VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

On

MACHINE LEARNING

Submitted by

Mohammad Adnan Khan (1BM21CS107)

in partial fulfilment for the award of the degree of
BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)

BENGALURU-560019

March 2024 to June 2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum) Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled **"MACHINE LEARNING"** carried out by **Mohammad Adnan Khan (1BM21CS107)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Machine Learning Lab - **(22CS6PCMAL)** work prescribed for the said degree.

Sonika S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index

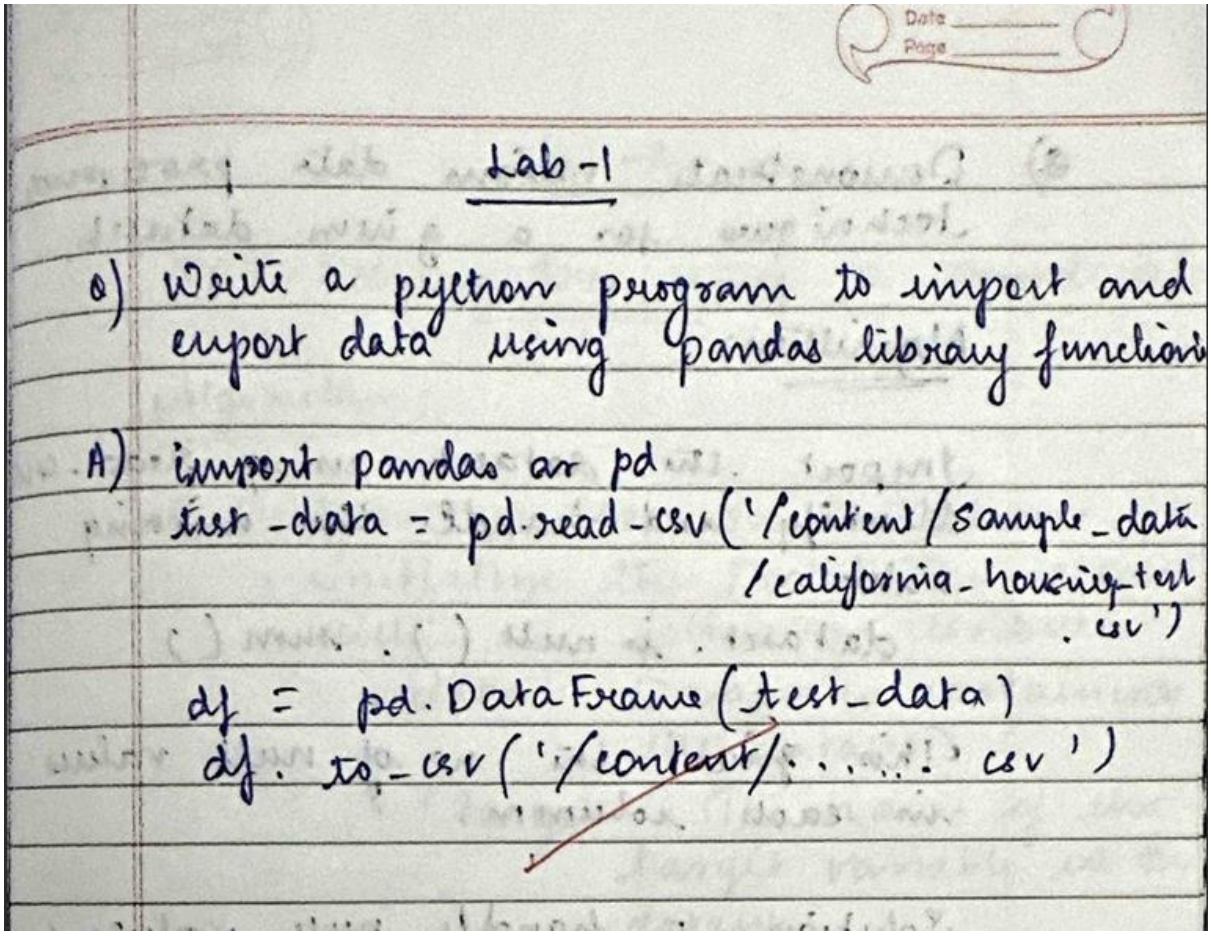
Sl. No.	Experiment Title	Page No.
1	Write a python program to import and export data using Pandas library functions	1
2	Demonstrate various data pre-processing techniques for a given dataset	4
3	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	6
4	Build KNN Classification model for a given dataset.	13
5	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	20
6	Build Logistic Regression Model for a given dataset	27
7	Build Support vector machine model for a given dataset	37
8	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	43
9	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	47
10	Build Artificial Neural Network model with back propagation on a given dataset	51
11	a) Implement Random Forest ensemble method on a given dataset. b) Implement Boosting ensemble method on a given dataset.	54

