

NAME: SIRI PRAHLAD
USN: 1BM20CS160

MACHINE LEARNING LAB OBSERVATION

Date: 1-04-2023

Lab 1: Exploring Datasets

IRIS DATASET:

- 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 [8]: from sklearn.datasets import load_iris
iris=load_iris()

In [9]: print(iris)

{'data': array([[5.1, 3.5, 1.4, 0.2],
   [4.9, 3. , 1.4, 0.2],
   [4.7, 3.2, 1.3, 0.2],
   [4.6, 3.1, 1.5, 0.2],
   [5. , 3.6, 1.4, 0.2],
   [5.4, 3.9, 1.7, 0.4],
   [4.6, 3.4, 1.4, 0.3],
   [5. , 3.4, 1.5, 0.2],
   [4.4, 2.9, 1.4, 0.2],
   [4.9, 3.1, 1.5, 0.1],
   [5.4, 3.7, 1.5, 0.2],
   [4.8, 3.4, 1.6, 0.2],
   [4.8, 3. , 1.4, 0.1],
   [4.3, 3. , 1.1, 0.1],
   [5.8, 4. , 1.2, 0.2],
   [5.7, 4.4, 1.5, 0.4],
   [5.4, 3.9, 1.3, 0.4],
   [5.1, 3.5, 1.4, 0.3],
   [5.7, 3.8, 1.7, 0.3],
   [5.1, 3.0, 1.5, 0.2],
   [4.7, 3.2, 1.3, 0.2],
   [4.5, 2.3, 1.3, 0.3],
   [4.4, 3.2, 1.3, 0.2],
   [5. , 3.5, 1.6, 0.6],
   [5.1, 3.8, 1.9, 0.4],
   [4.8, 3. , 1.4, 0.3],
   [5.1, 3.8, 1.6, 0.2],
```



```
In [5]: type(iris)
Out[5]: function

In [12]: iris.keys()

Out[12]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])

In [13]: iris

[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
```

```
In [17]: print(iris['target_names'])  
['setosa' 'versicolor' 'virginica']
```

```
In [20]: n_samples,n_features=iris.data.shape  
print("no.of samples:",n_samples)  
print("no.of features:",n_features)  
  
no.of samples: 150  
no.of features: 4
```

```
In [28]: iris.data[[12,26,89,114]]
```

```
Out[28]: array([[4.8, 3. , 1.4, 0.1],  
                 [5. , 3.4, 1.6, 0.4],  
                 [5.5, 2.5, 4. , 1.3],  
                 [5.8, 2.8, 5.1, 2.4]])
```

```
In [29]: print(iris.data.shape)
```

```
(150, 4)
```

```
In [31]: print(iris.target.shape)
```

```
(150,)
```

```
In [32]: import numpy as np  
np.bincount(iris.target)
```

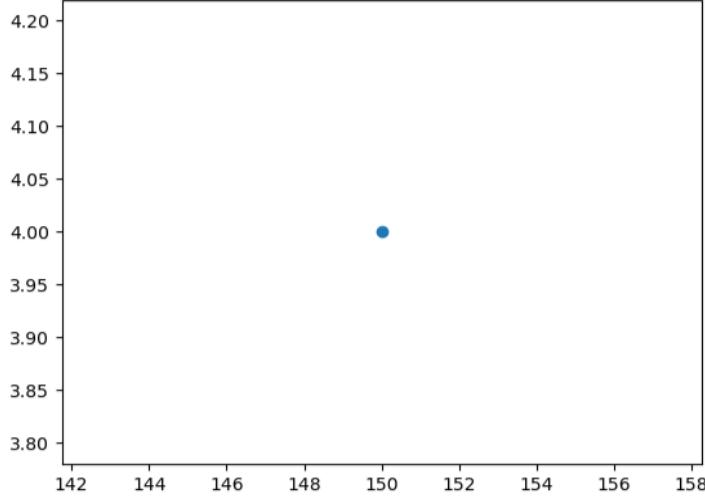
Scattered graph for samples vs features.

```
In [32]: import numpy as np  
np.bincount(iris.target)
```

```
Out[32]: array([50, 50, 50], dtype=int64)
```

```
In [42]: import matplotlib.pyplot as plt  
plt.scatter(n_samples,n_features)
```

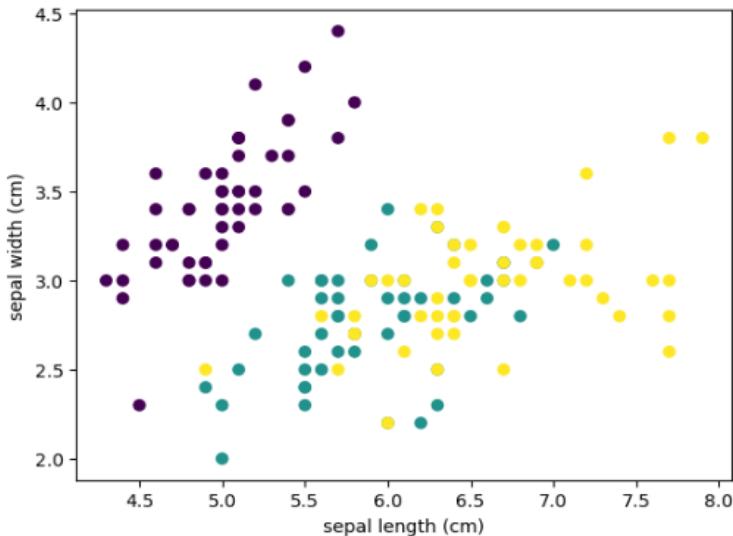
```
Out[42]: <matplotlib.collections.PathCollection at 0x1d1c8c45550>
```



Scattered graph: with first two features(septal width vs septal length)
The three colors represents three different classes respectively.

In [47]:

```
features = iris.data.T  
  
plt.scatter(features[0], features[1],  
           c=iris.target)  
plt.xlabel(iris.feature_names[0])  
plt.ylabel(iris.feature_names[1]);
```



In [49]: `iris.data[[1,2,3,4,5]]`

Out[49]: `array([[4.9, 3., 1.4, 0.2],
[4.7, 3.2, 1.3, 0.2],
[4.6, 3.1, 1.5, 0.2],
[5., 3.6, 1.4, 0.2],
[5.4, 3.9, 1.7, 0.4]])`

WINE DATASET:

In [51]: `from sklearn.datasets import load_wine
wine=load_wine()`

In [52]: `print(wine)`

```
{'data': array([[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,
```

In [57]: `wine.data`

Out[57]: `array([[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 [58]: `wine.keys()`

Out[58]: `dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])`

In [60]: `print(wine['target_names'])`

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

```
In [9]: print(wine['feature_names'])

['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proa
nthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']
```

