B. M.S. COLLEGE OF ENGINEERING

(Autonomous College under VTU)

Bull Temple Road, Basavangudi, Bangalore - 560019

MACHINE LEARNING LAB

(20CS6PCMAL)

Report

Submitted by

Nithin B S
(1BM20CS100)
VI B

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

Lab Incharge
Dr. ASHA G R
Assistant Professor
BMS College Of Engineering

2023-2024

LIST OF PROGRAMS

Date: 01.04.2023

Program 1 – a) Dataset Exploration – Iris data set

- Features in the Iris dataset:
    1. Sepal length in cm
    2. Sepal width in cm
    3. Petal length in cm
    4. Petal width in cm
- Target classes to predict:
    1. Iris Setosa
    2. Iris Versicolour
    3. Iris Virginica

In [1]:
```
import pandas as pd

#To import the dataset that is present in Iris.csv file
data = pd.read_csv("Iris.csv")
```

In [29]:
```
data
```

Out[29]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [2]:
```
#printing the type of dataset
type(data)
```

Out[2]:
```
pandas.core.frame.DataFrame
```

In [3]:
```
#Inorder to know information about the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [4]:
```
#Printing only particular column in the dataframe
print(data['Species']) #target column
```

```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
            ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

In [30]:
```
print(data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]) #Feature columns
```

```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0              5.1           3.5            1.4           0.2
1              4.9           3.0            1.4           0.2
2              4.7           3.2            1.3           0.2
3              4.6           3.1            1.5           0.2
4              5.0           3.6            1.4           0.2
..             ...           ...            ...           ...
145            6.7           3.0            5.2           2.3
146            6.3           2.5            5.0           1.9
147            6.5           3.0            5.2           2.0
148            6.2           3.4            5.4           2.3
149            5.9           3.0            5.1           1.8

[150 rows x 4 columns]
```

In [5]:
```
#To print first n rows
data.head()
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
In [27]: data.shape
         #Species number of rows and columns that are present
```

```
Out[27]: (150, 6)
```

```
In [28]: data.describe()
```

Out[28]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

b. Practice Exercise - Dataset Exploration – wine data set

- **Features**
  1. Alcohol
  2. Malic Acid
  3. Ash
  4. Alcalinity of ash
  5. Magnesium
  6. Total phenols
  7. Falvanoids
  8. Nonflavanoid phenols
  9. Proanthocyanins
  10. Color intensity
  11. Hue
  12. od280/od315 of diluted wines
  13. Proline
- **Target**
  1. Class_0
  2. Class_1
  3. Class_2

In [34]:
```
from sklearn.datasets import load_wine
#loading wine dataset from sklearn library
wine = load_wine()
```

In [40]:
```
print(wine.data)
```

```
[[1.423e+01 1.710e+00 2.430e+00 ... 1.040e+00 3.920e+00 1.065e+03]
 [1.320e+01 1.780e+00 2.140e+00 ... 1.050e+00 3.400e+00 1.050e+03]
 [1.316e+01 2.360e+00 2.670e+00 ... 1.030e+00 3.170e+00 1.185e+03]
 ...
 [1.327e+01 4.280e+00 2.260e+00 ... 5.900e-01 1.560e+00 8.350e+02]
 [1.317e+01 2.590e+00 2.370e+00 ... 6.000e-01 1.620e+00 8.400e+02]
 [1.413e+01 4.100e+00 2.740e+00 ... 6.100e-01 1.600e+00 5.600e+02]]
```

In [41]:
```
print(wine.data[0])
#can print specific row in the dataset
```

```
[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
 2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
```

In [42]:
```
wine.data.size
```

Out[42]:
```
2314
```

In [43]:
```
type(wine)
```

Out[43]:
```
sklearn.utils._bunch.Bunch
```

In [52]:
```
print(wine.data.shape)
print(wine.target.shape)
```

```
(178, 13)
(178,)
```

In [52]:
```
print(wine.data.shape)
print(wine.target.shape)
```

```
(178, 13)
(178,)
```

In [51]:
```
print(wine.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

In [54]:
```
n_samples, n_features = wine.data.shape
print("Number of samples: ",n_samples)
print("Number of features: ",n_features)
```

```
Number of samples:  178
Number of features:  13
```

In [56]:
```
print(wine.target_names)
```

```
['class_0' 'class_1' 'class_2']
```

Date: 08.04.2023

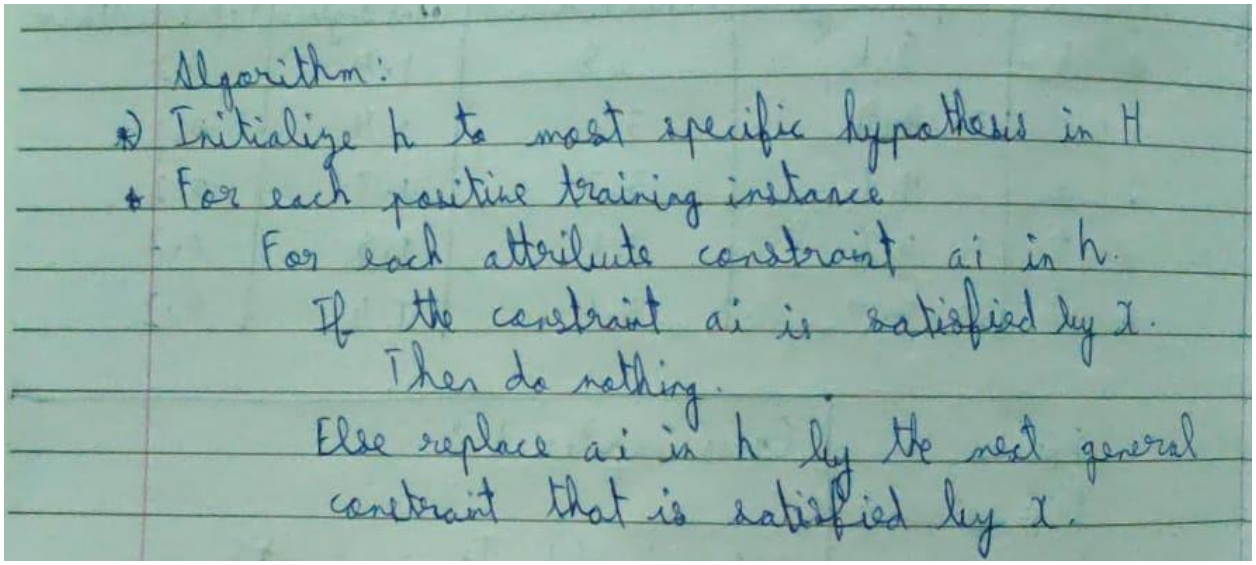
Program 2 – Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file

Data set:

a. Enjoysport

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

Algorithm:



2) Finds algorithm ENJOYSPORT.csv

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	1
1	Sunny	Warm	High	Strong	Warm	Same	1
2	Rainy	Cold	High	Strong	Warm	Change	0
3	Sunny	Warm	High	Strong	Cool	Change	1

Algorithm:

* Initialize h to most specific hypothesis in H
* For each positive training instance
  For each attribute constraint ai in h.
    If the constraint ai is satisfied by x.
      Then do nothing.
    Else replace ai in h by the next general constraint that is satisfied by x.

In[1]:
```
import pandas as pd
import numpy as np
data = pd.read_csv("D:\\ENJOYSPORT.csv")
data
```

Out[1]:

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	1
1	Sunny	Warm	High	Strong	Warm	Same	1
2	Rainy	Cold	High	Strong	Warm	Change	0
3	Sunny	Warm	High	Strong	Cool	Change	1

In[2]:
```
d = np.array(data)[:,:-1]
```

Attributes are [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

In[3]
```
target = np.array(data)[:, -1]
print("\n Target is:", target)
```

Target is: [1 1 0 1]

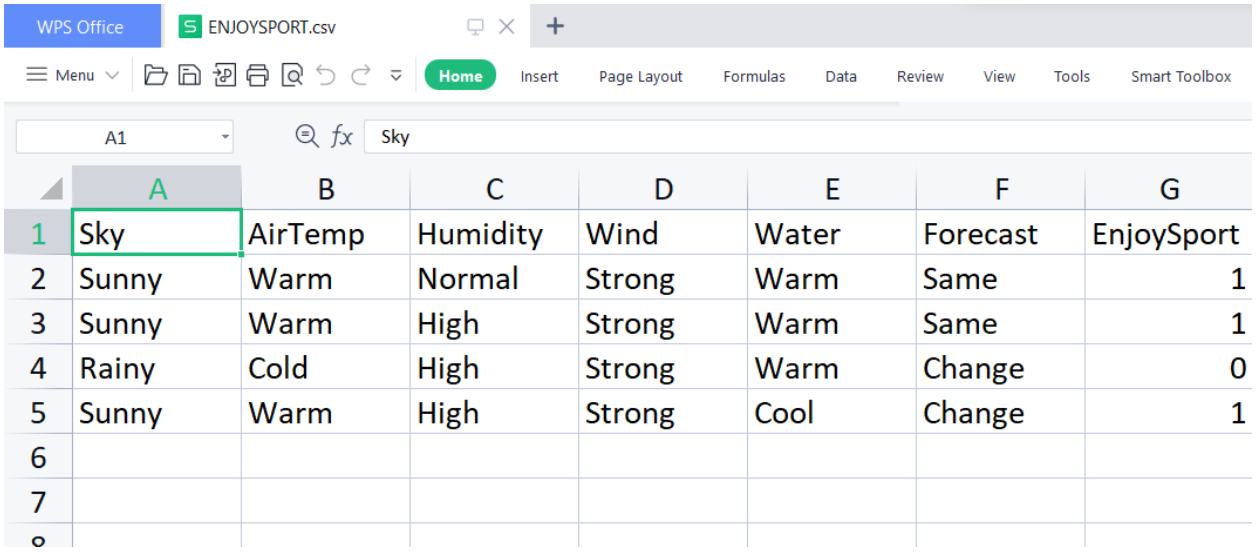
In[4]
```
def findS(c, t):
    for i, val in enumerate(t):
        if val == 1:
            specific_hypothesis = c[i].copy()
            break
    for i, val in enumerate(c):
        if t[i] == 1:
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass
    return specific_hypothesis