Course outcomes:

CO1	Apply machine learning techniques in computing systems
CO2	Evaluate the model using metrics
CO3	Design a model using machine learning to solve a problem
CO4	Conduct experiments to solve real-world problems using appropriate machine learning techniques

5.04.2024 1BM21CS107

Write a python program to import and export data using Pandas library functions



Lab-1

Q) Write a python program to import and export data using pandas library functions

A) import pandas as pd
test_data = pd.read_csv('/content/sample_data/california_housing_test.csv')
df = pd.DataFrame(test_data)
df.to_csv('/content/.....csv')

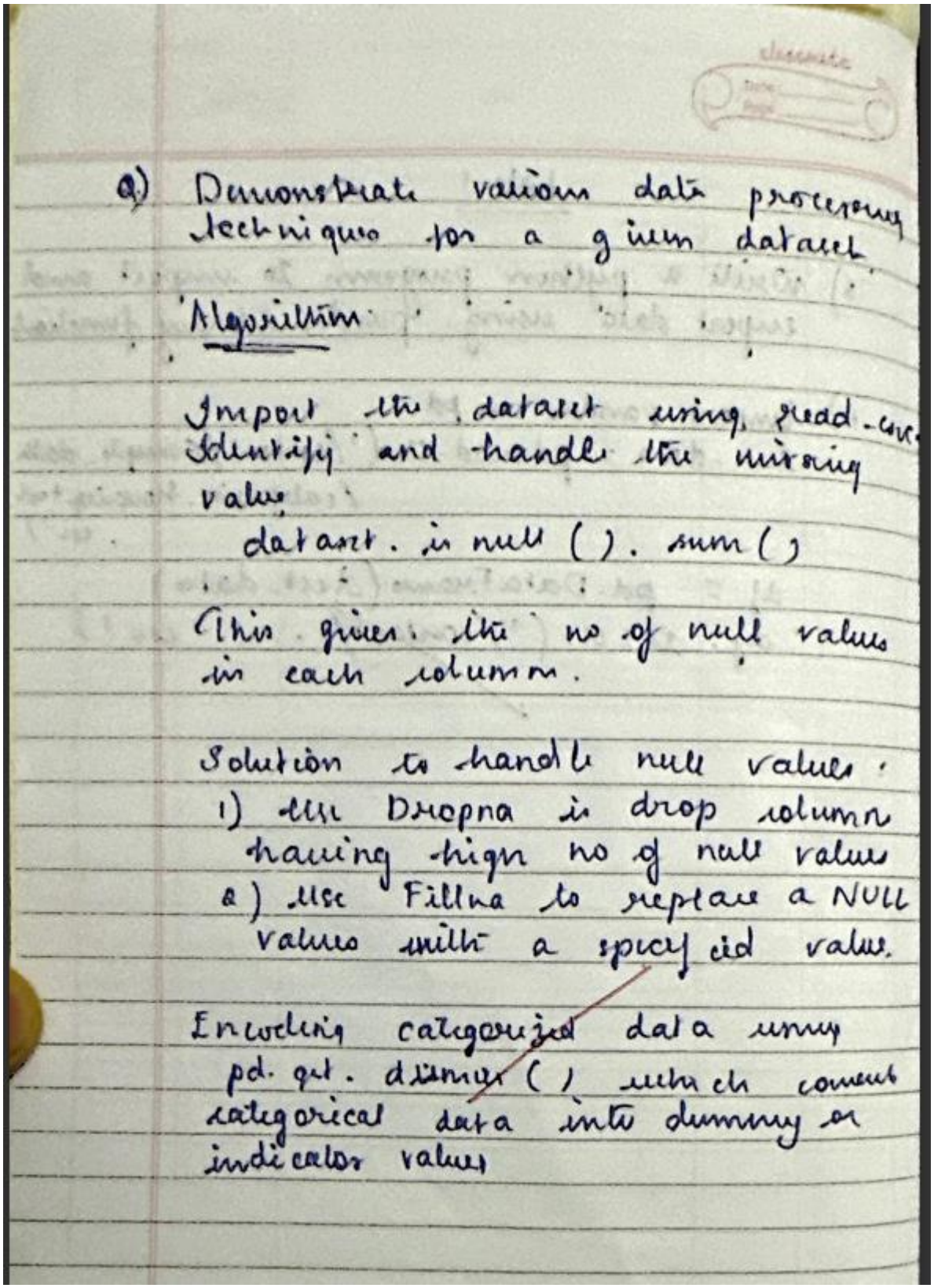
```
import pandas as pd
df=pd.read_csv("austinHousingData.csv")
print(df.head())
df.to_csv("exported_dataset.csv")
```