```
In [11]: import numpy as np
np.bincount(wine.target)
```

```
Out[11]: array([59, 71, 48], dtype=int64)
```

Date: 15/04/2023

Lab 2: FIND-S ALGORITHM FOR ENJOY SPORT:

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:Enjoysport

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:

initialize h to the most specific hypothesis in H $h-(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$

1. First training example $X_1 = < \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} >$. EnjoySport=+ve Observing. The first training example, it is clear that hypothesis h is too specific. None of the " \emptyset " constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example $h_1 = < \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} >$.
2. Consider the second training example $x_2 < \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Warm}, \text{Same} >$. 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 $h_2 = < \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} >$
3. Consider the third training example $x_3 < \text{Rainy}, \text{Cold}, \text{High}, \text{Strong}, \text{Warm}, \text{Change} >$. EnjoySport ve. The FIND-S algorithm simply ignores every negative example. So the hypothesis remain as before, so $h_3 = < \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} >$
4. Consider the fourth training example $x_4 < \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Cool}, \text{Change}, \text{EnjoySport} >$ +ve. The fourth example leads to a further generalization of h as $h_4 = < \text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ? >$
5. So the final hypothesis is $< \text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ? >$

5/4/23

 classmate
 Date 3
 Page 3

Lab Program 1

Find S algorithm

Dataset : enjoysports.csv file

Sample	Sky	Air Temp	Humidity	Wind	Water	Forecast	enjoys sports?
1) Sunny	Sunny	Warm	normal	Strong	warm	Same	Yes +
2) Sunny	Sunny	Warm	high	Strong	warm	Same	Yes +
3) Sunny	Sunny	Cold	high	Strong	warm	Same	No -
4) Sunny	Sunny	Warm	high	Strong	warm	Same	Yes +

* Find S algorithm: It's a basic-concept-learning algo in ML.

* It finds what is most-specific hypothesis that fits all the "Positive" examples.

* This algo starts with the most specific hypothesis and moves to the most general hypothesis.

? → accepts any value General.

∅ → accepts No value. Specific(value)

MSD → (?? ??) accepts everything.

MSD → (∅ ∅ ∅ ∅) accepts none

initial hypo : (∅ ∅ ∅ ∅) → Null.

iteration 1 h₁ = < 'Sunny', 'warm', 'normal', 'strong', 'warm', 'Same' > +ve

iteration 2 h₂ = < 'Sunny', 'warm', 'high', 'strong', 'warm', 'Same' >

iteration 3 h₃ = < 'Rainy', 'cold', 'high', 'strong', 'warm', 'change' >

iteration 4 h₄ = < 'Sunny', 'warm', 'high', 'strong', 'cool', 'change' >

(Not considered) 1. x) similar to h₄

(and 'rainy', 'cool', 'change')

i) Initialize 'h' to the most specific hypo in H.

2) For each positive training instance 'x' each attribute constraint as in 'h'. If the constraint a_i is not satisfied by 'x' then do nothing. Else replace a_i in 'h' by the next more general constraint that is required by 'x' hypothesis 'h'.

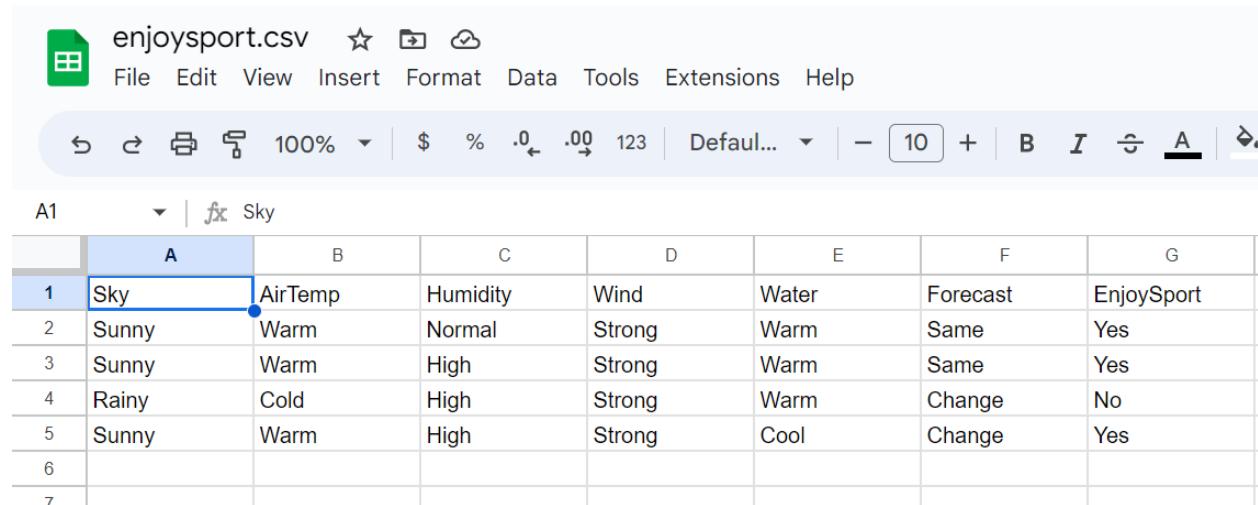
3) Output hypothesis 'h'.

Program

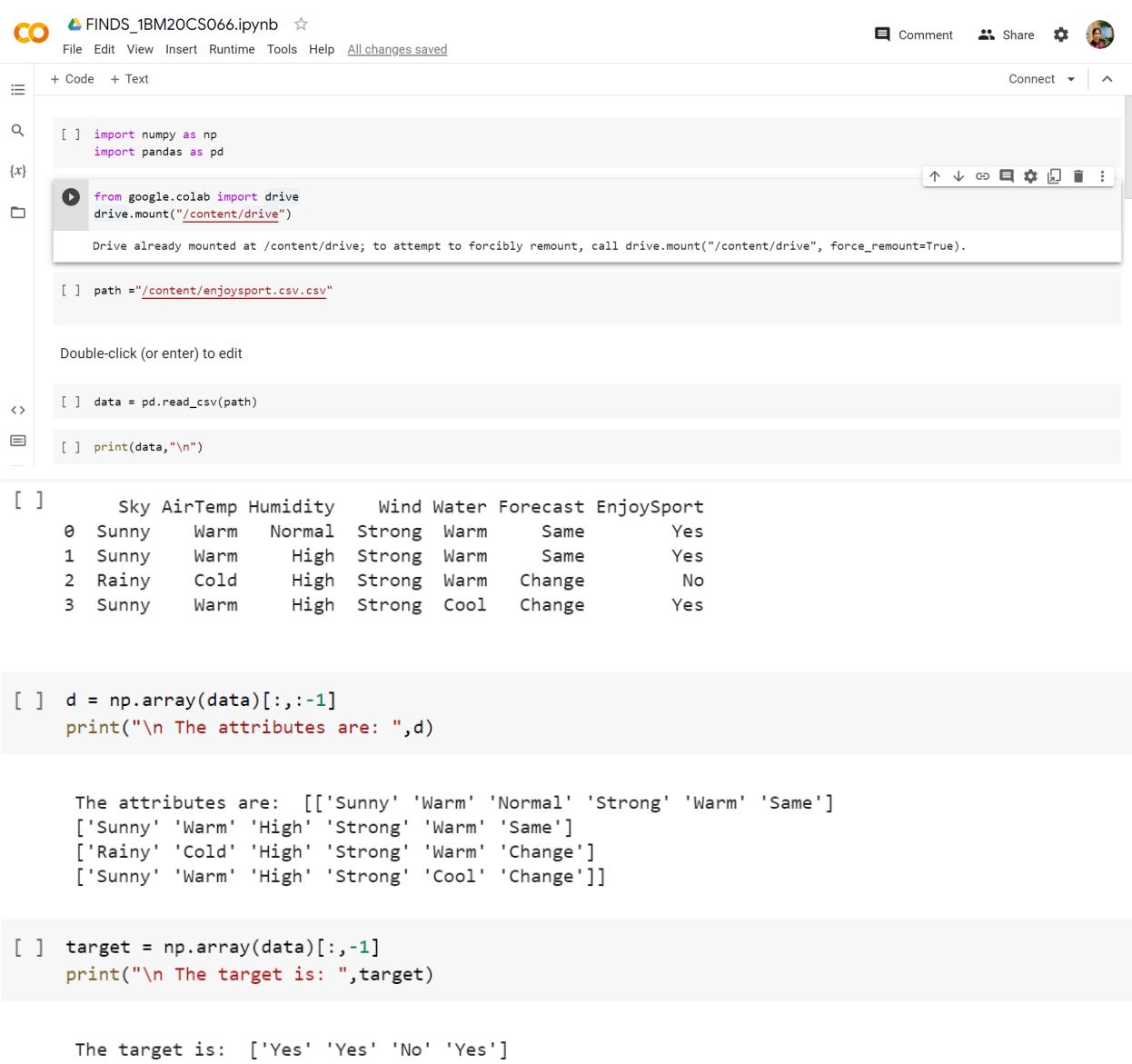
```

import csv
def updateHypothesis(x_h):
    if h == []:
        return x
    for i in range(0, len(h)):
        if x[i].upper() != h[i].upper():
            h[i] = '?'
    return h
if name == 'main':
    data = []
    h = []
    with open("Desktop\FindS.csv", "r") as file:
        reader = csv.reader(file)
        print("Data:")
        for row in reader:
            data.append(row)
            print(row)
    if data:
        for x in data:
            if x[-1].upper() == "yes": x.pop()
            updateHypothesis(x, h)
        print("In Hypothesis:", h)
    
```

CREATING CSV FILE:



	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes
3	Sunny	Warm	High	Strong	Warm	Same	Yes
4	Rainy	Cold	High	Strong	Warm	Change	No
5	Sunny	Warm	High	Strong	Cool	Change	Yes
6							
7							



```
[ ] import numpy as np
import pandas as pd
from google.colab import drive
drive.mount("/content/drive")
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
[ ] path ="/content/enjoysport.csv.csv"
Double-click (or enter) to edit
[ ] data = pd.read_csv(path)
[ ] print(data,"\n")
[ ] Sky AirTemp Humidity Wind Water Forecast 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
[ ] d = np.array(data)[:, :-1]
print("\n The attributes are: ",d)

The 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']]

[ ] target = np.array(data)[:, -1]
print("\n The target is: ",target)

The target is: ['Yes' 'Yes' 'No' 'Yes']
```

```
[ ] def finds(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

print("\n The final hypothesis is:",finds(d,target))
```

The final hypothesis is: ['Sunny' 'Warm' '?' 'Strong' '?' '?']

SECOND DATASET: FIND-S ALGORITHM

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

CREATING CSV FILE

A1	A	B	C	D	E	F
1	citation	size	inLibrary	price	editions	buy
2	some	small	no	affordable	many	no
3	many	big	no	expensive	one	yes
4	some	big	always	expensive	few	no
5	many	medium	no	expensive	many	yes
6	many	small	noo	affordable	many	yes
7						
8						