print("\n Final hypothesis is:", findS(d, target))
```

Final hypothesis is: ['Sunny' 'Warm' '?' 'Strong' '?' '?']

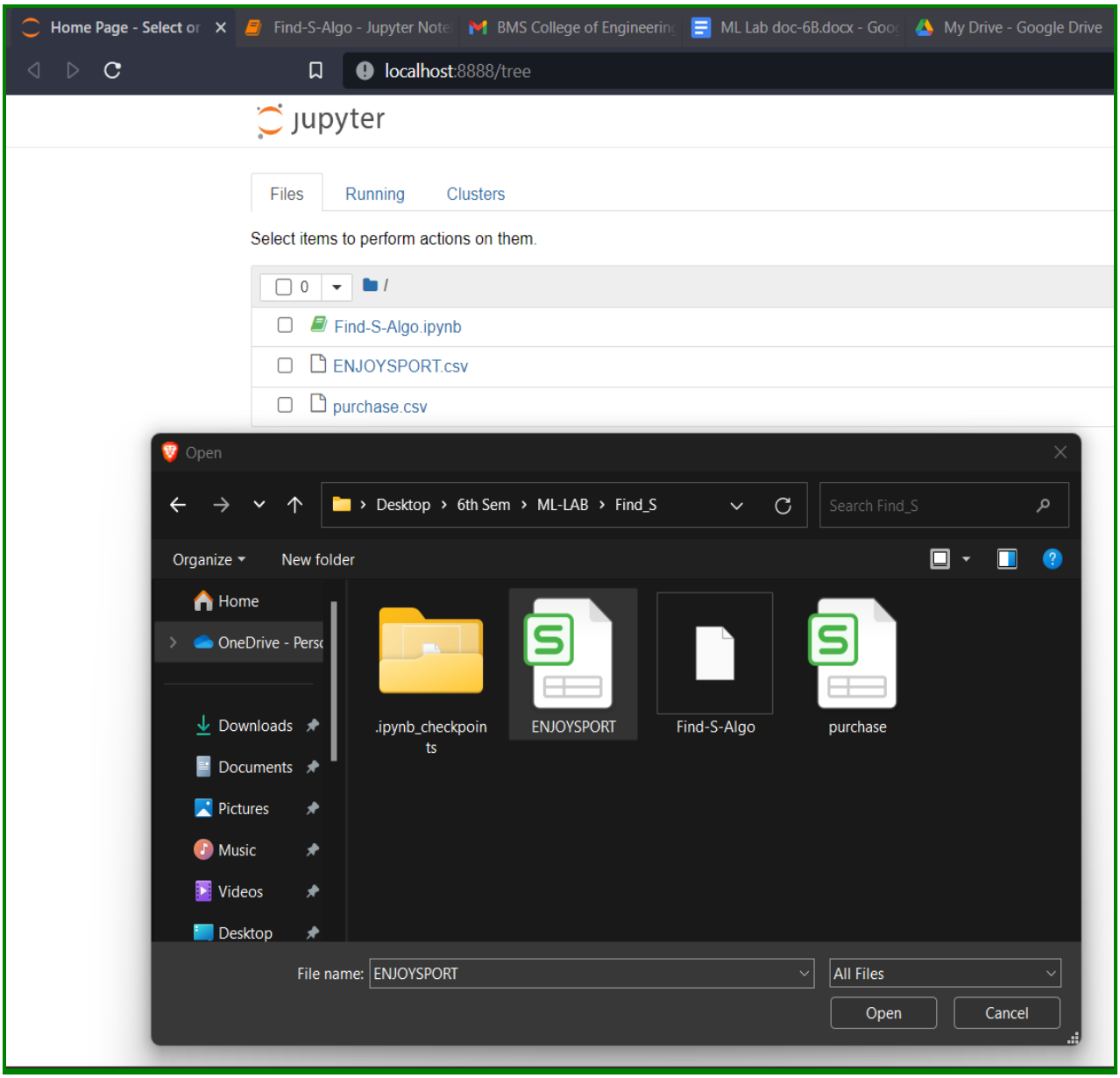
1. The first step of FIND-S is to initialize h to the most specific hypothesis in H **h — (Ø, Ø, Ø, Ø, Ø, Ø)**
2. First training example x1 = < Sunny, Warm, Normal, Strong ,Warm ,Same>, EnjoySport = +ve. Observing the first training example, it is clear that hypothesis h is too specific. None of the "Ø" constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example **h1 = < Sunny, Warm, Normal, Strong ,Warm, Same>**.

3. Consider the second training example x2 = < Sunny, Warm, High, Strong, Warm, Same>, EnjoySport = +ve. The second training example forces the algorithm to further generalize h, this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example. Now **h2 =< Sunny, Warm, ?, Strong, Warm, Same>**
4. Consider the third training example x3 =< Rainy, Cold, High, Strong, Warm, Change>,EnjoySport = — ve. The FIND-S algorithm simply ignores every negative example. So the hypothesis remain as before, so **h3 =< Sunny, Warm, ?, Strong, Warm, Same>**
5. Consider the fourth training example x4 =<Sunny,Warm,High,Strong, Cool,Change>, EnjoySport =+ve. The fourth example leads to a further generalization of h as **h4 =< Sunny, Warm, ?, Strong, ?, ?>**
6. So the final hypothesis is **< Sunny, Warm, ?, Strong, ?, ?>**

Uploading the csv file.



	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	1
3	Sunny	Warm	High	Strong	Warm	Same	1
4	Rainy	Cold	High	Strong	Warm	Change	0
5	Sunny	Warm	High	Strong	Cool	Change	1
6							
7							


Home Page - Select or
Find-S-Algo - Jupyter Note
BMS College of Engineerin
ML Lab doc-6B.docx - Goo
My Drive - Google Drive
localhost:8888/tree
jupyter
Files
Running
Clusters
Select items to perform actions on them.
0
/
Find-S-Algo.ipynb
ENJOYSPORT.csv
purchase.csv
Open
Desktop › 6th Sem › ML-LAB › Find_S
Search Find_S
Organize
New folder
Home
OneDrive - Perso
Downloads
Documents
Pictures
Music
Videos
Desktop
.ipynb_checkpoints
ENJOYSPORT
Find-S-Algo
purchase
File name: ENJOYSPORT
All Files
Open
Cancel

Program:

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
enjoy = pd.read_csv("ENJOYSPORT.csv")
```

In [3]:
```
enjoy.head()
```

Out[3]:

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	1
1	Sunny	Warm	High	Strong	Warm	Same	1
2	Rainy	Cold	High	Strong	Warm	Change	0
3	Sunny	Warm	High	Strong	Cool	Change	1

In [48]:
```
X = np.array(enjoy)[:,:-1]
Y = np.array(enjoy)[:,-1]
print(X)
print(Y)
```

```
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
[1 1 0 1]
```

In [49]:
```
def train(X,Y):
    final = []
    for array,target in zip(X,Y):
        if target==0:
            continue
        else:
```

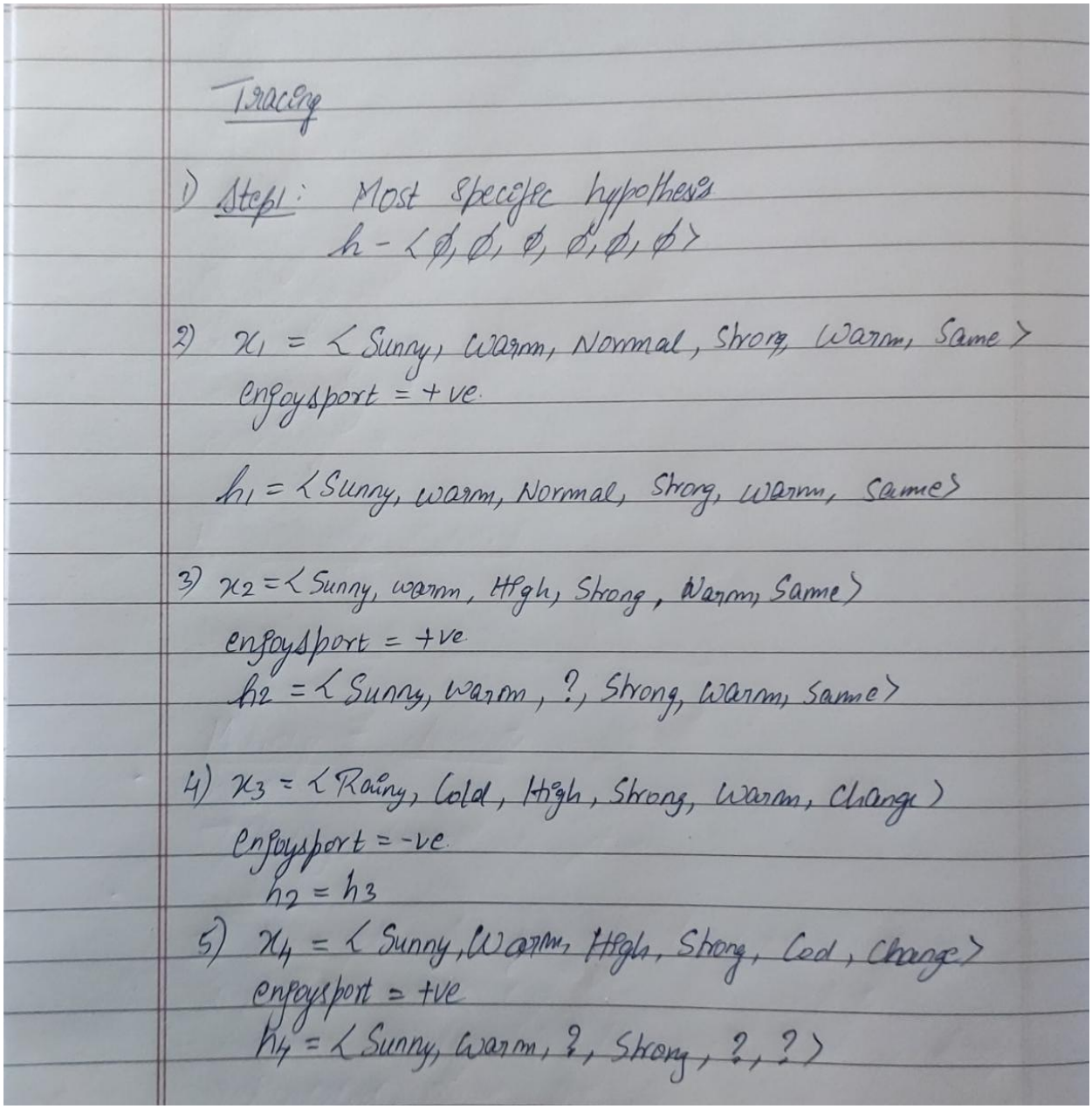
```
        else:
            if len(final)==0:
                final = array
            else:
                for i in range(len(final)):
                    if final[i] != array[i]:
                        final[i] = '?'
                        print(final)
```

Output:

```
In [50]: train(X,Y)
```

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

Tracing:



b. Purchase.csv

example	*citations*	*size*	*inLibrary*	*price*	*editions*	*buy*
1	some	small	no	affordable	many	no
2	many	big	no	expensive	one	yes
3	some	big	always	expensive	few	no
4	many	medium	no	expensive	many	yes
5	many	small	no	affordable	many	yes

Program:

In [26]:
```
purchase = pd.read_csv("purchase.csv")
purchase.head()
```

Out[26]:

	example	citations	size	inLibrary	price	editions	buy
0	1	some	small	no	affordable	many	no
1	2	many	big	no	expensive	one	yes
2	3	some	big	always	expensive	few	no
3	4	many	medium	no	expensive	many	yes
4	5	many	small	no	affordable	many	yes

In [51]:
```
features = np.array(purchase)[:,1:-1]
target = np.array(purchase)[:,-1]
print(features)
```

```
[['some' 'small' 'no' 'affordable' 'many']
 ['many' 'big' 'no' 'expensive' 'one']
 ['some' 'big' 'always' 'expensive' 'few']
 ['many' 'medium' 'no' 'expensive' 'many']
 ['many' 'small' 'no' 'affordable' 'many']]
```

In [52]:
```
for i in range(len(target)):
    if target[i]=='no':
        target[i] = 0
    else:
        target[i] = 1
print(target)
```

```
[0 1 0 1 1]
```

Output:

```
In [53]: train(features,target)
```

```
['many' '?' 'no' 'expensive' 'one']
['many' '?' 'no' 'expensive' '?']
['many' '?' 'no' 'expensive' '?']
['many' '?' 'no' '?' '?']
['many' '?' 'no' '?' '?']
```

Date:15/4/23

Program 3- 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 training examples.