Output:

```
        zpid         city         streetAddress  zipcode  ... numOfBathrooms  numOfBedrooms  numOfStories                                  homeImage
0  111373431  pflugerville   14424 Lake Victor Dr    78660  ...            3.0              4             2  111373431_ffce26843283d3365c11d81b8e6bdc6f-p_f...
1  120900430  pflugerville     1104 Strickling Dr    78660  ...            2.0              4             1  120900430_8255c127be8dcf0a1a18b7563d987088-p_f...
2 2084491383  pflugerville    1408 Fort Dessau Rd    78660  ...            2.0              3             1  2084491383_a2ad649e1a7a098111dcea084a11c855-p_...
3  120901374  pflugerville     1025 Strickling Dr    78660  ...            2.0              3             1  120901374_b469367a619da85b1f5ceb69b675d88e-p_f...
4   60134862  pflugerville  15005 Donna Jane Loop    78660  ...            3.0              3             2  60134862_b1a48a3df3f111e005bb913873e98ce2-p_f.jpg

[5 rows x 47 columns]
```

2. Demonstrate various data pre-processing techniques for a given dataset



Q) Demonstrate various data processing techniques for a given dataset.

Algorithm

Import the dataset using read_csv. Identify and handle the missing value.

dataset.isnull().sum()

This gives the no of null values in each column.

Solution to handle null values:

1) Use Dropna to drop column having high no of null values
2) Use Fillna to replace a NULL values with a specified value.

Encoding categorized data using pd.get_dummies() which convert categorical data into dummy or indicator values

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
```

[1] Python

```
dataset = pd.read_csv("/content/Data.csv")
```

[2] Python

```
df = pd.DataFrame(dataset)
```

[3] Python

```
df
```

[4] Python

Outputs are collapsed …

```
X = df.iloc[:, :-1].values  Add to CodeGPT Chat (Ctrl + Shift + E).  Add to chat (Ctrl+L) | Edit highlighted code (Ctrl+I).
y = df.iloc[:, -1].values
```

[5] Python

```
print(X)
```

[6] Python

Outputs are collapsed …

```
print(y)
```

[7] Python

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