```
▶ import numpy as np  
import pandas as pd
```

```
[ ] from google.colab import drive  
drive.mount("/content/drive")
```

Mounted at /content/drive

```
[ ] path ="/content/finds_1BM20CS066 - Sheet1.csv"
```

```
[ ] data = pd.read_csv(path)
```

```
[ ] print(data, "\n")
```

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

```
[ ] d = np.array(data)[:, :-1]  
print("\n The attributes are: ",d)
```

The attributes are: [['some' 'small' 'no' 'affordable' 'many']
['many' 'big' 'no' 'expensive' 'one']
[['some' 'big' 'always' 'expensive' 'few']]
[['many' 'medium' 'no' 'expensive' 'many']]
[['many' 'small' 'noo' 'affordable' 'many']]

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

The target is: ['no' 'yes' 'no' 'yes' 'yes']

+ Code + Text

```
[ ] def find_s(d,target):  
    for i, val in enumerate(target):  
        if val=='yes':  
            hypothesis=d[i].copy()  
            break  
  
    for i,var in enumerate(d):  
        if target[i]=="yes":  
            for x in range(len(hypothesis)):  
                if var[x]!=hypothesis[x]:  
                    hypothesis[x]='?'  
                else:  
                    pass  
  
    return hypothesis  
  
print("The Hypothesis is",find_s(d,target))
```

The Hypothesis is ['many' '?' '?' '?' '?']

DATE: 15/04/2023

LAB 3: CANDIDATE- ELIMINATION- ENJOY SPORT

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 the training examples.
Data set: Enjoy sport

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:

Step1: Load Data set

Step2: Initialize General Hypothesis and Specific Hypothesis.

Step3: For each training example

Step4: If example is positive example

 if attribute_value == hypothesis_value:

 Do nothing

 else:

 replace attribute value with '?' (Basically generalizing it)

Step5: If example is Negative example

 Make generalize hypothesis more specific.

12/4/23

classmate

Date _____

Page _____

5

Lab Program 2

Candidate Elimination Algorithm

	Example	Sky	Air	temp	Humidity	Wind	Water
1	Sunny	Sunny	Normal	High	Strong	Warm	Weak
2	Sunny	Sunny	Normal	High	Strong	Weak	Weak
3	Rainy	Rainy	Cold	High	Strong	Weak	Weak
4	Sunny	Sunny	Normal	High	Strong	Weak	Weak

forecast	Enjoy Sport?	Target variable
Same	Yes +ve	variable
Same	Yes +ve	6 attributes candidate
Change	No -ve	concept learning
Change	Yes +ve	

gives out binary results.

Considers both negative and Positive values.

To find consistent hypothesis for a given solution of training example

most General $G_0 = < ?, ?, ?, ?, ?, ? >$

most Specific $S_0 = < \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$

Start from Generic Boundary,

first takes generic attribute values.

Whenever matched retain generic values if hypothesis matches expected is +ve and outcome +ve.

$$G_1 = G_0$$

A null value in S_0 is replaced by S_1 .

$$S_1 = < \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$$

No match \rightarrow negative classification

* All question marks match with example, hence +ve classification. Target variable = +ve \Rightarrow GB

- * All null values doesn't match, hence -ve classifier. But expected = +ve classifier, therefore it is inconsistent hypothesis. When inconsistency exists write next general hypothesis that is:

→ Replace Null values by 1st examples.

I) $G_1 = \langle ?, ?, ?, ?, ?, ?, ? \rangle$

$S_1 = \langle \text{Sunny, warm, Normal, Strong, warm, same} \rangle$

II) $G_1 = \langle ?, ?, ?, ?, ?, ?, ? \rangle$

Consider prev generic hypothesis.

$G_2 = G_1$

if generic +ve → retain all

if match retain

if Target value -ve → start from S.

if Target value +ve → start from G.

$S_2 = \langle \text{Sunny, warm, ?, Strong, warm, same} \rangle$

- * 6B, all ? matches with S_1 , hence +ve classification and expected.

- * SB, when inconsistency make it General (?)

III) $S_3 = \langle \text{Sunny, warm, ?, Strong, warm, same} \rangle$ ^{Target value} -ve

$G_3 = \{ \langle \text{Sunny, ?, ?, ?, ?, ?} \rangle \}$

- * Since all values are generic in previous hypothesis, only possible when example is +ve. And if there exists inconsistency, then all hypotheses which are consistent with all the training examples are now.

* ? match with all the attribute but expected -ve, hence inconsistency.

* All hypothesis which are consistent till now.
→ To do that, consider 1 ? at a time.

```
Program
import numpy as np
import Pandas as pd.
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
Concepts = np.array(data.iloc[:, 0:-1])
print(Concepts)
target = np.array(data.iloc[:, -1])
print(general-h)
print(specific-h).
```

$x_4(+)$

$s_4 = \langle ?, \text{large}, \text{light}, ?, \text{thick} \rangle$

$g_4 = \langle ?, ?, \text{light}, ??, \langle ??? \text{thick} \rangle$

Dataset

Size	Trunk	Fuel economy	No of Passengers	Type	Target value.
Small	Available	High	4	economy	Y
Big	Available	Low	2	Sports	N
Small	Available	High	4	economy	Y
Small	Not Available	Low	2	Sports	N

$g_0 = \langle ?, ?, ?, ?, ?, ? \rangle$

$s_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

$s_1 = \langle \text{small}, \text{available}, \text{high}, 4, \text{economy} \rangle$

$g_1 = \langle ?, ?, 2, ?, ? \rangle$

$s_2 = \langle \text{small}, ?, \text{high}, 4, \text{economy} \rangle$

$g_2 = \{ \langle \text{small}, ?, ?, ?, ?, ? \rangle ; \langle ?, ?, \text{high}, ?, ? \rangle ;$
 $\{ \langle ?, ?, ?, 4, ?, ? \rangle ; \langle ?, ?, ?, \text{economy} \rangle \}$

$S_3 = \langle \text{small}, ?, \text{high}, 4, \text{economy} \rangle$

$G_3 = \langle \text{small}, ?, ?, ?, ?, ? \rangle ; \langle ?, ?, \text{high}, ?, ?, ? \rangle$
 $\langle ?, ?, ?, 4, ? \rangle ; \langle ?, ?, ?, \text{economy} \rangle$

$S_4 = \langle ?, ?, \text{high}, 4, \text{economy} \rangle$

$G_4 = \langle ?, ?, \text{high}, ?, ?, ? \rangle ; \langle ?, ?, ?, 4, ? \rangle ;$
 $\langle ?, ?, ?, ?, \text{economy} \rangle$

CREATING CSV FILE:

	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes
3	Sunny	Warm	High	Strong	Warm	Same	Yes
4	Rainy	Cold	High	Strong	Warm	Change	No
5	Sunny	Warm	High	Strong	Cool	Change	Yes
6							
7							

```
[ ] import numpy as np
import pandas as pd

[ ]
from google.colab import drive
drive.mount('/content/drive')

[ ]
data = pd.DataFrame(data=pd.read_csv('/content/enjoysport.csv.csv'))

[ ]
print(data,"\n")

   Sky  AirTemp  Humidity  Wind  Water  Forecast  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
```

```
[ ] concepts = np.array(data.iloc[:,0:-1])

[ ]
print(concepts)

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

[ ] target = np.array(data.iloc[:,-1])
print(target)

['Yes' 'Yes' 'No' 'Yes']

