0.1)
input_data = np.array ([[0], [0]])
prediction = nn.predict (input_data)
print ("prediction for [0,0] :", prediction)

input_data = np.array ([[1], [1]])
prediction = nn.predict (input_data)
print ("prediction for [1,1] : " prediction)
```

POORNIMA

* Experiment - (7) *

*** Aim :-**

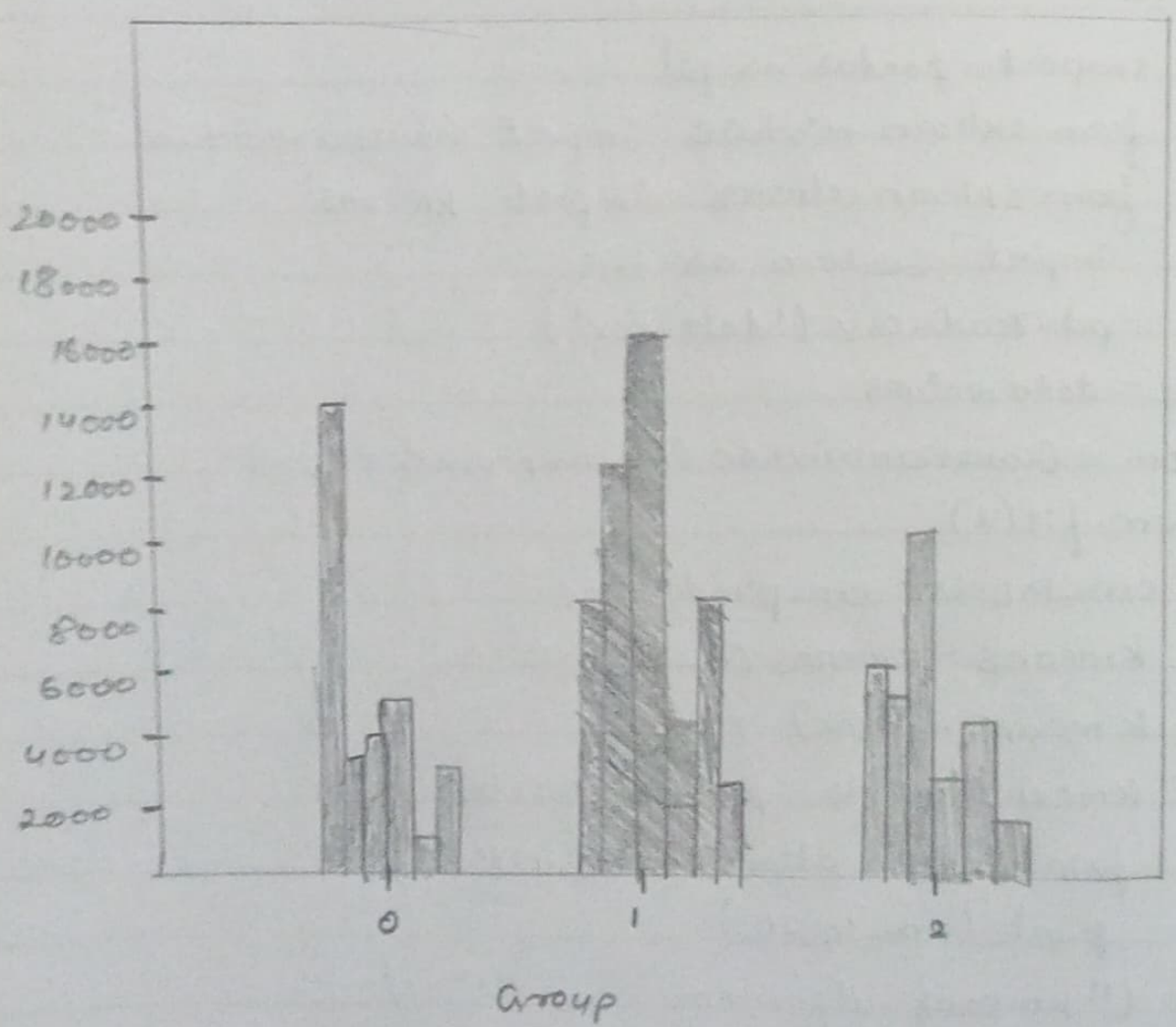
Use the .csv as data file and perform clustering using k-means algorithm. Compare the results of these two algorithms and comment on the quality of clustering.

*** Source-code :-**