```
df.isnull().sum()
```

[8] Python

Outputs are collapsed …

Dropna

```
df1 = df.copy()
```

[30] Python

```
# summarize the shape of the raw data
print("Before:",df1.shape)

# drop rows with missing values
df1.dropna(inplace=True)

# summarize the shape of the data with missing rows removed
print("After:",df1.shape)
```

[11] Python

Outputs are collapsed …

```
# summarize the shape of the raw data
print("Before:",df1.shape)

# drop rows with missing values
df1.dropna(inplace=True)

# summarize the shape of the data with missing rows removed
print("After:",df1.shape)
```

[13] Python

Outputs are collapsed …

Scikit-Learn

```
X
```

[33] Python

Outputs are collapsed …

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

[33] Python

```
print(X)
```

[34] Python

Outputs are collapsed …

Using Dummies

```
df2
```

[35] Python

Outputs are collapsed …

```
pd.get_dummies(df2)  Add to CodeGPT Chat (Ctrl + Shift + E).  Add to chat (Ctrl+L) | Edit highlighted code (Ctrl+I).
```

[36] Python

Outputs are collapsed …

Output:

```
df
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

```
X = df.iloc[:, :-1].values  Add to CodeGPT Chat (Ctrl + Shift + E).  Add to chat (Ctrl+L) | Edit highlighted code (Ctrl+I).
y = df.iloc[:, -1].values
```

```
print(X)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

```
print(y)
```

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

```
df.isnull().sum()
```

```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

```
# summarize the shape of the raw data
print("Before:",df1.shape)

# drop rows with missing values
df1.dropna(inplace=True)

# summarize the shape of the data with missing rows removed
print("After:",df1.shape)
```

```
Before: (10, 4)
After: (8, 4)
```

Scikit-Learn

```
X
```

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

```
print(X)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

Using Dummies