[ ] import csv
```


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.

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

ALGORITHM:

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples
- Otherwise Begin
 - $A \leftarrow$ the attribute from Attributes that best* classifies Examples
 - The decision attribute for Root $\leftarrow A$
 - For each possible value, v_i , of A,
 - Add a new tree branch below Root, corresponding to the test $A = v_i$
 - Let $Examples_{v_i}$ be the subset of Examples that have value v_i for A
 - If $Examples_{v_i}$ is empty
 - Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
 - Else below this new branch add the subtree $ID3(Examples_{v_i}, Target_attribute, Attributes - \{A\})$
 - End
- Return Root

19/4/23

Date _____
Page 9

Decision Tree - Play Golf

1) Day

	Outlook	Temp	Humidity	Wind	Play
1	Sunny	Hot	high	weak	No
2	Sunny	Hot	high	Strong	No
3	Overcast	Hot	high	weak	Yes
4	Rain	Mild	high	weak	Yes
5	Rain	Cool	Normal	weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

$$= (-0.11u(-0.637u)) + (-0.51u(-1.485u)) \\ = -0.941$$

Information Gain: $I(S, A, T)$

$$= E(S) - \sum_{A \in Attr(A)} \sum_{Sv} E(Sv)$$

$$\text{Entropy } (S) = -P_0 \log_2 P_0 - P_1 \log_2 P_1$$

$$G(s, \text{temp}) = E(s) - \left[\frac{4}{14} * E(\text{temp} = \text{hot}) + \frac{6}{14} * E(\text{temp} = \text{mild}) + \frac{9}{14} * E(\text{temp} = \text{cool}) \right]$$

$$E(s) = \left[\frac{4}{14} * \left(-2 \log_2 \frac{2}{4} \right) + \left(-2 \log_2 \frac{2}{4} \right) + \right.$$

$$\left. \frac{6}{14} * \left(-\frac{4}{6} \log_2 \frac{4}{6} \right) + \left(-\frac{4}{6} \log_2 \frac{4}{6} \right) + \right]$$

$$\left. \frac{9}{14} * \left(-\frac{3}{9} \log_2 \frac{3}{9} \right) + \left(-\frac{1}{9} \log_2 \frac{1}{9} \right) \right]$$

$$= 0.94 - 0.989$$

$$= \boxed{0.029}$$

$$\begin{aligned} G(s, \text{Humidity}) &= 0.151 \text{ Entropy}(s) \\ &= -(7/14) \text{ entropy } (\text{S high}) \\ &\quad -(7/14) \text{ entropy } (\text{S normal}) \\ &= 9.40 - (7/14) \cdot 0.985 - (7/14) \cdot 0.592 \\ &= \boxed{1.004189} \end{aligned}$$

~~$$(14/14)$$~~

$$G(s, \text{wind}) = \text{Entropy}(s)$$

$$\begin{aligned} &\quad - (8/14) \text{ entropy } (\text{S weak}) \\ &\quad - (6/14) \text{ entropy } (\text{S strong}) \\ &= 0.940 - (8/14) \cdot 0.811 - (6/14) \cdot \\ &= \boxed{0.48} \end{aligned}$$

Algorithm

ID3 (Example, Target-attribute, attribute)

- * Create a root node for the tree.
- * If all examples are +ve,
return the single node tree
Root with label = +ve.
- * If all examples are -ve,
return the single node tree
Root with label = -ve.
- * Otherwise Begin,

→ A \leftarrow the attribute from attributes that best*
Classifies examples.

→ The decision attribute for root \leftarrow A

→ Add a new tree-branch below root, corresponding to
the test $A = V^i$

→ Let example V^i , be the subset of example that have
values V^i from A.

→ If example V^i , that is empty.

- Then a below this new branch at a leaf
node with label S' most common value of
Target-attribute in example.

→ Else, below this new branch add the subtree
ID3 (example: V_i , target attribute, attribute_{A_j})

* End

* Return Root.

O/P See

13/5/23.

1BM20CS066_ID3

	A	B	C	D	E
1	outlook	temperature	humidity	wind	play tennis
2	sunny	hot	high	weak	no
3	sunny	hot	high	strong	no
4	overcast	hot	high	weak	yes
5	rain	mild	high	weak	yes
6	rain	cool	normal	weak	yes
7	rain	cool	normal	strong	no
8	overcast	cool	normal	strong	yes
9	sunny	mild	high	weak	no
10	sunny	cool	normal	weak	yes
11	rain	mild	normal	weak	yes
12	sunny	mild	normal	strong	yes
13	overcast	mild	high	strong	yes
14	overcast	hot	normal	weak	yes
15	rain	mild	high	strong	no
16					

Files

Search [x]

- ..
- sample_data
- 1BM20CS066_ID3.csv

+ Code + Text

```
✓ [53] import math
      import csv

✓ [55] def load_csv(filename):
      lines=csv.reader(open(filename,"r"))
      dataset = list(lines)
      headers = dataset.pop(0)
      return dataset,headers

✓ [56] class Node:
      def __init__(self,attribute):
          self.attribute=attribute
          self.children={}
          self.answer=""

      def subtables(data,col,delete):
          dic={}
          coldata=[row[col] for row in data]
          attr=list(set(coldata))

          counts=[0]*len(attr)
          r=len(data)
          c=len(data[0])
          for x in range(len(attr)):
              for y in range(r):
                  if data[y][col]==attr[x]:
                      counts[x]+=1

          for x in range(len(attr)):
              dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
              pos=0
              for y in range(r):
                  if data[y][col]==attr[x]:
                      if delete:
                          del data[y][col]
                      dic[attr[x]][pos]=data[y]
                      pos+=1
          return attr,dic
```

<>

☰

Disk

84.31 GB available

```
✓ [58] def entropy(s):
    attr=list(set(s))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in s if attr[i]==x])/(len(s)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums
```

```
✓ [59] def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy
```

```
✓ [60] def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]

    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node
```

```
✓  def print_tree(node,level):
    if node.answer!="":
        print(" "+*level,node.answer)
        return

        print(" "+*level,node.attribute)
        for value,n in node.children:
            print(" "+*(level+1),value)
            print_tree(n,level+2)
```

```
✓ [62] def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)
```

```
✓ [63]
dataset,features=load_csv("1BM20CS066_ID3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("1BM20CS066_ID3.csv")

for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:")
    classify(node1,xtest,features)
```

```
The decision tree for the dataset using ID3 algorithm is
outlook
  rain
    wind
      weak
        yes
        strong
        no
      sunny
        humidity
          high
          no
        normal
        yes
      overcast
        yes
```

```
The test instance: ['sunny', 'hot', 'high', 'weak', 'no']
The label for test instance:
no
The test instance: ['sunny', 'hot', 'high', 'strong', 'no']
The label for test instance:
no
The test instance: ['overcast', 'hot', 'high', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'mild', 'high', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'cool', 'normal', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'cool', 'normal', 'strong', 'no']
The label for test instance:
no
The test instance: ['overcast', 'cool', 'normal', 'strong', 'yes']
The label for test instance:
yes
The test instance: ['sunny', 'mild', 'high', 'weak', 'no']
The label for test instance:
no
The test instance: ['sunny', 'cool', 'normal', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'mild', 'normal', 'weak', 'yes']
The label for test instance:
yes
```

```

The test instance: ['sunny', 'mild', 'normal', 'strong', 'yes']
The label for test instance:
yes
The test instance: ['overcast', 'mild', 'high', 'strong', 'yes']
The label for test instance:
yes
The test instance: ['overcast', 'hot', 'normal', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'mild', 'high', 'strong', 'no']
The label for test instance:
no
```

PROGRAM 5: Simple linear regression program

Dataset used:

	A	B
1	x	y
2		1
3		2
4		3
5		4
6	5	3.75
7		2.25

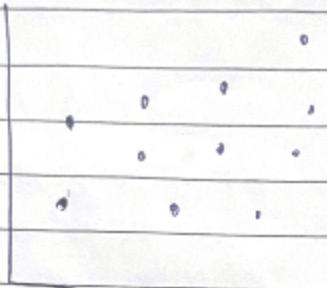
ALGORITHM:

- The main function to calculate values of coefficients
- Initialize the parameters.
- Predict the value of a dependent variable by giving an independent variable.
- Calculate the error in prediction for all data points.
- Calculate partial derivatives w.r.t a_0 and a_1 .
- Calculate the cost for each number and add them.
- Update the values of a_0 and a_1 .

Linear Regression

$$b_0 = \frac{(\sum y) (\sum x^2) - (\sum x)(\sum xy)}{n (\sum x^2) - (\sum x)^2}$$

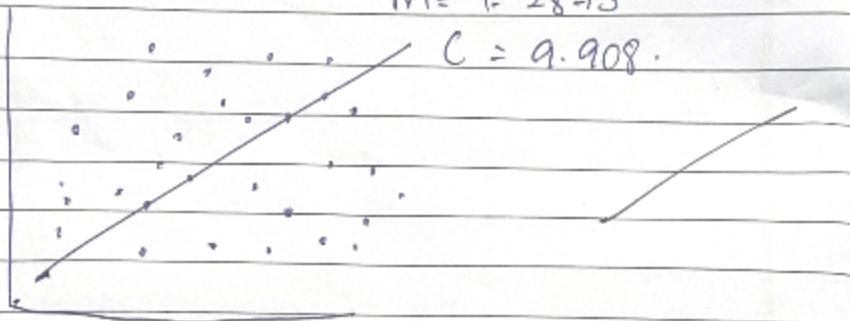
$$b_1 = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$



$$Y = MX + C$$

$$M = 1.2873$$

$$C = 9.908$$



A straight-line equation involving slope (dy/dx) and y -intercept

$$Y = MX + C$$

y = dependent value of x

$$Y = b_1 x_i + b_0$$

y_i = Predicted Y Value for observation

b_0 = Estimate of Regression intercept

b_1 = Estimate of regression slope

x_i = Input