Data set:

a. Enjoysport

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

Observation:

Page No. 5
Date

29/4/23

3. Candidate elimination algorithm

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	1
1	Sunny	Warm	High	Strong	Warm	Same	1
2	Rainy	Cold	High	Strong	Warm	Change	0
3	Sunny	Warm	High	Strong	Cool	Change	1

Algorithm:

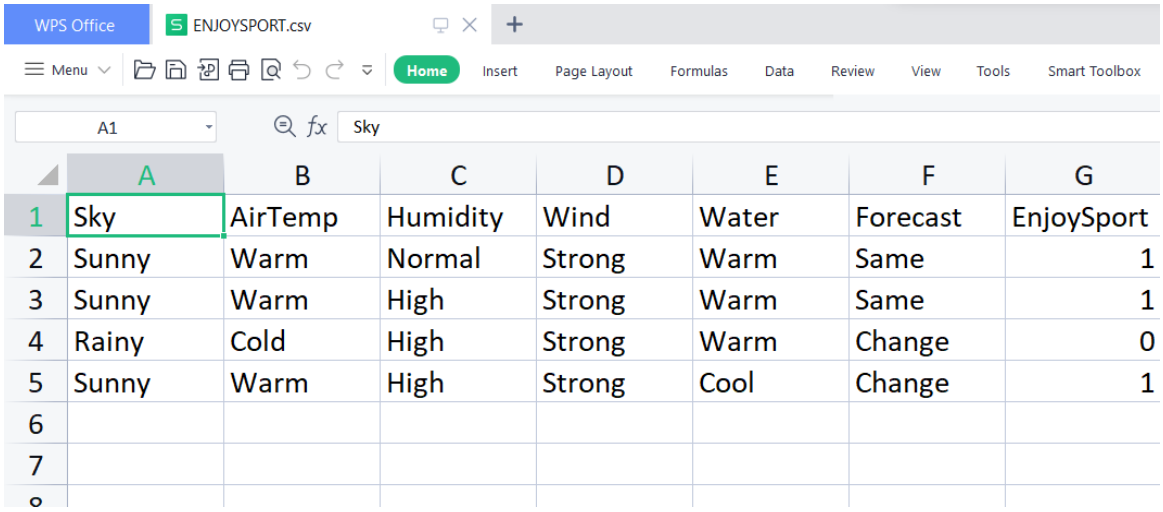
* Initialize G to the set of maximally general hypothesis in H.
* Initialize S to the set of maximally specific hypothesis in H.
* For each training example d, do
  If d is a +ve example:
  (i) Remove from G any hypothesis inconsistent with d.
  (ii) For each hypothesis in S that is not consistent with d
    (a) Remove s from S
    (b) Add to S all minimal generalization h of s such that h is consistent with d, & some member of G is more general than h.
    (c) Remove from S any hypothesis that is more general than another hypothesis in S.
  else:
  If d is a -ve example:
  Follow the above steps that S ↔ G

Code:

```
import pandas as pd
import numpy as np
data = pd.DataFrame(data = pd.read_csv('/content/
ENJOYSPORT.csv'))
concepts = np.array(data.iloc[:, 0:-1])
print(concepts)
target = np.array(data.iloc[:, -1])
print(target)
def learn(concepts, target):
    spec_h = concepts[0].copy()
    print("initialization of spec_h & gen_h")
    print(spec_h)
    gen_h = [["?" for i in range(len(spec_h))] for
    i in range(len(spec_h))]
    print(gen_h)
    for i, h in enumerate(concepts):
        print("For loop starts")
        if target[i]=="yes":
            print("If positive")
            for x in range(len(spec_h)):
```

Output:

Uploading CSV:

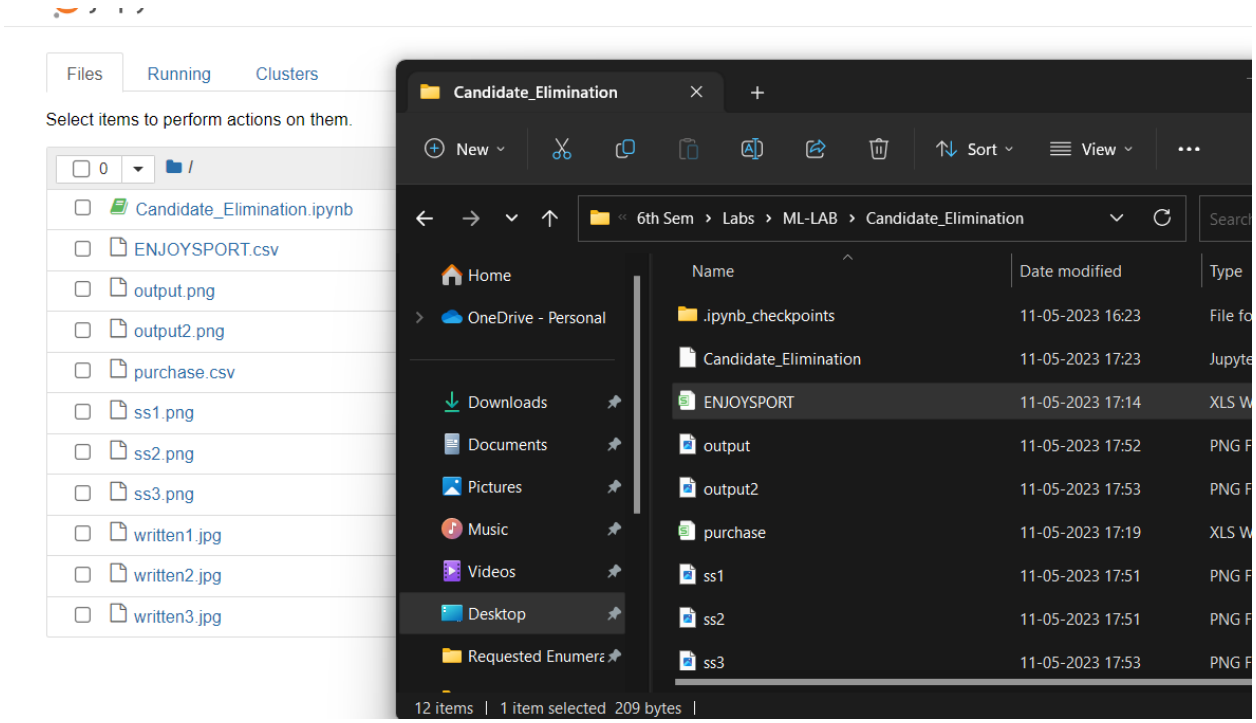


WPS Office | ENJOYSPORT.csv

Menu | Home | Insert | Page Layout | Formulas | Data | Review | View | Tools | Smart Toolbox

A1 | fx Sky

	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	1
3	Sunny	Warm	High	Strong	Warm	Same	1
4	Rainy	Cold	High	Strong	Warm	Change	0
5	Sunny	Warm	High	Strong	Cool	Change	1
6							
7							



Program:

In [97]:
```
import numpy as np
import pandas as pd
```

In [98]:
```
data = pd.read_csv('./ENJOYSPORT.csv')
```

In [99]:
```
data
```

Out[99]:

	sky	airtemp	humidity	wind	water	forcast	enjoysport
0	sunny	warm	normal	strong	warm	same	yes
1	sunny	warm	high	strong	warm	same	yes
2	rainy	cold	high	strong	warm	change	no
3	sunny	warm	high	strong	cool	change	yes

In [100]:
```
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
```

```
Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
```

In [101]:
```
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
```

```
Target Values are:  ['yes' 'yes' 'no' 'yes']
```

In [102]:
```
def learn(concepts, target):
```

In [102]:
```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print(specific_h)
        print(general_h)
        print("\n")
    valuer = []
    for i in concepts[0]:
        valuer.append('?')
    indices = [i for i, val in enumerate(general_h) if val == valuer]
    for i in indices:
        general_h.remove(valuer)
    return specific_h, general_h
```

Output:

```
In [103]: s_final, g_final = learn(concepts, target)

          print("Final Specific_h: ", s_final, sep="\n")
          print("Final General_h: ", g_final, sep="\n")
```

```
Initialization of specific_h and genearal_h

Specific Boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


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

b. Purchase.csv

example	*citations*	*size*	*inLibrary*	*price*	*editions*	*buy*
1	some	small	no	affordable	many	no
2	many	big	no	expensive	one	yes
3	some	big	always	expensive	few	no
4	many	medium	no	expensive	many	yes
5	many	small	no	affordable	many	yes

Execution:

In [ ]:
```
##SECOND DATASET PURCHASE.csv
```

In [104]:
```
data = data=pd.read_csv('./purchase.csv')
```

In [105]:
```
data
```

Out[105]:

	citations	size	inLibrary	price	editions	buy
0	many	big	no	expensive	one	yes
1	some	big	always	expensive	few	no
2	many	medium	no	expensive	many	yes
3	many	small	no	affordable	many	yes

In [106]:
```
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
```

```
Instances are:
 [['many' 'big' 'no' 'expensive' 'one']
 ['some' 'big' 'always' 'expensive' 'few']
 ['many' 'medium' 'no' 'expensive' 'many']
 ['many' 'small' 'no' 'affordable' 'many']]
```

In [107]:
```
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
```

```
Target Values are:  ['yes' 'no' 'yes' 'yes']
```

In [108]:
```
s_final, g_final = learn(concepts, target)
```

Output:

```
In [108]: s_final, g_final = learn(concepts, target)

          print("Final Specific_h:", s_final, sep="\n")
          print("Final General_h:", g_final, sep="\n")
```

```
Initialization of specific_h and genearal_h

Specific Boundary:  ['many' 'big' 'no' 'expensive' 'one']

Generic Boundary:  [['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?']]