```
df2
```

Python

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

```
pd.get_dummies(df2)  Add to CodeSPT Chat (Ctrl + Shift + E).        Add to chat (Ctrl+L) | Edit highlighted code (Ctrl+I).
```

Python

	Age	Salary	Country_France	Country_Germany	Country_Spain	Purchased_No	Purchased_Yes
0	44.0	72000.0	True	False	False	True	False
1	27.0	48000.0	False	False	True	False	True
2	30.0	54000.0	False	True	False	True	False
3	38.0	61000.0	False	False	True	True	False
4	40.0	NaN	False	True	False	False	True
5	35.0	58000.0	True	False	False	False	True
6	NaN	52000.0	False	False	True	True	False
7	48.0	79000.0	True	False	False	False	True
8	50.0	83000.0	False	True	False	True	False
9	37.0	67000.0	True	False	False	False	True

12/04/2024

Decision Tree Algorithm:

classmate
Date
Page

Lab - 2

ID3 Decision tree using a sample set

Algorithm:

1. Decision Tree Class Initialization:
   - Initializing the DecisionTree class with the following attributes
     - 'data' : Dataframe containing the dataset
     - 'target' : The name of the target variable in the dataset
     - 'positive' : Value representing the positive class in the target variable
     - 'parent_val' : The value of the parent node in the decision tree
     - 'parent' : The name of the parent node
     - 'childs' : List to store child nodes
     - 'decision' : Empty string to store decision made at node

2. _get_entropy Method :

- Calculate the entropy of a given dataset based on the target variable
- Formula - p, ratio * $\log_2$ (p, ratio) - n, ratio * $\log_2$ (n, ratio)

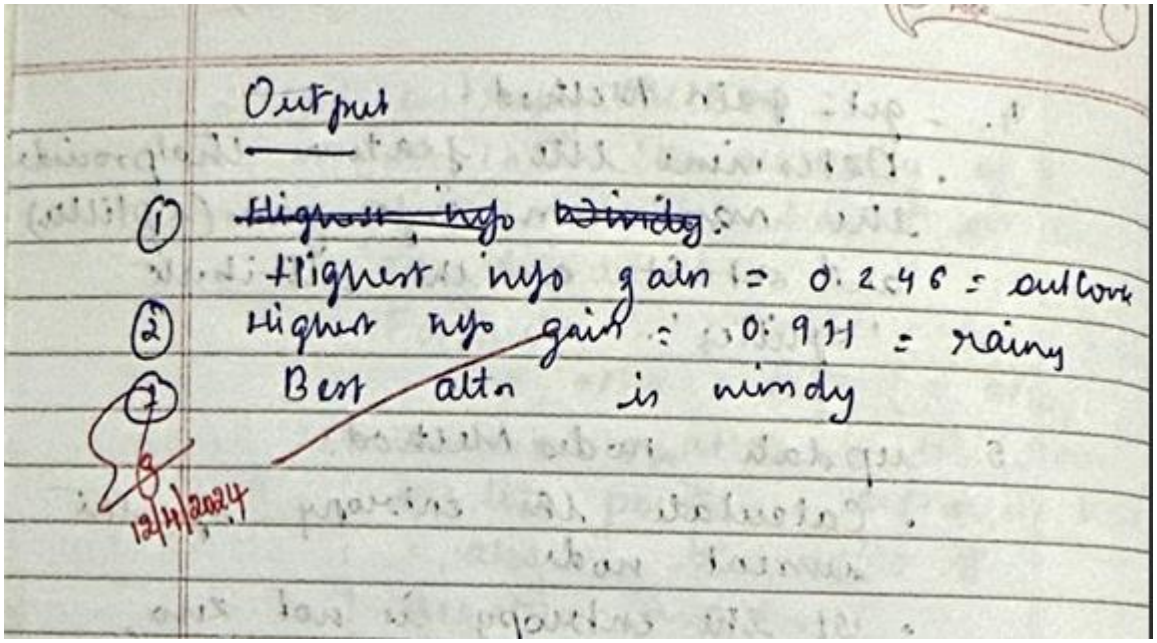
where p, ratio is the ratio of the positive instances and n, ratio is the ratio of negative instance

3. _get_gain Method :

- Calculate the information gain for a given feature
- Iterate through unique values of the feature and calculate the weighted average of entropy for each value
- Formula : Entropy (D) - $\sum_{v \in Values(feat}$ $\frac{|D_v|}{|D|}$ * Entropy (D_v

4. _get_gain Method:
   - Determine the feature that provides the maximum info gain (splits) and set it as the attribute 'splitter'.

5. update nodes Method:
   - Calculate the entropy of the current node
   - If the entropy is not zero, calculate the info gain for each feature.
   - Determine the feature that provides the maximum info gain (splitter)
   - Iterate through unique values of the splitter feature and child nodes
   - Recursively call the 'update_nodes' method for each child node.



+ Code + Markdown

```
# Importing the required libraries
import pandas as pd
import numpy as np
import math

# Reading the dataset (Tennis-dataset)
data = pd.read_csv('/content/PlayTennis.csv')
```

[1] Python

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

Python

```
def highlight(cell_value):
    '''
    Highlight yes / no values in the dataframe
    '''
    color_1 = 'background-color: pink;'
    color_2 = 'background-color: lightgreen;'

    if cell_value == 'no':
        return color_1
    elif cell_value == 'yes':
        return color_2

data.style.applymap(highlight)\
    .set_properties(subset=data.columns, **{'width': '100px'})\
    .set_table_styles([{'selector': 'th', 'props': [('background-color', 'lightgray'), ('border', '1px solid gray'),
                                                    ('font-weight', 'bold')]},
    {'selector': 'tr:hover', 'props': [('background-color', 'white'), ('border', '1.5px solid black')]}])
```

[3] Python

Outputs are collapsed ...