```
[ ] import numpy as np
import matplotlib.pyplot as plt

[ ] def plot_regression_line(x, y, b):

    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)

    y_pred = b[0] + b[1]*x

    plt.plot(x, y_pred, color = "g")

    plt.xlabel('x CO-EFF')
    plt.ylabel('y CO-EFF')

    plt.show()
```

```
[ ] def estimate_coef(x, y):

    n = np.size(x)

    m_x = np.mean(x)
    m_y = np.mean(y)

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

    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)
```

```
▶ def plot_regression_line(x, y, b):
    plt.scatter(x, y, color = "b",
                marker = "*", s = 30)

    y_pred = b[0] + b[1]*x

    plt.plot(x, y_pred, color = "y")

    plt.xlabel('x')
    plt.ylabel('y')

    plt.show()
```

```

def main():

    x = np.array([1,2,3,4,5])
    y = np.array([1,2,1.3,3.75,2.25])

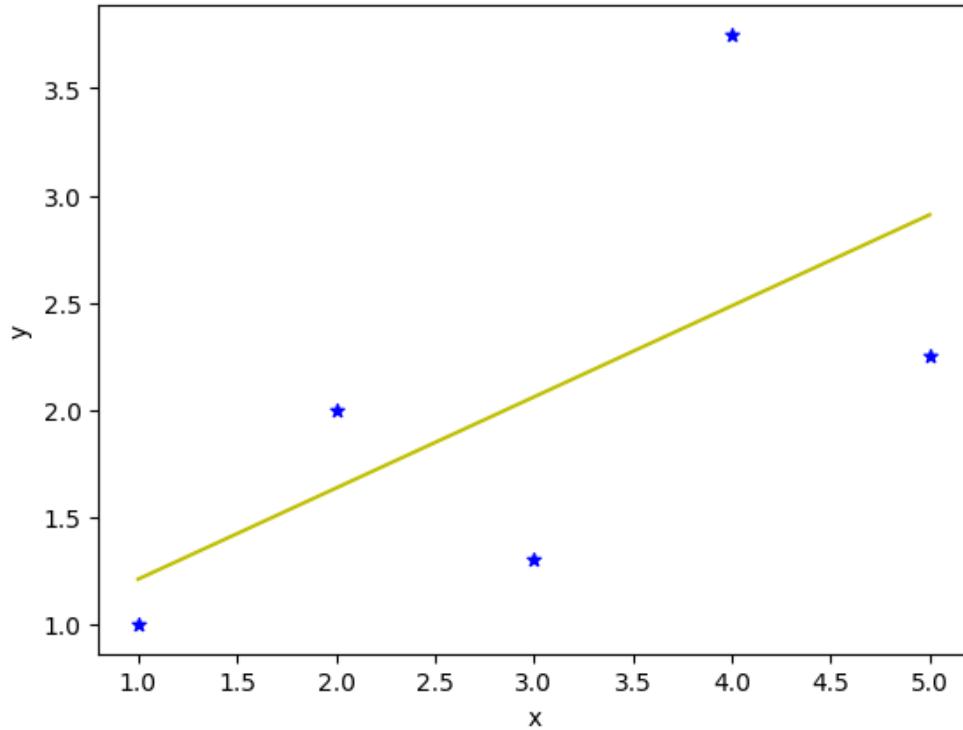
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {} \
        \nb_1 = {}".format(b[0], b[1]))

    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()

```

Estimated coefficients:
 $b_0 = 0.785000000000001$
 $b_1 = 0.4249999999999966$



Conclusion:

This model is not appropriate for this model. All the points of this dataset are away from the prediction line.

Program 6: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Data set used:

	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

Algorithm:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Formula for naive bayes classifier is as follows →

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.
4. Test accuracy of the result and visualizing the test set result.

The screenshot shows a Jupyter Notebook interface with the following details:

- File Name:** 1BM20CS066_NBC.ipynb
- Cells:** There are three code cells labeled [7], [8], and [9].
- Cell 7:** Imports numpy, math, csv, and pdb.
- Cell 8:** Contains a function `read_data(filename)` which reads a CSV file into a list of rows. It uses a context manager to open the file and a for loop to read each row into `traindata`.
- Cell 9:** Contains a function `splitDataset(dataset, splitRatio)` which splits the dataset into a training set and a test set based on the specified ratio.

```
✓ 0s def classify(data,test):  
  
    total_size = data.shape[0]  
    print("\n")  
    print("training data size=",total_size)  
    print("test data size=",test.shape[0])  
  
    countYes = 0  
    countNo = 0  
    probYes = 0  
    probNo = 0  
    print("\n")  
    print("target      count      probability")  
  
    for x in range(data.shape[0]):  
        if data[x,data.shape[1]-1] == 'Yes':  
            countYes +=1  
        if data[x,data.shape[1]-1] == 'No':  
            countNo +=1  
  
    probYes=countYes/total_size  
    probNo= countNo / total_size  
  
    print('Yes','\t',countYes,'\t',probYes)  
    print('No','\t',countNo,'\t',probNo)  
  
    prob0 =np.zeros((test.shape[1]-1))  
    prob1 =np.zeros((test.shape[1]-1))  
    accuracy=0  
    print("\n")  
    print("instance prediction  target")  
  
    for t in range(test.shape[0]):  
        for k in range (test.shape[1]-1):  
            count1=count0=0  
            for j in range (data.shape[0]):  
                #how many times appeared with no  
                if test[t,k] == data[j,k] and data[j,data.shape[1]-1]=='No':  
                    count0+=1  
                #how many times appeared with yes  
                if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='Yes':  
                    count1+=1  
  
            prob0[k]=count0/countNo  
            prob1[k]=count1/countYes  
  
            probno=probNo  
            probyes=probYes  
            for i in range(test.shape[1]-1):  
                probno=probno*prob0[i]  
                probyes=probyes*prob1[i]  
            if probno>probyes:  
                predict='No'  
            else:  
                predict='Yes'  
  
            print(t+1," \t",predict," \t ",test[t,test.shape[1]-1])  
            if predict == test[t,test.shape[1]-1]:  
                accuracy+=1  
    final_accuracy=(accuracy/test.shape[0])*100  
    print("accuracy",final_accuracy,"%")  
    return
```

```
metadata,traindata= read_data("/content/1BM20CS066_NBC.csv")
splitRatio=0.6
trainingset, testset=splitDataset(traindata, splitRatio)
training=np.array(trainingset)
print("\n The Training data set are:")
for x in trainingset:
    print(x)

testing=np.array(testset)
print("\n The Test data set are:")
for x in testing:
    print(x)
classify(training,testing)
```

output:

```
The Training data set are:
['rainy', 'Yes']
['sunny', 'Yes']
['overcast', 'Yes']
['overcast', 'Yes']
['sunny', 'No']
['rainy', 'Yes']
['sunny', 'Yes']
['overcast', 'Yes']
```

```
The Test data set are:
['rainy' 'No']
['sunny' 'No']
['sunny' 'Yes']
['rainy' 'No']
['overcast' 'Yes']
['overcast' 'Yes']
```

```
training data size= 8
test data size= 6
```

target	count	probability
Yes	7	0.875
No	1	0.125

instance	prediction	target
1	Yes	No
2	Yes	No
3	Yes	Yes
4	Yes	No
5	Yes	Yes
6	Yes	Yes

accuracy 50.0 %

Naive Bayes

Training Dataset

Color	Type	Origin	Stolen	
Red	Sports	domestic	Yes	
Red	Sports	Domestic	No	Size = 6
Red	Sports	Domestic	Yes	
Yellow	Sports	Domestic	No	
Yellow	Sports	Imported	Yes	
Yellow	SUV	Imported	No	

Test Data Set:

Color	Type	Origin	Stolen
Yellow	SUV	imported	Yes
Yellow	SUV	Domestic	No
Red	SUV	Imported	No
Red	Sports	Imported	Yes

Target	Count	Probability
Yes	3	1/2
No	3	1/2

Instance	Prediction	Target
1	No	Yes
2	No	No
3	No	No
4	Yes	Yes

Accuracy : 75.0 %

$$P(h|D) = \frac{P(D|h) \cdot P(h)}{P(D)}$$

$P(h|D)$ = Posterior Probability

$P(h)$ = Prior Probability

$P(D)$ = Probability over data set

$P(D|h)$ = Current Probability

~~Off-line
125/23~~

Program 7:K- means clustering

Algorithm:

Initialize k means with random values

For a given number of iterations:

Iterate through items:

Find the mean closest to the item by calculating the euclidean distance of the item with each of the means

Assign item to mean

Update mean by shifting it to the average of the items in that cluster

Dataset:

Kmeans_1BM20CS066.csv X				...
1	Name	Age	Income(\$)	Filter
2	Rob	27	70000	
3	Michael	29	90000	
4	Mohan	29	61000	
5	Ismail	28	60000	
6	Kory	42	150000	
7	Gautam	39	155000	
8	David	41	160000	
9	Andrea	38	162000	
10	Brad	36	156000	
11	Angelina	35	130000	
12	Donald	37	137000	
13	Tom	26	45000	
14	Arnold	27	48000	
15	Jared	28	51000	
16	Stark	29	49500	
17	Ranbir	32	53000	
18	Dipika	40	65000	
19	Priyanka	41	63000	
20	Nick	43	64000	
21	Alia	39	80000	
22	Sid	41	82000	
21	Abdul	39	58000	

Show 25 per page

K-means Algorithm

- ① Select the number K to decide the number of clusters.
 - ② Select random K points or centroids.
 - ③ Assign each data point to their closest centroid which will form the predefined K cluster.
 - ④ Calculate the variance and new place centroid of each cluster.
 - ⑤ Repeat the third steps, which means re-sign each datapoint to new closest centroid
 - ⑥ If any re-assignment occurs, go to step 4 else FINISH
- ~~Q: How~~
⑦ Model is ready.
7/16/25

GMM - Gaussian Mixture model.