Instance 1 is  ['many' 'big' 'no' 'expensive' 'one']
Instance is Positive
['many' 'big' 'no' 'expensive' 'one']
[['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?']]



Instance 2 is  ['some' 'big' 'always' 'expensive' 'few']
Instance is Negative
['many' 'big' 'no' 'expensive' 'one']
[['many', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?',
'?', '?', 'one']]



Instance 3 is  ['many' 'medium' 'no' 'expensive' 'many']
Instance is Positive
['many' '?' 'no' 'expensive' '?']
[['many', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?']]



Instance 4 is  ['many' 'small' 'no' 'affordable' 'many']
Instance is Positive
['many' '?' 'no' '?' '?']
[['many', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?']]


Final Specific_h:
['many' '?' 'no' '?' '?']
Final General_h:
[['many', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?']]
```

Date:29-04-2023

Program-4:

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.

Dataset: Playtennis

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

Observation:

29/4/23

Page No. 7
Date

Lab - 4

ID3 algorithm Dataset : play-tennis.csv
ID3.csv , id3_test_1.csv

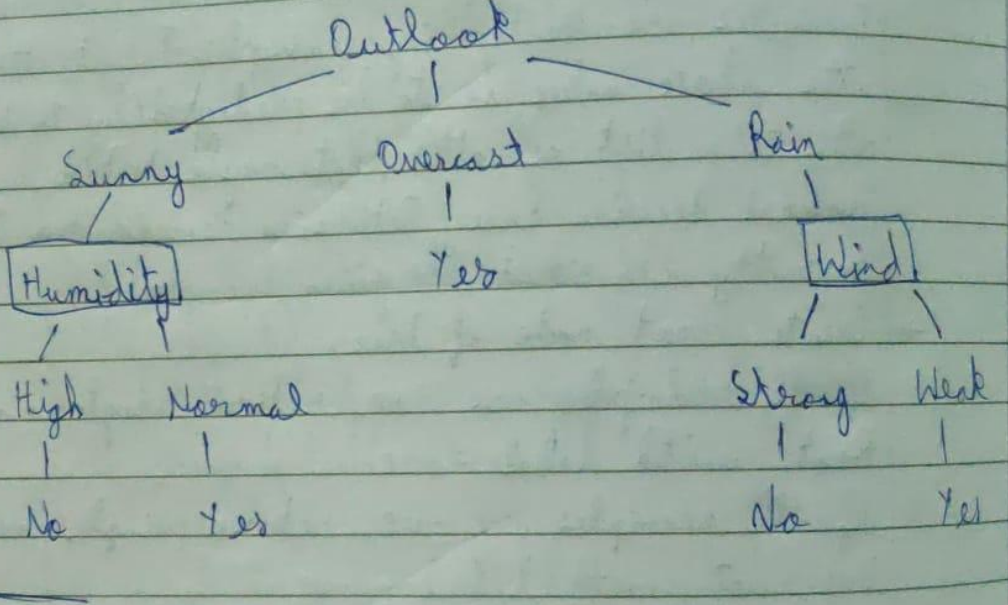
Algo :
ID3 (Examples, Target_Attr, Attr)
Examples are the training examples. Target attr is the attr whose value is to be predicted by the tree. Attr is a list of attr that maybe tested by the learned decision tree. Returns a decision tree that correctly classifies the given examples.

→ Create root node of tree.
→ If all examples are +ve, return single-node tree Root, with label = +ve.
→ If all examples are -ve, return single-node tree tree root, with label = -ve
→ If attr is empty, Return a single node tree root, with label = most common value of target_attr in eg.
→ Otherwise Begin
  A ← the attr from attr that best classifies examples.
  The decision attr for root ← A
  For each possible value vi of A,
  → Add a new tree branch below Root, corresponding to the test A = vi
  → let examples vi, be the subset of examples that has vi for A.
  → If examples vi is empty
    → then below this new branch add a leaf node with label = most common value of target_attr in examples.

→ else below this new branch add subtree
→ ID3 (Examples, Target_attr, attr - {A})

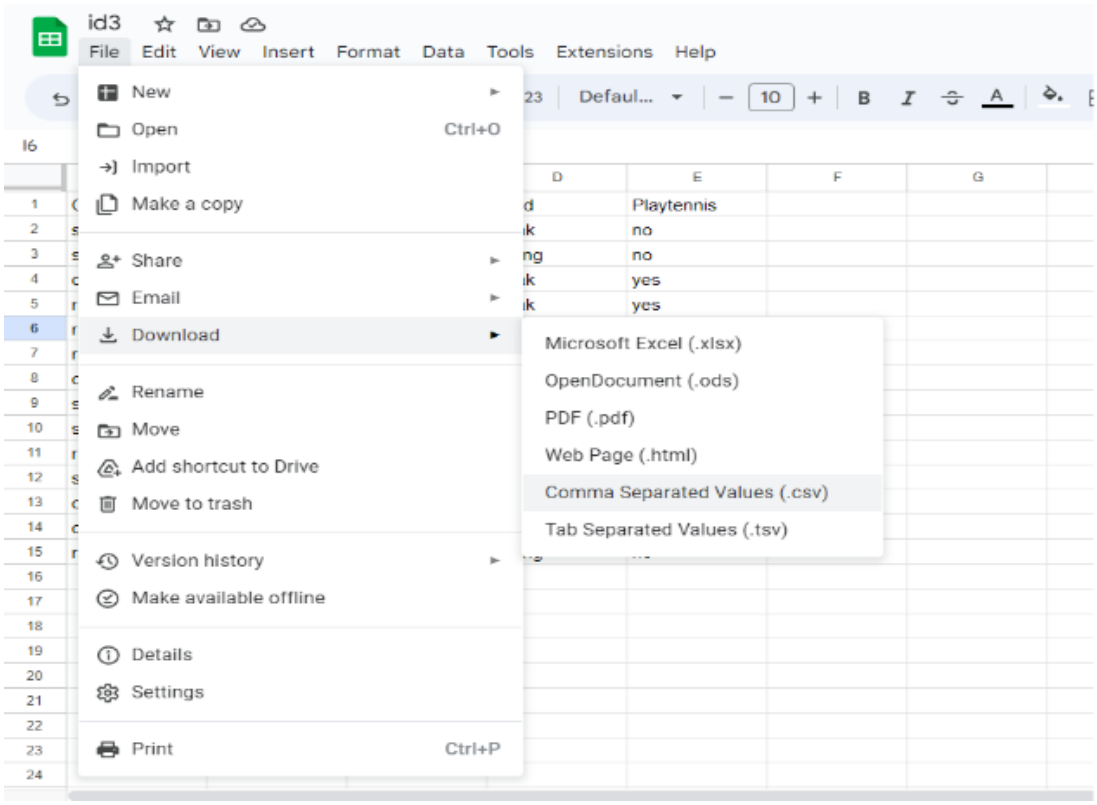
→ End
→ Return Root

o/p :-



ALP
12/5/2023

Uploading CSV :



```
[ ]  class Node:
         def __init__(self):
             self.children = []
             self.value = ""
             self.isLeaf = False
             self.pred = ""
```

```
[ ]  def entropy(examples):
         pos = 0.0
         neg = 0.0
         for _, row in examples.iterrows():
             if row["play"] == "Yes":
                 pos += 1
             else:
                 neg += 1
         if pos == 0.0 or neg == 0.0:
             return 0.0
         else:
             p = pos / (pos + neg)
             n = neg / (pos + neg)
             return -(p * math.log(p, 2) + n * math.log(n, 2))
```

```
[ ]  def info_gain(examples, attr):
         uniq = np.unique(examples[attr])
         #print ("\n",uniq)
         gain = entropy(examples)
         #print ("\n",gain)
         for u in uniq:
             subdata = examples[examples[attr] == u]
             #print ("\n",subdata)
             sub_e = entropy(subdata)
             gain -= (float(len(subdata)) / float(len(examples))) * sub_e
             #print ("\n",gain)
         return gain
```

+ Code + Text

```
def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["play"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root
```

```
[ ]  def printTree(root: Node, depth=0):
         for i in range(depth):
             print("\t", end="")
         print(root.value, end="")
         if root.isLeaf:
             print(" -> ", root.pred)
         print()
         for child in root.children:
             printTree(child, depth + 1)
```

```
[ ]  root = ID3(df, features)
     printTree(root)
```

Output:

```
[ ]  root = ID3(df, features)
     printTree(root)

     Outlook
             Overcast ->  ['Yes']

             Rain
                     Wind
                             Strong ->  ['No']

                             Weak ->  ['Yes']

             Sunny
                     Humidity
                             High ->  ['No']

                             Normal ->  ['Yes']
```

Date:12-05-2023

Program-5: Linear regression

Dataset:

	A	B
1	YearsExperience	Salary
2	1.1	39343
3	1.3	46205
4	1.5	37731
5	2	43525
6	2.2	39891
7	2.9	56642
8	3	60150
9	3.2	54445
10	3.2	64445
11	3.7	57189
12	3.9	63218
13	4	55794
14	4	56957
15	4.1	57081
16	4.5	61111
17	4.9	67938
18	5.1	66029
19	5.3	83088
20	5.9	81363
21	6	93940
22	6.8	91738
23	7.1	98273
24	7.9	101302
25	8.2	113812
26	8.7	109431
27	9	105582
28	9.5	116969
29	9.6	112635
30	10.3	122391
31	10.5	121872

Observation:

12/5/23

Page No. 9
Date

Lab - 5

Linear regression Dataset : salary_data.csv

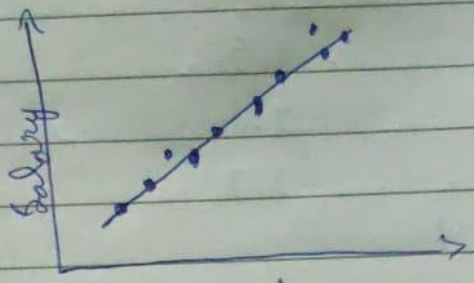
Dataset :

Index	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

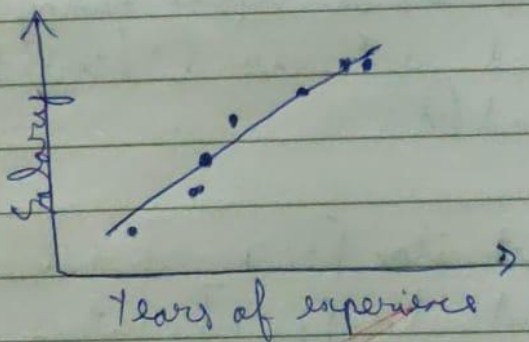
Code :