```
import pandas as pd
from sklearn.mixture import GaussianMixture
from sklearn.cluster import kmeans
 import seaborn as sns
data = pd.read_csv ('data.csv')
  x = data.values
  em = GaussianMixture (n_components = 3)
  em.fit(x)
   em_labels = em.predict(x)
   kmeans = kmeans (n_clusters = 3)
   k means.fit(x)
   kmeans_labels = kmeans.labels_
   print ("Em algorithm results :")
    print (em_labels)
print ("kmeans algorithm results :")
  print (kmeans_labels)

data = pd.concat [[ data, pd.DataFrame (em_labels, columns
        = ['Group'] )] , axis = 1 , join = 'inner')
```

O/P :-

	channel	Region	Fresh	milk	Grocery	Frozen	Detelgents	Grop
0	2	3	12669	9656	7561	214	2674	2
1	2	3	7057	9810	1762	3293	1776	2
2	2	3	6353	7684	2405	3516	7844	2
3	1	3	13265	4221	6404	507	1788	1
4	2	3	22615	7198	3915	1777	5185	2



```
data.head()
data.groupby("Group").aggregate("mean").plot.bar()
```

POORNIMA

* Experiment-no-⑧ *

* **Aim :-**

write a program to construct a Bayesian network considering modeling data.

* **Source Code :-**

```
import pandas as pd
data = pd.read_csv("heartdisease.csv")
heart_disease = pd.DataFrame(data)
print(heart_disease)
from pgmpy.models import BayesianModel
model = BayesianModel([
        ('age', 'lifestyle'), ('Gender', 'Lifestyle'),
        ('Family', 'Heartdisease'), ('diet', 'cholestrol'),
        ('Lifestyle', 'diet'), ('cholestrol', heart_disease)
    ('diet', 'cholestrol')
        ])
from pgmpy.estimators import maximumLikelihoodEstimator
model.fit(heart_disease, estimator = maximumlikelihood
    Estimator)
from pgmpy.inference import VariableElimination
Heart_disease_infer = VariableElimination(model)
```

O/P :-

For Heart Disease

Enter age: (

Enter Gender: 1

Enter Family History: 0

Enter Diet: 1

Heart Disease	phi (heartdisease)
heartdisease_0	0.000
heartdisease_1	1.000

POORNIMA

```
print ("For age Enter{ SuperSeniorcitizen:0 , seniorcitizen:1,
        middleaged:2, Youth:3, teen:4}")
print ("For bender Enter {male:0 , female:1}")
print (" For family history Enter { Yes:0 , No:1 }")
print (" For Diet Enter { High:0, medium:1 }")

q= HeartDisease_infer.query (variables =['heartdisease'],
        evidence={
    'age': int (input ("Enter age:")),
    'benders': int (input ("Enter benders:")),
    'Family': int (input (" Enter Family history :")),
    'diet': int (input ("Enter diet:")).
        })