```
[1] import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
```

```
[2] df = pd.read_csv('/content/Kmeans_1BM20CS066.csv')
df.head(10)
```

	1	Name	Age	Income(\$)
0	2	Rob	27	70000
1	3	Michael	29	90000
2	4	Mohan	29	61000
3	5	Ismail	28	60000
4	6	Kory	42	150000
5	7	Gautam	39	155000
6	8	David	41	160000
7	9	Andrea	38	162000
8	10	Brad	36	156000
9	11	Angelina	35	130000

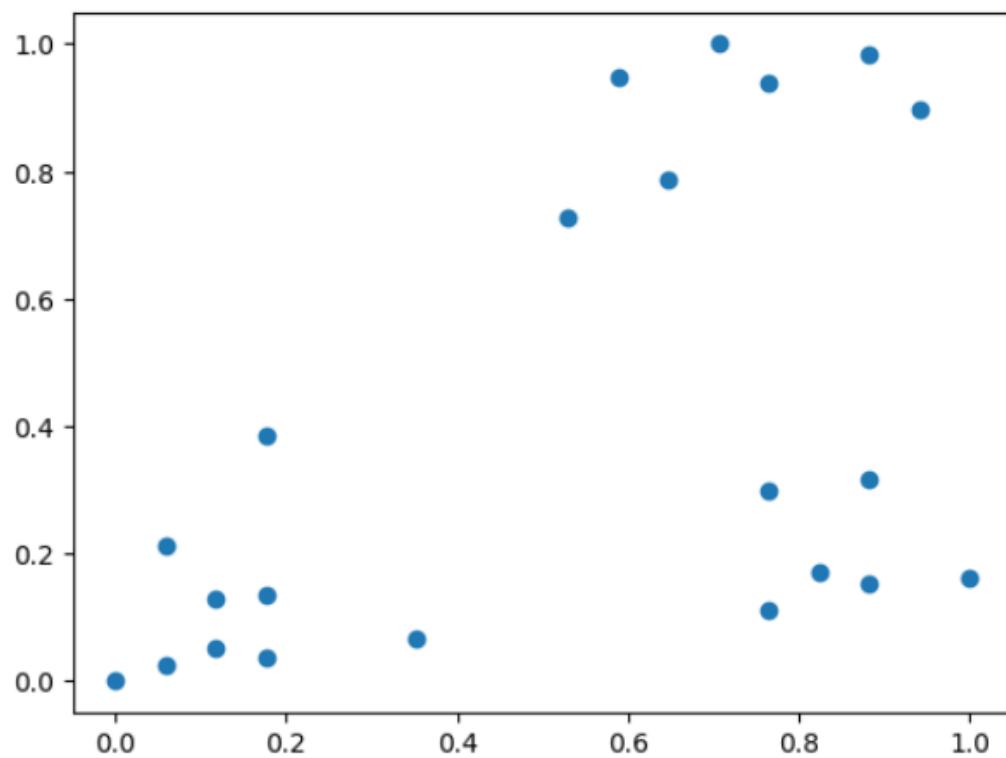
```
[3] scaler = MinMaxScaler()
scaler.fit(df[['Age']])
df[['Age']] = scaler.transform(df[['Age']])

scaler.fit(df[['Income($)']])
df[['Income($)']] = scaler.transform(df[['Income($)']])
df.head(10)
```

	1	Name	Age	Income(\$)
0	2	Rob	0.058824	0.213675
1	3	Michael	0.176471	0.384615
2	4	Mohan	0.176471	0.136752
3	5	Ismail	0.117647	0.128205
4	6	Kory	0.941176	0.897436
5	7	Gautam	0.764706	0.940171
6	8	David	0.882353	0.982906
7	9	Andrea	0.705882	1.000000
8	10	Brad	0.588235	0.948718
9	11	Angelina	0.529412	0.726496

```
▶ plt.scatter(df['Age'], df['Income($)'])
```

```
◀ <matplotlib.collections.PathCollection at 0x7f43820d1a50>
```



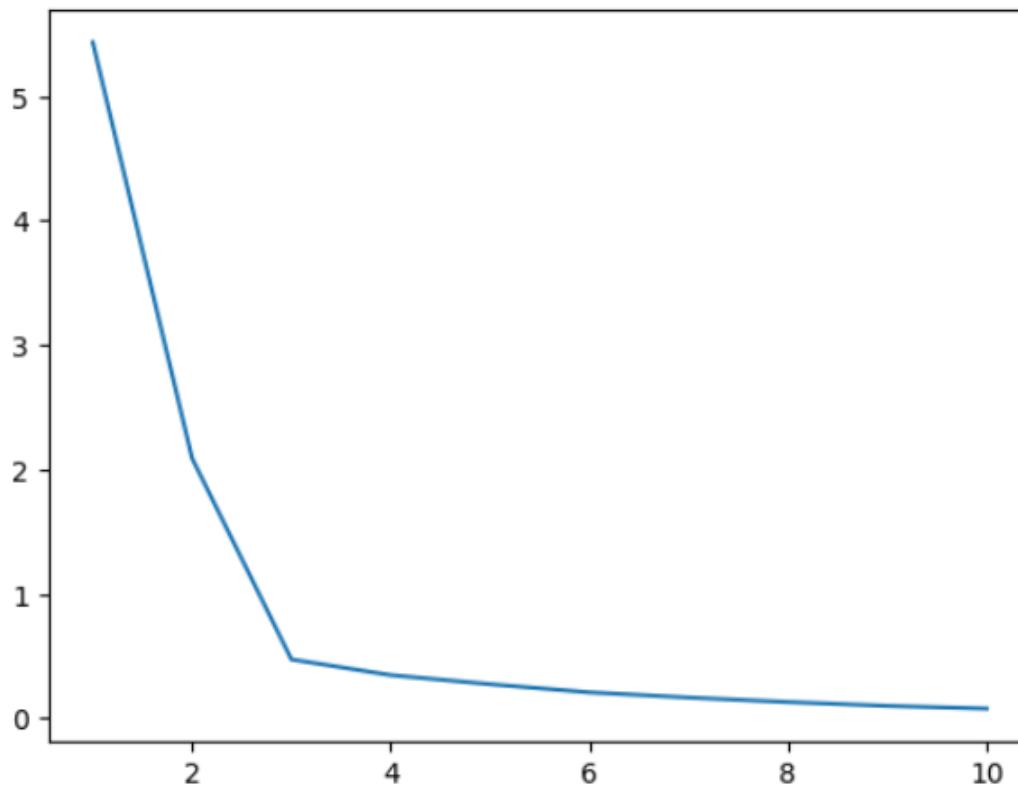
```
▶ k_range = range(1, 11)
```

```
sse = []
for k in k_range:
    kmc = KMeans(n_clusters=k)
    kmc.fit(df[['Age', 'Income($)']])
    sse.append(kmc.inertia_)
sse
```

```
.....
[5.434011511988178,
 2.091136388699078,
 0.4750783498553096,
 0.3491047094419566,
 0.2798062931046179,
 0.2203764169077067,
 0.1685851223602976,
 0.13265419827245162,
 0.1038375258660356,
 0.08510915216361345]
```

```
plt.xlabel = 'Number of Clusters'  
plt.ylabel = 'Sum of Squared Errors'  
plt.plot(k_range, sse)
```