```
import pandas as pd
import matplotlib.pyplot as plt
import ~~pandas as~~ import numpy as np
dataset = pd.read_csv ('/content/Salary_data.csv')
dataset.head()

x = dataset.iloc[:, 1:-1].values
Y = dataset.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x, Y
                                   test_size = 1/3, random_state = 0)
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit ( X_train, Y_train)
Y_pred = model.predict(X_test)
X_pred = model.predict (X_train)
```

```
plt.scatter (x_train, y_train, color = "Green")
plt.plot (x_train, x_pred, color = "Red")
plt.title ("Salary v/s Experience (Training Dataset)")
plt.xlabel ("Years of experience")
plt.ylabel ("Salary")
plt.show ()
```



Training set



Test set

AL
12/5/2023

Program Execution:

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

```
def estimate_coef(x, y):
  n = np.size(x)
  mean_x = np.mean(x)
  mean_y = np.mean(y)
  co_var_xy = np.sum(y*x) - n*mean_y*mean_x
  var_xx = np.sum(x*x) - n*mean_x*mean_x

  beta_1 = co_var_xy / var_xx
  beta_0 = mean_y - beta_1*mean_x

  return (beta_0, beta_1)
```

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

```
[ ] def main():
  dataset = pd.read_csv("/content/Salary_Data.csv")
  X = np.array(dataset.iloc[:,0])
  y = np.array(dataset.iloc[:,-1])
  b = estimate_coef(X, y)
  print("Estimated coefficients:\nb_0 = {} \nb_1 = {}".format(b[0], b[1]))
  plot_regression_line(X, y, b)
```

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

Outputs:


Salary VS Experience (Training set)
120000
100000
80000
60000
40000
Salary
2
4
6
8
10
Year of Experience


Salary VS Experience (Test set)
120000
100000
80000
60000
40000
Salary
2
4
6
8
10
Year of Experience

Date: 19.05.2023

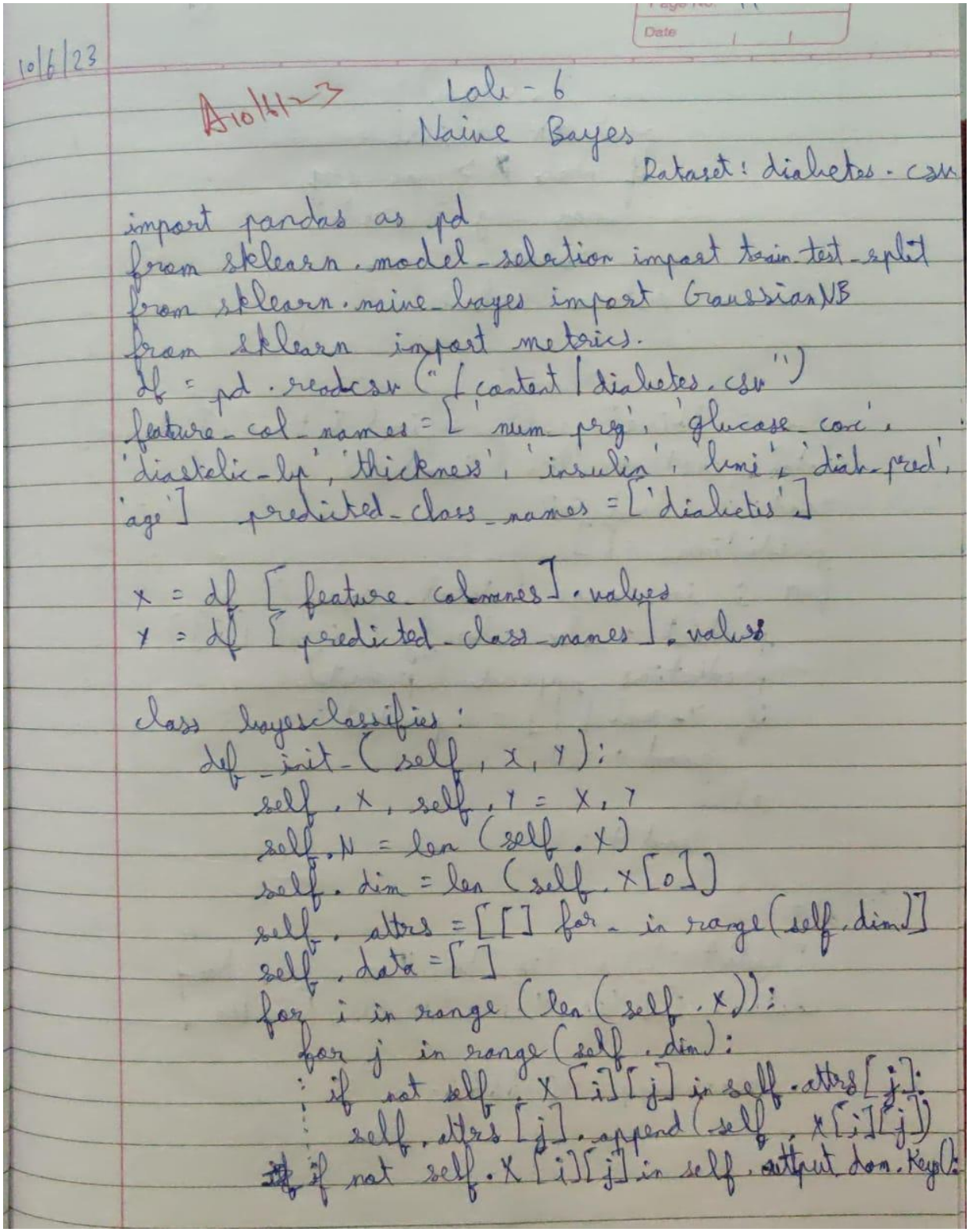
Program 6 – Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

Data set: Climate Dataset

nbc.csv

	A	B
1	outlook	play
2	rainy	Yes
3	sunny	Yes
4	overcast	Yes
5	overcast	Yes
6	sunny	No
7	rainy	Yes
8	sunny	Yes
9	overcast	Yes
10	rainy	No
11	sunny	No
12	sunny	Yes
13	rainy	No
14	overcast	Yes
15	overcast	Yes

Observation:



10/6/23

A10/11 -> Lab - 6

Naive Bayes

Dataset: diabetes.csv

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics.
df = pd.read_csv("/content/diabetes.csv")
feature_col_names = ['num_preg', 'glucose_conc',
'diastolic_bp', 'thickness', 'insulin', 'bmi', 'dia_pred',
'age'] predicted_class_names = ['diabetes']

X = df[feature_columns].values
Y = df[predicted_class_names].values

class bayesclassifier:
    def __init__(self, X, Y):
        self.X, self.Y = X, Y
        self.N = len(self.X)
        self.dim = len(self.X[0])
        self.attrs = [[] for _ in range(self.dim)]
        self.data = []
        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])
            if not self.X[i][j] in self.output_dom.keys():
```

```
n = len(cases)
prob = n/self.N
if prob > max_arg:
    max_arg = prob
    solve = y
return solve
```

```
nbc = NaiveBayesClassifier(X_train, Y_train)
total_cost = len(Y_val)
good = 0, bad = 0
predictions = []
for i in range(total_cost):
    predict = nbc.classify(X_val[i])
    predictions.append(predict)
    if Y_val[i] == predict:
        good += 1
    else
        bad += 1
```

o/p :

Predict = [No, Yes, No, Yes, Yes, No]
Actual = [Yes, Yes, Yes, Yes, Yes, No]

Accuracy = 0.6666

Program Execution:

```
y_train = y[:8]
y_val = y[8:]
X_train = X[:8]
X_val = X[8:]
print(f'Target Values: {y_train}')
print(f'Features: \n{X_train}')
print(f"Number of instances in training set: {len(X_train)}")
print(f"Number of instances in testing set: {len(X_val)}")
```

```
class NaiveBayesClassifier:
    def __init__(self, X, y):
        self.X, self.y = X, y
        self.N = len(self.X)
        self.dim = len(self.X[0])
        self.attrs = [[] for _ in range(self.dim)]
        self.output_dom = {}
        self.data = []
        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])
            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1
            else:
                self.output_dom[self.y[i]] += 1
            self.data.append([self.X[i], self.y[i]])
    def classify(self, entry):
        solve = None
        max_arg = -1
        for y in self.output_dom.keys():
            prob = self.output_dom[y]/self.N
            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                n = len(cases)
                prob *= n/self.N
            if prob > max_arg:
                max_arg = prob
                solve = y
        return solve
```

```
nbc = NaiveBayesClassifier(X_train, y_train)
total_cases = len(y_val)
good = 0
bad = 0
predictions = []
for i in range(total_cases):
    predict = nbc.classify(X_val[i])
    predictions.append(predict)
    if y_val[i] == predict:
        good += 1
    else:
        bad += 1
print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', good)
print('Number of wrong predictions:', bad)
print()
print('Accuracy of Bayes Classifier:', good/total_cases)
```

Output:

```
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```

Date: 19.05.2023

Program 7 – Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

Data set:

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa
5.1,3.3,1.7,0.5,Iris-setosa
4.8,3.4,1.9,0.2,Iris-setosa
5.0,3.0,1.6,0.2,Iris-setosa
5.0,3.4,1.6,0.4,Iris-setosa
5.2,3.5,1.5,0.2,Iris-setosa
5.2,3.4,1.4,0.2,Iris-setosa
4.7,3.2,1.6,0.2,Iris-setosa
4.8,3.1,1.6,0.2,Iris-setosa
5.4,3.4,1.5,0.4,Iris-setosa
5.2,4.1,1.5,0.1,Iris-setosa
```

Observation:

Page No. 13
Date

10/6/23

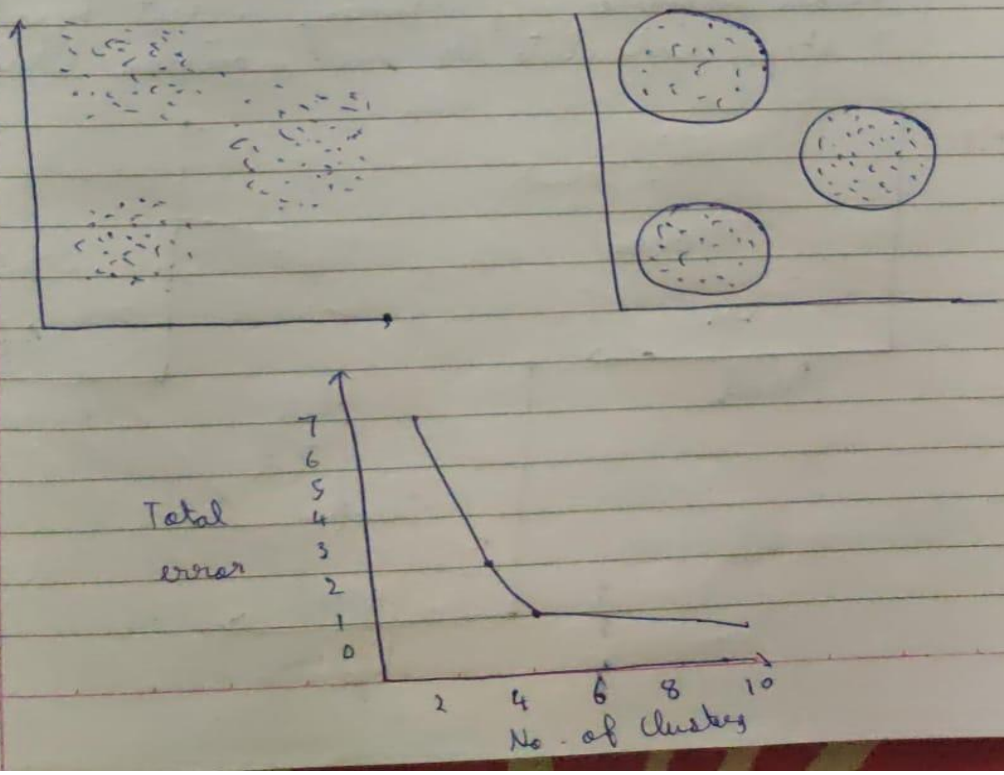
Lab - 7

K means Clustering

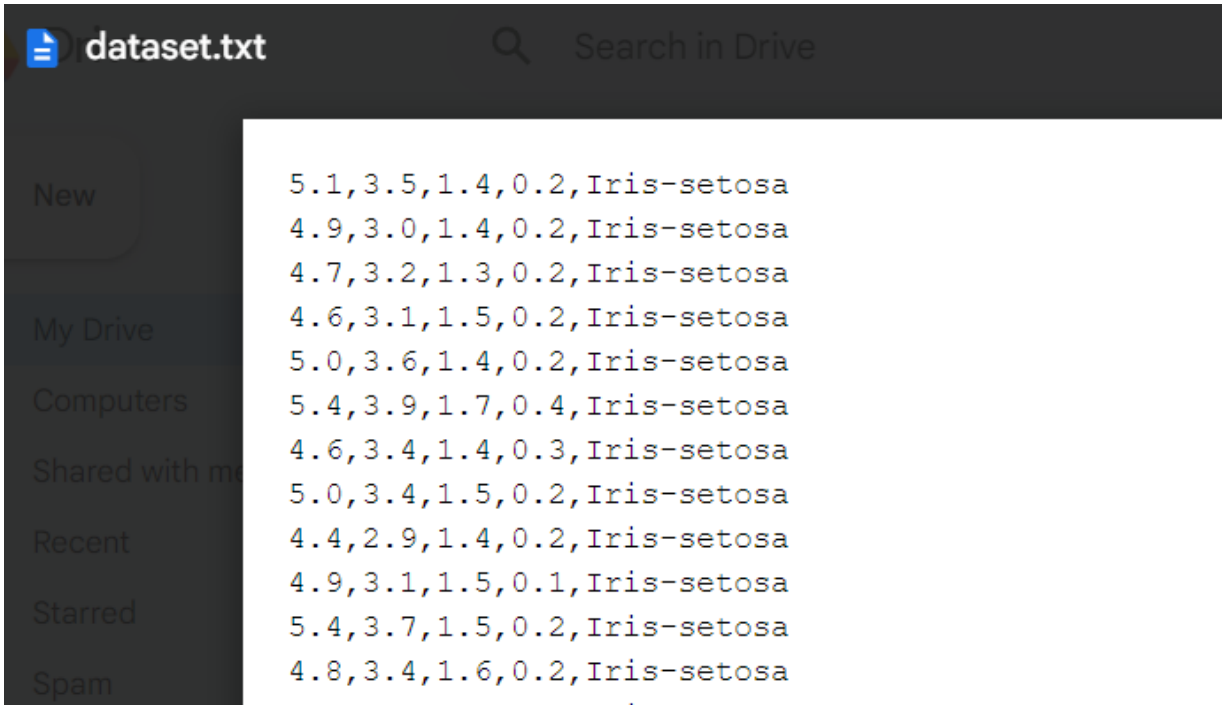
Algorithm:

* Select no K to decide the no. of clusters.
* Select random K points centroid.
* Assign each data pt to their closest centroid.
* Calc the variance & place a new centroid of each cluster
* Repeat the 3 steps, which means reassign each datapoint to the new closest centroid of each cluster
* If any reassignment occurs, then go to Step 4 else finish
* Model is ready.

O/P:



Uploading CSV:



Program Execution:

```
def ReadData(fileName):
    f = open(fileName,'r')
    lines = f.read().splitlines()
    f.close()

    items = []

    for i in range(1,len(lines)):
        line = lines[i].split(',')
        itemFeatures = []

        for j in range(len(line)-1):
            v = float(line[j])
            itemFeatures.append(v)
        items.append(itemFeatures)

    shuffle(items)

    return items
```

```
[ ] def FindColMinMax(items):
    n = len(items[0])
    minima = [float('inf') for i in range(n)]
    maxima = [float('-inf') -1 for i in range(n)]

    for item in items:
        for f in range(len(item)):
            if(item[f] < minima[f]):
                minima[f] = item[f]

            if(item[f] > maxima[f]):
                maxima[f] = item[f]

    return minima,maxima
```

```
[ ]  def EuclideanDistance(x,y):
         S = 0
         for i in range(len(x)):
             S += math.pow(x[i]-y[i],2)

         return math.sqrt(S)
```

```
     def InitializeMeans(items,k,cMin,cMax):
         f = len(items[0])
         means = [[0 for i in range(f)] for j in range(k)]

         for mean in means:
             for i in range(len(mean)):
                 mean[i] = uniform(cMin[i]+1,cMax[i]-1)

         return means
```

```
[ ]  def UpdateMean(n,mean,item):
         for i in range(len(mean)):
             m = mean[i]
             m = (m*(n-1)+item[i])/float(n)
             mean[i] = round(m,3)

         return mean
```

```
[ ]  def FindClusters(means,items):
         clusters = [[] for i in range(len(means))]

         for item in items:
             index = Classify(means,item)
             clusters[index].append(item)

         return clusters
```

```
def Classify(means,item):

    minimum = float('inf');
    index = -1

    for i in range(len(means)):
        dis = EuclideanDistance(item,means[i])

        if(dis < minimum):
            minimum = dis
            index = i

    return index
```

```
[ ] def CalculateMeans(k,items,maxIterations=100000):
        cMin, cMax = FindColMinMax(items)

        means = InitializeMeans(items,k,cMin,cMax)

        clusterSizes = [0 for i in range(len(means))]

        belongsTo = [0 for i in range(len(items))]

        for e in range(maxIterations):
            noChange = True;
            for i in range(len(items)):
                item = items[i];
                index = Classify(means,item)
                clusterSizes[index] += 1
                cSize = clusterSizes[index]
                means[index] = UpdateMean(cSize,means[index],item)

                if(index != belongsTo[i]):
                    noChange = False
                belongsTo[i] = index

            if (noChange):
                break

        return means
```

```
[ ]  def CutToTwoFeatures(items,indexA,indexB):
         n = len(items)
         X = []
         for i in range(n):
             item = items[i]
             newItem = [item[indexA],item[indexB]]
             X.append(newItem)

         return X
```

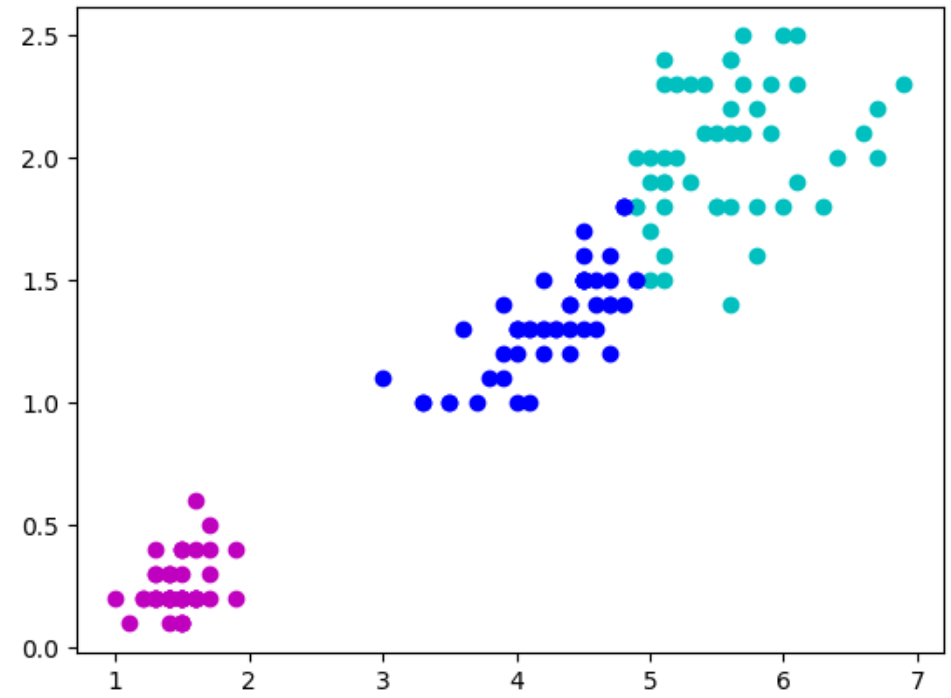
```
     def PlotClusters(clusters):
         n = len(clusters)
         X = [[] for i in range(n)]

         for i in range(n):
             cluster = clusters[i]
             for item in cluster:
                 X[i].append(item)

         colors = ['r','b','g','c','m','y']

         for x in X:
             c = choice(colors)
             colors.remove(c)

             Xa = []
             Xb = []

             for item in x:
                 Xa.append(item[0])
                 Xb.append(item[1])

             pyplot.plot(Xa,Xb,'o',color=c)

         pyplot.show()
```

```
[ ]      def main():
           items = ReadData('data.txt')
           k = 3
           items = CutToTwoFeatures(items,2,3)
           print(items)
           means = CalculateMeans(k,items)
           print("\nMeans = ", means)

           clusters = FindClusters(means,items)

           PlotClusters(clusters)
           newItem = [1.5,0.2]
           print(Classify(means,newItem))
```