```python
def find_entropy(data):
    """
    Returns the entropy of the class or features
    formula: - Σ P(X)logP(X)
    """
    entropy = 0
    for i in range(data.nunique()):
        x = data.value_counts()[i]/data.shape[0]
        entropy += (- x * math.log(x,2))
    return round(entropy,3)


def information_gain(data, data_):
    """
    Returns the information gain of the features
    """
    info = 0
    for i in range(data_.nunique()):
        df = data[data_ == data_.unique()[i]]
        w_avg = df.shape[0]/data.shape[0]
        entropy = find_entropy(df.play)
        x = w_avg * entropy
        info += x
    ig = find_entropy(data.play) - info
    return round(ig, 3)


def entropy_and_infogain(datax, feature):
    """
    Grouping features with the same class and computing their
    entropy and information gain for splitting
    """
    for i in range(data[feature].nunique()):
        df = datax[datax[feature]==data[feature].unique()[i]]
        if df.shape[0] < 1:
            continue

        display(df[[feature, 'play']].style.applymap(highlight)\
                .set_properties(subset=[feature, 'play'], **{'width': '80px'})\
                .set_table_styles([{'selector': 'th', 'props': [('background-color', 'lightgray'),
                                                                ('border', '1px solid gray'),
                                                                ('font-weight', 'bold')]},
                                   {'selector': 'td', 'props': [('border', '1px solid gray')]},
                                   {'selector': 'tr:hover', 'props': [('background-color', 'white'),
                                                                      ('border', '1.5px solid black')]}]))

        print(f'Entropy of {feature} - {data[feature].unique()[i]} = {find_entropy(df.play)}')
    print(f'Information Gain for {feature} = {information_gain(datax, datax[feature])}')
```

[6] Python

```python
print(f'Entropy of the entire dataset: {find_entropy(data.play)}')
```

[9] Python

```
Entropy of the entire dataset: 0.94
```

```python
entropy_and_infogain(data, 'temp')
```

[6] Python

Outputs are collapsed …

```python
entropy_and_infogain(data, 'humidity')
```

[7] Python

Outputs are collapsed …

```python
entropy_and_infogain(data, 'windy')
```

[8] Python

Outputs are collapsed …

#Rainy -outlook

```python
rainy = data[data['outlook'] == 'rainy']
rainy.style.applymap(highlight)\
    .set_properties(subset=data.columns, **{'width': '100px'})\
    .set_table_styles([{'selector': 'th', 'props': [('background-color', 'lightgray'), ('border', '1px solid gray'),
                                                    ('font-weight', 'bold')]},
     {'selector': 'tr:hover', 'props': [('background-color', 'white'), ('border', '1.5px solid black')]}])
```

[9] Python

Outputs are collapsed …

```python
print(f'Entropy of the Rainy dataset: {find_entropy(rainy.play)}')
```

[10] Python

```
Entropy of the Rainy dataset: 0.971
```

```python
entropy_and_infogain(rainy, 'temp')
```

[11] Python

Outputs are collapsed …

```python
entropy_and_infogain(rainy, 'humidity')
```

[12] Python

```python
entropy_and_infogain(rainy, 'temp')
```

[11] Python

Outputs are collapsed …

```python
entropy_and_infogain(rainy, 'humidity')
```

[12] Python

Outputs are collapsed …

```python
entropy_and_infogain(rainy, 'windy')
```

[13] Python

Outputs are collapsed …

wind has highest information gain

Output:

```
def highlight(cell_value):
    '''
    Highlight yes / no values in the dataframe
    '''
    color_1 = 'background-color: pink;'
    color_2 = 'background-color: lightgreen;'

    if cell_value == 'no':
        return color_1
    elif cell_value == 'yes':
        return color_2