```
[8] [ <matplotlib.lines.Line2D at 0x7f438004a6e0>]
```



```
[8] km = KMeans(n_clusters=3)  
km
```

```
▼ KMeans  
KMeans(n_clusters=3)
```

```
y_predict = km.fit_predict(df[['Age', 'Income($)']])  
y_predict  
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of  
warnings.warn(  
array([1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2],  
      dtype=int32)
```

```
[10] df['cluster'] = y_predict  
df.head()
```

	1	Name	Age	Income(\$)	cluster
0	2	Rob	0.058824	0.213675	1
1	3	Michael	0.176471	0.384615	1
2	4	Mohan	0.176471	0.136752	1
3	5	Ismail	0.117647	0.128205	1
4	6	Kory	0.941176	0.897436	0

```
[11] df0 = df[df.cluster == 0]  
df0
```

	1	Name	Age	Income(\$)	cluster
4	6	Kory	0.941176	0.897436	0
5	7	Gautam	0.764706	0.940171	0
6	8	David	0.882353	0.982906	0
7	9	Andrea	0.705882	1.000000	0
8	10	Brad	0.588235	0.948718	0
9	11	Angelina	0.529412	0.726496	0

```
✓ [12] df1 = df[df.cluster == 1]
      df1
```

	1	Name	Age	Income(\$)	cluster
0	2	Rob	0.058824	0.213675	1
1	3	Michael	0.176471	0.384615	1
2	4	Mohan	0.176471	0.136752	1
3	5	Ismail	0.117647	0.128205	1
11	13	Tom	0.000000	0.000000	1
12	14	Arnold	0.058824	0.025641	1
13	15	Jared	0.117647	0.051282	1
14	16	Stark	0.176471	0.038462	1
15	17	Ranbir	0.352941	0.068376	1

```
✓ [13] df2 = df[df.cluster == 2]
      df2
```

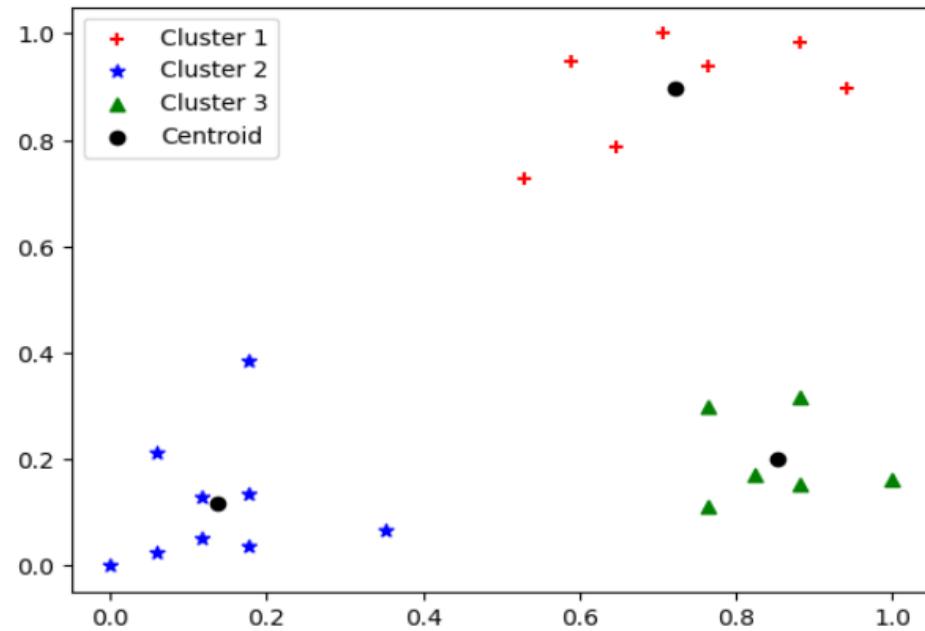
	1	Name	Age	Income(\$)	cluster
16	18	Dipika	0.823529	0.170940	2
17	19	Priyanka	0.882353	0.153846	2
18	20	Nick	1.000000	0.162393	2
19	21	Alia	0.764706	0.299145	2
20	22	Sid	0.882353	0.316239	2
21	21	Abdul	0.764706	0.111111	2

```
✓ [14] km.cluster_centers_
```

```
array([[0.72268908, 0.8974359 ],
       [0.1372549 , 0.11633428],
       [0.85294118, 0.2022792 ]])
```

```
[17] p1 = plt.scatter(df0['Age'], df0['Income($)'), marker='+', color='red')
    p2 = plt.scatter(df1['Age'], df1['Income($)'), marker='*', color='blue')
    p3 = plt.scatter(df2['Age'], df2['Income($)'), marker='^', color='green')
    c = plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color='black')
    plt.legend((p1, p2, p3, c),
               ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))
```

<matplotlib.legend.Legend at 0x7f437d4c73a0>

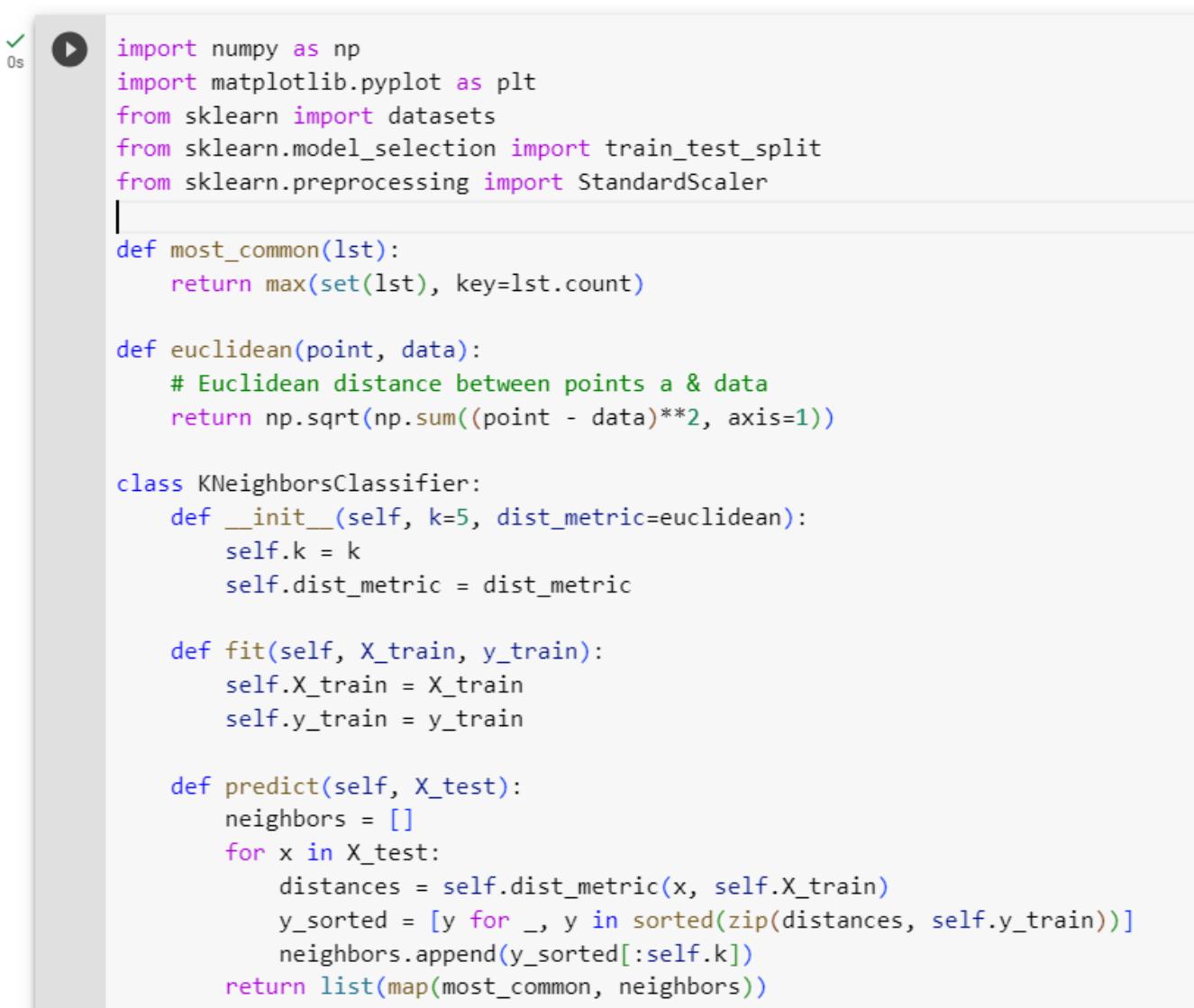


Program 8: KNN ALGORITHM

Dataset used: Iris dataset

Algorithm:

- Select the number K of the neighbor
- Calculate the Euclidean distance of K number of neighbors
- Take the K nearest neighbors as per the calculated Euclidean distance.
- Among these k neighbors, count the number of the data points in each category.
- Assign the new data points to that category for which the number of the neighbor is maximum.



The screenshot shows a Jupyter Notebook cell with the following code:

```
✓ 0s
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
|
def most_common(lst):
    return max(set(lst), key=lst.count)

def euclidean(point, data):
    # Euclidean distance between points a & data
    return np.sqrt(np.sum((point - data)**2, axis=1))

class KNeighborsClassifier:
    def __init__(self, k=5, dist_metric=euclidean):
        self.k = k
        self.dist_metric = dist_metric

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        neighbors = []
        for x in X_test:
            distances = self.dist_metric(x, self.X_train)
            y_sorted = [y for _, y in sorted(zip(distances, self.y_train))]
            neighbors.append(y_sorted[:self.k])
        return list(map(most_common, neighbors))
```

```
✓ 0s
def evaluate(self, X_test, y_test):
    y_pred = self.predict(X_test)
    accuracy = sum(y_pred == y_test) / len(y_test)
    return accuracy

iris = datasets.load_iris()
X = iris['data']
y = iris['target']

# Split data into train & test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Preprocess data
ss = StandardScaler().fit(X_train)
X_train, X_test = ss.transform(X_train), ss.transform(X_test)

# Test knn model across varying ks
accuracies = []
ks = range(1, 30)
for k in ks:
    knn = KNeighborsClassifier(k=k)
    knn.fit(X_train, y_train)
    accuracy = knn.evaluate(X_test, y_test)
    accuracies.append(accuracy)
# Visualize accuracy vs. k
fig, ax = plt.subplots()
ax.plot(ks, accuracies)
ax.set(xlabel="k",
       ylabel="Accuracy",
       title="Performance of knn")
plt.show()
```

K-nearest Neighbor Algorithm

- * For each given training example $(x, f(x))$, add the example to the list training examples to the list training examples classification algorithm.
- * Given a query instance x_q to be classified,
let x_1, \dots, x_n denote the K instances from training examples that are nearest to x_q .

* Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^n f(x_i)}{K}$$

Separate

Output.

Sepal-length sepal-width petal-length petal-width

$$\begin{bmatrix} [5.1 & 3.5 & 1.4 & 0.2] \\ [4.9 & 3 & 1.4 & 0.2] \\ [4.7 & 3.2 & 1.3 & 0.2] \\ [4.6 & 3.1 & 1.5 & 0.2] \\ [5.0 & 3.6 & 1.4 & 0.2] \end{bmatrix}$$

⋮ ⋮ ⋮ ⋮ ⋮

$$\begin{bmatrix} [6.2 & 3.4 & 5.4 & 2.3] \\ [5.9 & 3 & 5.1 & 1.8] \end{bmatrix}$$

Class : 0 - Iris-setosa, 1 - Iris-Versicolor, 2 - Iris-

Virginica

$$[000 \dots 0011 \dots 11222 \dots 22]$$

Confusion Matrix

$$\begin{bmatrix} 20 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 1 & 14 \end{bmatrix}$$

Accuracy Metrics

Precision Recall f1-score support

0	1.00	1.00	1.00	20
1	0.91	1.00	0.95	20
2	1.00	0.93	0.97	15

Avg/total 0.98 0.98 0.98 45

✓ Ophelia

7/6/23

Program 9: Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

Algorithm for k means clustering:

- Initialize k means with random values
- For a given number of iterations:
- Iterate through items:
- Find the mean closest to the item by calculating the euclidean distance of the item with each of the means
- Assign item to mean
- Update mean by shifting it to the average of the items in that clusters

Algorithm for EM algorithm:

- The very first step is to initialize the parameter values. Further, the system is provided with incomplete observed data with the assumption that data is obtained from a specific model.
- This step is known as Expectation or E-Step, which is used to estimate or guess the values of the missing or incomplete data using the observed data. Further, E-step primarily updates the variables.
- This step is known as Maximization or M-step, where we use complete data obtained from the 2nd step to update the parameter values. Further, M-step primarily updates the hypothesis.
- The last step is to check if the values of latent variables are converging or not.

Dataset: Iris dataset



```
2s  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

```

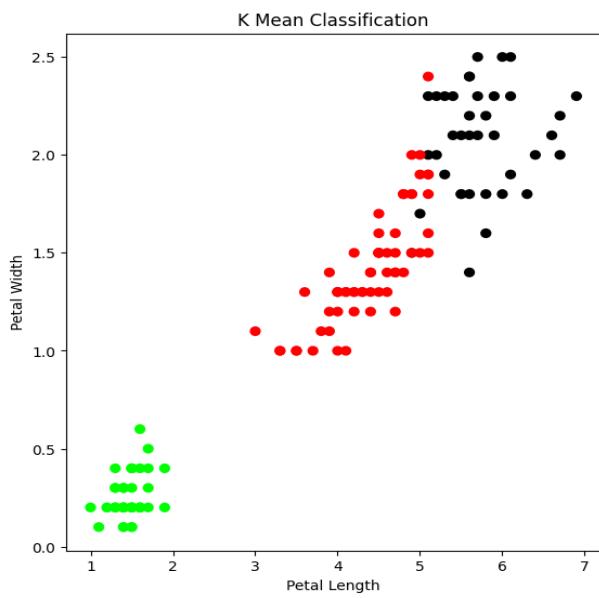
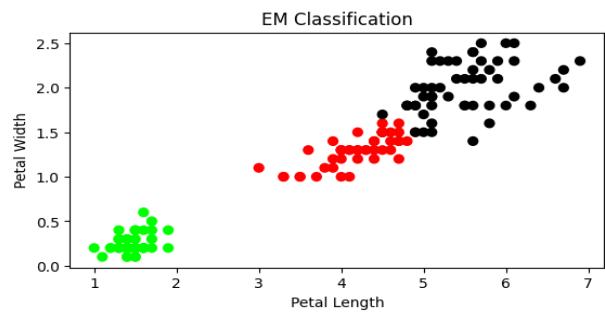
```

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

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

```

The accuracy score of K-Mean: 0.24
The Confusion matrixof K-Mean: [[0 50 0]
[48 0 2]
[14 0 36]]
The accuracy score of EM: 0.3333333333333333
The Confusion matrix of EM: [[0 50 0]
[45 0 5]
[0 0 50]]



EM-Algorithm

- * Expectation Step (E step): It involves the estimation of all missing values in dataset so that after completing this step, there should not be any missing value.
 - * Maximize step (M-step): This step involves the use of estimated data in E-step and updating the parameter.
 - * Repeat E step and M step until the convergence of value occurs.
- ① Initialize Parameter Values. Further, the system is provided with incomplete observed data with assumption that data is obtained from specific model.
 - ② E-step, which is used to estimate or guess the value of the missing data using the observed data.
 - ③ Maximization step, where we use the complete data obtained from 2nd step to update Parameter values.
- * The last step is to check if value of variables are converging or not.
 - * If yes, stop process else repeat until convergence occurs.

Euclidean distance formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

x_1 = x co-ordinate of point 1

y_1 = y co-ordinate of point 1

x_2 = x co-ordinate of pt 2

y_2 = y co-ordinate of pt 2

✓ Only one
way

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.

Algorithm:

1. F is approximated near X_q using a linear function:

$$\hat{f}(x) = w_0 + \sum_{u=1}^k w_u K_u(d(x_u, x))$$

2. Minimize the squared error:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) (f(x) - \hat{f}(x)) a_j(x)$$

3. It is weighted because the contribution of each training example is weighted by its distance from the query point.

Dataset: tip.csv

```

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

[ ] def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

[ ] def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

```

```
def localWeightRegression(xmat, ymat, k):
    m, n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i], xmat, ymat, k)
    return ypred
```

```
[ ] def graphPlot(X, ypred):
    sortindex = X[:,1].argsort(0)
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill, tip, color='green')
    ax.plot(xsort[:,1], ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();
```

```
data = pd.read_csv('/content/tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

mbill = np.mat(bill)
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

# increase k to get smooth curves
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```

Lab-7 Locally weighted Regression Algorithm

- 1) Read the given data sample to x and the curve (linear or non linear) to y .
- 2) Set the value of Smoothing Parameter or Free Parameter say t .
- 3) Set the bias / Point of interest set x_0 which is subset of x .
- 4) Determine the weight matrix using:
$$W(w_i, w_j) = e^{\frac{-(x_i - x_j)^2}{2t^2}}$$
- 5) Determine the value of model term parameter β using:

$$\hat{\beta}(x_0) = (X^T W X)^{-1} X^T W y$$

- 6) Prediction = $x_0 \cdot \hat{\beta}$.