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

Output:

```
[[5.1, 1.8], [5.0, 1.9], [1.6, 0.6], [4.5, 1.5], [3.7, 1.0], [1.5, 0.2], [5.6, 1.8], [4
Means =  [[5.553, 2.013], [1.462, 0.258], [4.228, 1.325]]
```



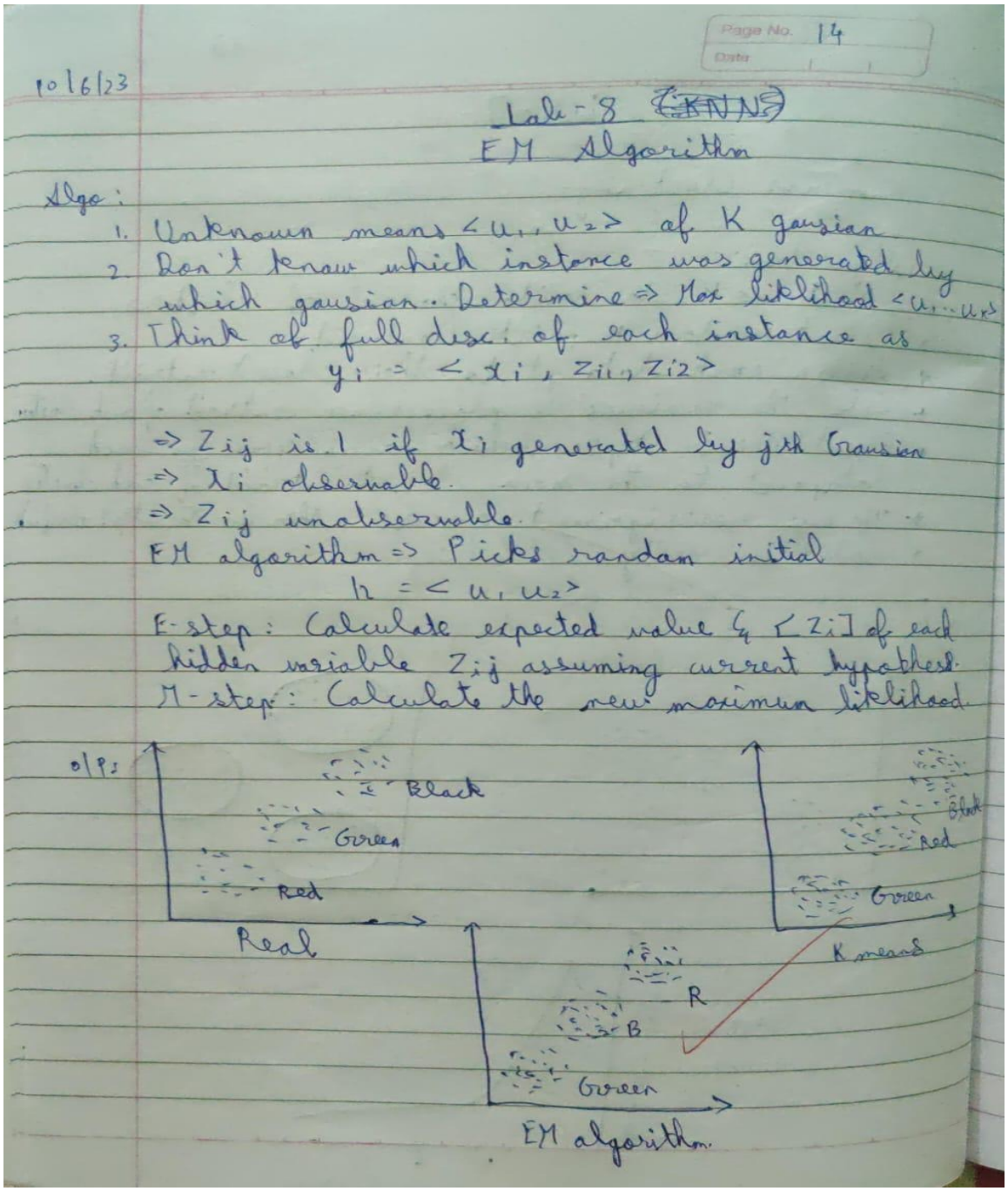

```
1
```

Date: 10.06.2023

Program 8 – Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

Data set: Iris Dataset

Observation:



Program Execution:

```
[ ]  import matplotlib.pyplot as plt
     from sklearn import datasets
     from sklearn.cluster import KMeans
     import sklearn.metrics as sm
     import pandas as pd
     import numpy as np
```

```
[ ]  iris = datasets.load_iris()
```

```
[ ]  X = pd.DataFrame(iris.data)
     X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
```

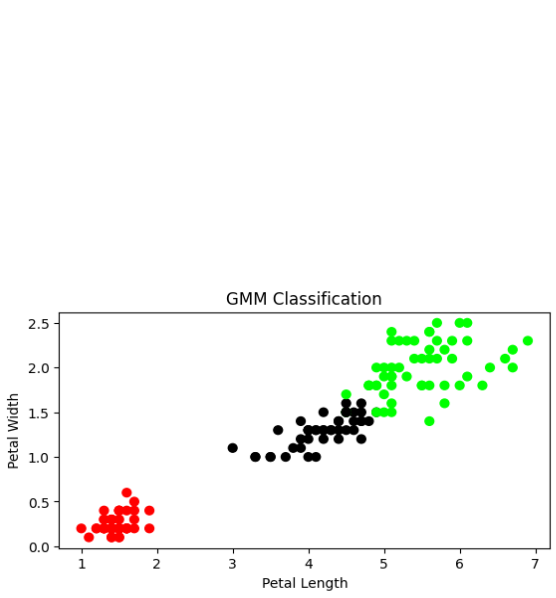
```
     y = pd.DataFrame(iris.target)
     y.columns = ['Targets']

     model = KMeans(n_clusters=3)
     model.fit(X)
```

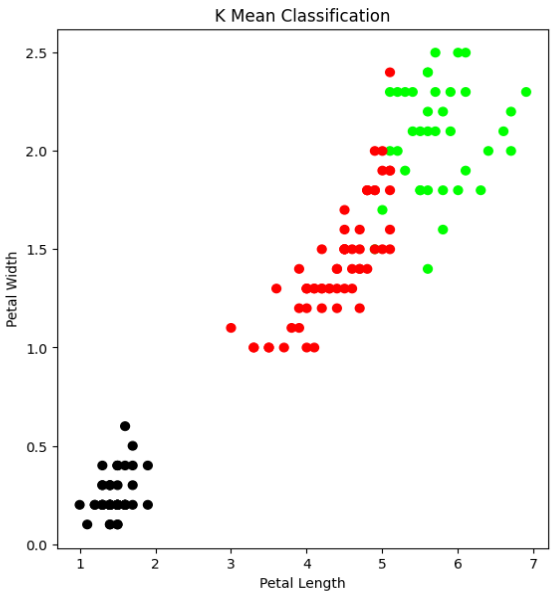
```
[ ]  plt.figure(figsize=(14,7))

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

     # Plot the Original Classifications
     plt.subplot(1, 2, 1)
     plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
     plt.title('Real Classification')
     plt.xlabel('Petal Length')
     plt.ylabel('Petal Width')
     # Plot the Models Classifications
     plt.subplot(1, 2, 2)
     plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
     plt.title('K Mean Classification')
     plt.xlabel('Petal Length')
     plt.ylabel('Petal Width')
     print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
     print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))
```

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

Output:

```
The accuracy score of K-Mean:  0.013333333333333334
The Confusion matrixof K-Mean:  [[ 0  0 50]
 [48  2  0]
 [14 36  0]]
The accuracy score of EM:  0.36666666666666664
<ipython-input-1-36b47edc0eea>:56: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two minor rele
  plt.subplot(2, 2, 3)
The Confusion matrix of EM:  [[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
```





Date: 17.06.2023

Program 9– Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

Data set:

```
dataset = [
    [50,180,0],
    [51,167,0],
    [62,182,1],
    [69,176,1],
    [64,173,1],
    [65,172,1],
    [56,174,0],
    [57,173,1],
    [55,170,1],
    [49,182,0]
]
dataset
```

Observation:

Page No. 19
Date

17/6/23

Lab - 9 (KNN)

```
def Euclidean_dist(r1,r2):
d = 0
for i in range (len(r1)-1):
    d += (r1[i] - r2[i]) ** 2
return sort(d)

def Get-neighbours (train, test, num):
    d = []
    data = []
    for i in range (len(r1)-1):
        d += (r1[i] - r2[i
    for i in main:
        dist = Euclidean_dist(test, i)
        d.append(i)
    d = np.array(d)
    data = np.array(data)
    index_dist = d.argsort()
    data = data[index_dist]
    neighbours = data[:num]
    return neighbours
```

o/p: test = [60, 190]

Dataset:

```
[[50, 180, 0]
 [51, 167, 0]
 [62, 182, 1]
 [69, 176, 1]]
```

o/p :-
Expected 1, Got 1

Program Execution:

```
[ ]  def Euclidean_distance(row1, row2):
         distance = 0
         for i in range(len(row1)-1):
             distance += (row1[i] - row2[i])**2            #(x1-x2)**2+(y1-y2)**2
         return sqrt(distance)
```

```
     def Get_Neighbors(train, test_row, num):
         distance = list() # []
         data = []
         for i in train:
             dist = Euclidean_distance(test_row, i)
             distance.append(dist)
             data.append(i)
         distance = np.array(distance)
         data =np.array(data)
         index_dist = distance.argsort()
         data = data[index_dist]
         neighbors = data[:num]
         return neighbors
```

```
[ ]  def predict_classification(train, test_row, num):
         Neighbors = Get_Neighbors(train, test_row, num)
         Classes = []
         for i in Neighbors:
             Classes.append(i[-1])
         prediction = max(Classes, key= Classes.count)
         return prediction
```

```
[ ]  test=[60,150];
```

```
     prediction = predict_classification(dataset, test, 4)
     print("Got {}".format(prediction))
```

Output:

```
     test=[60,150];
```

```
[ ]  prediction = predict_classification(dataset, test, 4)
     print("Got {}".format(prediction))
```