print (q['heartdisease'])
```

POORNIMA

* Experiment- no-(9) *

* Aim :-

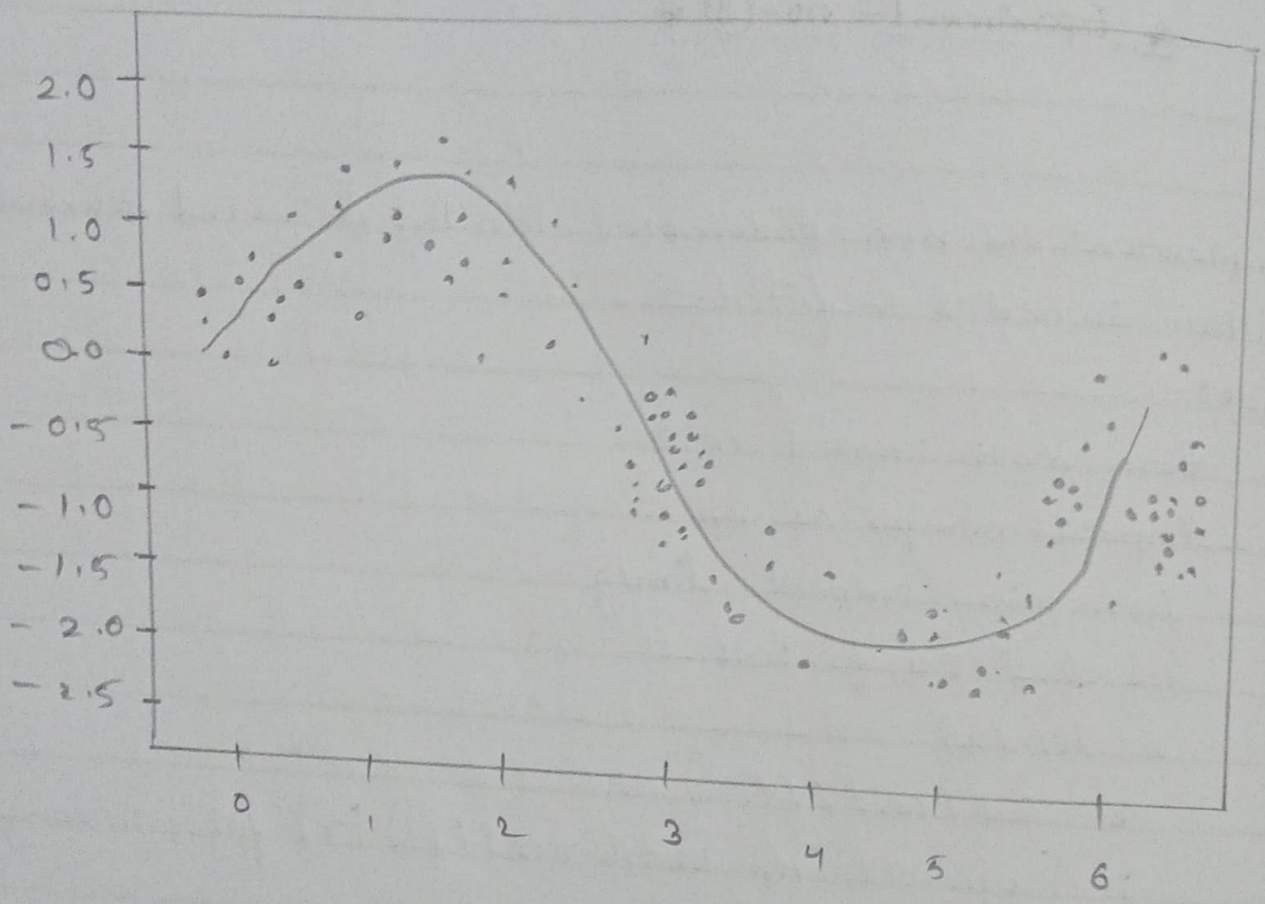
Implement the non-parametric locally weighted Regression algorithm in order to fit data.

* Source Code :-

```
from math import ceil
import numpy as np
from scipy import linalg
def l(x, y, f, i):
    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 is in range(i):
        for io 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(weigh*
        * x)], [np.sum(weights * x), np.sum
        (weights * beta = linalg.solve(A, b)
```

Page No.....

O/P :-



```
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
iz = 3
yest = l(x, y, f, is)
```

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

POORNIMA

Experiment - (10)

Aim :-

Assuming a set of documents that need to be classified, use the naive Bayesian classifier model to perform this task.

Source Code :-

```
import pandas as pd
msg = pd.read_csv('document.csv', names=['msg', 'label'])
print("Total Instances of dataset : ", msg.shape[0])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})
x = msg.message
y = msg.labelnum
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y)
from sklearn.feature_extraction.text import CountVectorizer
Count_v = CountVectorizer()
xtrain_dm = Count_v.fit_transform(xtrain)
xtest_dm = Count_v.transform(xtest)

df = pd.DataFrame(xtrain_dm.toarray(), columns = count_v.get_feature_names())
print(df[0:5])
from sklearn.naive_bayes import multinomialNB
clf = multinomialNB()
clf.fit(xtrain_dm, ytrain)
```

O/P :-

Accuracy metrics:

Accuracy : 0.8

Recall : 1.0

Precision : 0.6666666

Confusion matrix:

[[2 1]
[0 2]]

```
pred = clf.predict(xtest_dm)
for doc, p in zip(xtrain, pred):
    p = 'pos' if p == 1 else 'neg'
    print("%s -> %s" % (doc, p))
```

```
from sklearn.metrics import accuracy_score, confusion
_matrix, precision_score, recall_score
print('Accuracy metrics: \n')
print('Accuracy:', accuracy_score(ytest, pred))
print('Recall:', recall_score(ytest, pred))
print('precision:', precision_score(ytest, pred))
print('Confusion matrix: \n', confusion_matrix(ytest,
    pred))
```