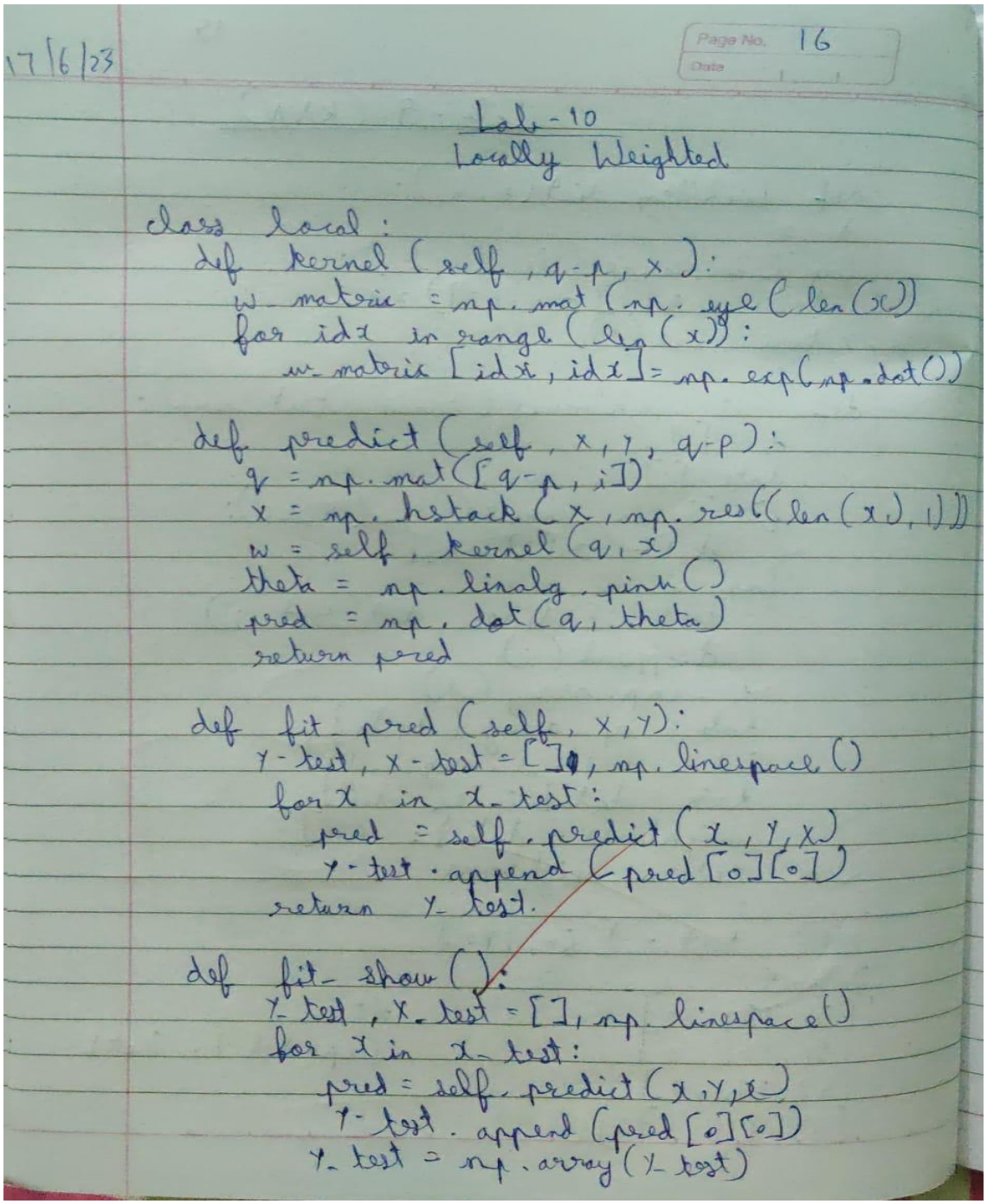
```
Got 1
```

Date: 17.05.2023

Program 10:

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

Observation:

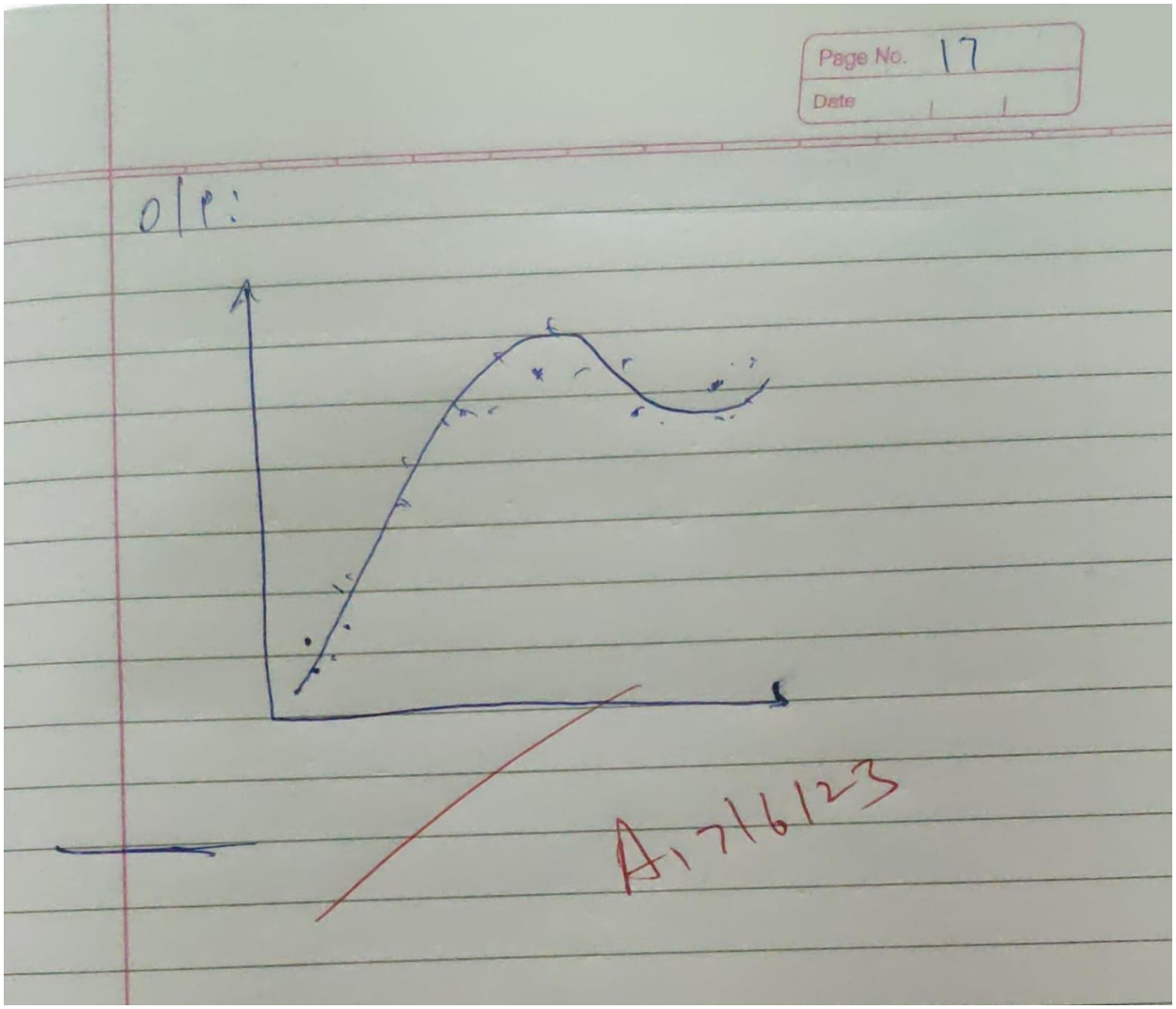


17/6/23

Page No. 16
Date

Lab-10
Locally Weighted

```
class local:
    def kernel(self, q-p, x):
        w_matrix = np.mat(np.eye(len(x)))
        for idx in range(len(x)):
            w_matrix[idx, idx] = np.exp(np.dot())

    def predict(self, x, y, q-p):
        q = np.mat([q-p, 1])
        x = np.hstack(x, np.ones((len(x), 1)))
        w = self.kernel(q, x)
        theta = np.linalg.pinv()
        pred = np.dot(q, theta)
        return pred

    def fit_pred(self, x, y):
        y_test, x_test = [], np.linspace()
        for x in x_test:
            pred = self.predict(x, y, x)
            y_test.append(pred[0][0])
        return y_test.

    def fit_show():
        y_test, x_test = [], np.linspace()
        for x in x_test:
            pred = self.predict(x, y, x)
            y_test.append(pred[0][0])
        y_test = np.array(y_test)
```


Page No. 17
Date
O/P:
A, 7/6/23

Program Execution:

```
class LocallyWeightedRegression:
    def __init__(self, tau = 0.01):
        self.tau = tau
    def kernel(self, query_point, X):
        Weight_matrix = np.mat(np.eye(len(X)))
        for idx in range(len(X)):
            Weight_matrix[idx,idx] = np.exp(np.dot(X[idx]-query_point, (X[idx]-query_point).T)/(-2*self.tau*self.tau))
        return Weight_matrix
    # function that makes the predictions of the output of a given query point
    def predict(self, X, Y, query_point):
        q = np.mat([query_point, 1])
        X = np.hstack((X, np.ones((len(X), 1))))
        W = self.kernel(q, X)
        theta = np.linalg.pinv(X.T*(W*X))*(X.T*(W*Y))
        pred = np.dot(q, theta)
        return pred
    #function that fits and predicts the output of all query points
    def fit_and_predict(self, X, Y):
        Y_test, X_test = [], np.linspace(-np.max(X), np.max(X), len(X))
        for x in X_test:
            pred = self.predict(X, Y, x)
            Y_test.append(pred[0][0])
        Y_test = np.array(Y_test)
        return Y_test
    # function that computes the score rmse
    def score(self, Y, Y_pred):
        return np.sqrt(np.mean((Y-Y_pred)**2))
    # function that fits as well as shows the scatter plot of all points
    def fit_and_show(self, X, Y):
        Y_test, X_test = [], np.linspace(-np.max(X), np.max(X), len(X))
        for x in X_test:
            pred = self.predict(X, Y, x)
            Y_test.append(pred[0][0])
        Y_test = np.array(Y_test)
        plt.style.use('seaborn')
        plt.title("The scatter plot for the value of tau = %.5f"% self.tau)
        plt.scatter(X, Y, color = 'red')
        plt.scatter(X_test, Y_test, color = 'green')
        plt.show()
# reading the csv files of the given dataset
dfx = pd.read_csv('/content/weightedX.csv')
dfy = pd.read_csv('/content/weightedY.csv')
# store the values of dataframes in numpy arrays
X = dfx.values
```

Output:


The scatter plot for the value of tau = 0.20000
2
1
0
−1
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5