data.style.applymap(highlight)\
    .set_properties(subset=data.columns, **{'width': '100px'})\
    .set_table_styles([{'selector': 'th', 'props': [('background-color', 'lightgray'), ('border', '1px solid gray'),
                                                    ('font-weight', 'bold')]},
    {'selector': 'tr:hover', 'props': [('background-color', 'white'), ('border', '1.5px solid black')]}])
```

[3] Python

outlook	temp	humidity	windy	play
sunny	hot	high	False	no
sunny	hot	high	True	no
overcast	hot	high	False	yes
rainy	mild	high	False	yes
rainy	cool	normal	False	yes
rainy	cool	normal	True	no
overcast	cool	normal	True	yes
sunny	mild	high	False	no
sunny	cool	normal	False	yes
rainy	mild	normal	False	yes
sunny	mild	normal	True	yes
overcast	mild	high	True	yes
overcast	hot	normal	False	yes
rainy	mild	high	True	no

```
print(f'Entropy of the entire dataset: {find_entropy(data.play)}')
```

[5] Python

```
Entropy of the entire dataset: 0.94
```

```
entropy_and_infogain(data, 'temp')
```

[6] Python

temp	play
hot	no
hot	no
hot	yes
hot	yes

```
Entropy of temp - hot = 1.0
```

temp	play
mild	yes
mild	no
mild	yes
mild	yes
mild	yes
mild	no

```
Entropy of temp - mild = 0.918
```

temp	play
cool	yes
cool	no
cool	yes
cool	yes

```
Entropy of temp - cool = 0.811
Information Gain for temp = 0.029
```

```
entropy_and_infogain(data, 'humidity')
```

[7] Python

humidity	play
high	no
high	no
high	yes
high	yes
high	no
high	yes
high	no

```
Entropy of humidity - high = 0.985
```

humidity	play
normal	yes
normal	no
normal	yes
normal	yes
normal	yes
normal	yes
normal	yes

```
Entropy of humidity - normal = 0.592
Information Gain for humidity = 0.151
```

```
entropy_and_infogain(data, 'windy')
```

Python

windy play

0	False	no
1	False	yes
2	False	yes
3	False	yes
4	False	no
5	False	yes
6	False	yes
7	False	no

```
Entropy of windy - False = 0.811
```

windy play

8	True	no
9	True	no
10	True	yes
11	True	yes
12	True	yes
13	True	no

```
Entropy of windy - True = 1.0
Information Gain for windy = 0.048
```

#Rainy -outlook

```
rainy = data[data['outlook'] == 'rainy']
rainy.style.applymap(highlight)\
    .set_properties(subset=data.columns, **{'width': '100px'})\
    .set_table_styles([{'selector': 'th', 'props': [('background-color', 'lightgray'), ('border', '1px solid gray'),
                                                    ('font-weight', 'bold')]},
     {'selector': 'tr:hover', 'props': [('background-color', 'white'), ('border', '1.5px solid black')]}])
```

Python

	outlook	temp	humidity	windy	play
0	rainy	mild	high	False	yes
1	rainy	cool	normal	False	yes
2	rainy	cool	normal	True	no
3	rainy	mild	normal	False	yes
4	rainy	mild	high	True	no

```
print(f'Entropy of the Rainy dataset: {find_entropy(rainy.play)}')
```

Python

```
Entropy of the Rainy dataset: 0.971
```

```
entropy_and_infogain(rainy, 'temp')
```

Python

temp play

0	mild	yes
1	mild	yes
2	mild	no

```
Entropy of temp - mild = 0.918
```

temp play

3	cool	yes
4	cool	no

```
Entropy of temp - cool = 1.0
Information Gain for temp = 0.02
```

```
entropy_and_infogain(rainy, 'humidity')
```

Python

humidity play

0	high	yes
1	high	no

```
Entropy of humidity - high = 1.0
```

humidity play

2	normal	yes
3	normal	no
4	normal	yes

```
Entropy of humidity - normal = 0.918
Information Gain for humidity = 0.02
```

```
entropy_and_infogain(rainy, 'windy')
```

Python

windy play

0	False	yes
1	False	yes
2	False	yes

```
Entropy of windy - False = 0.0
```

windy play

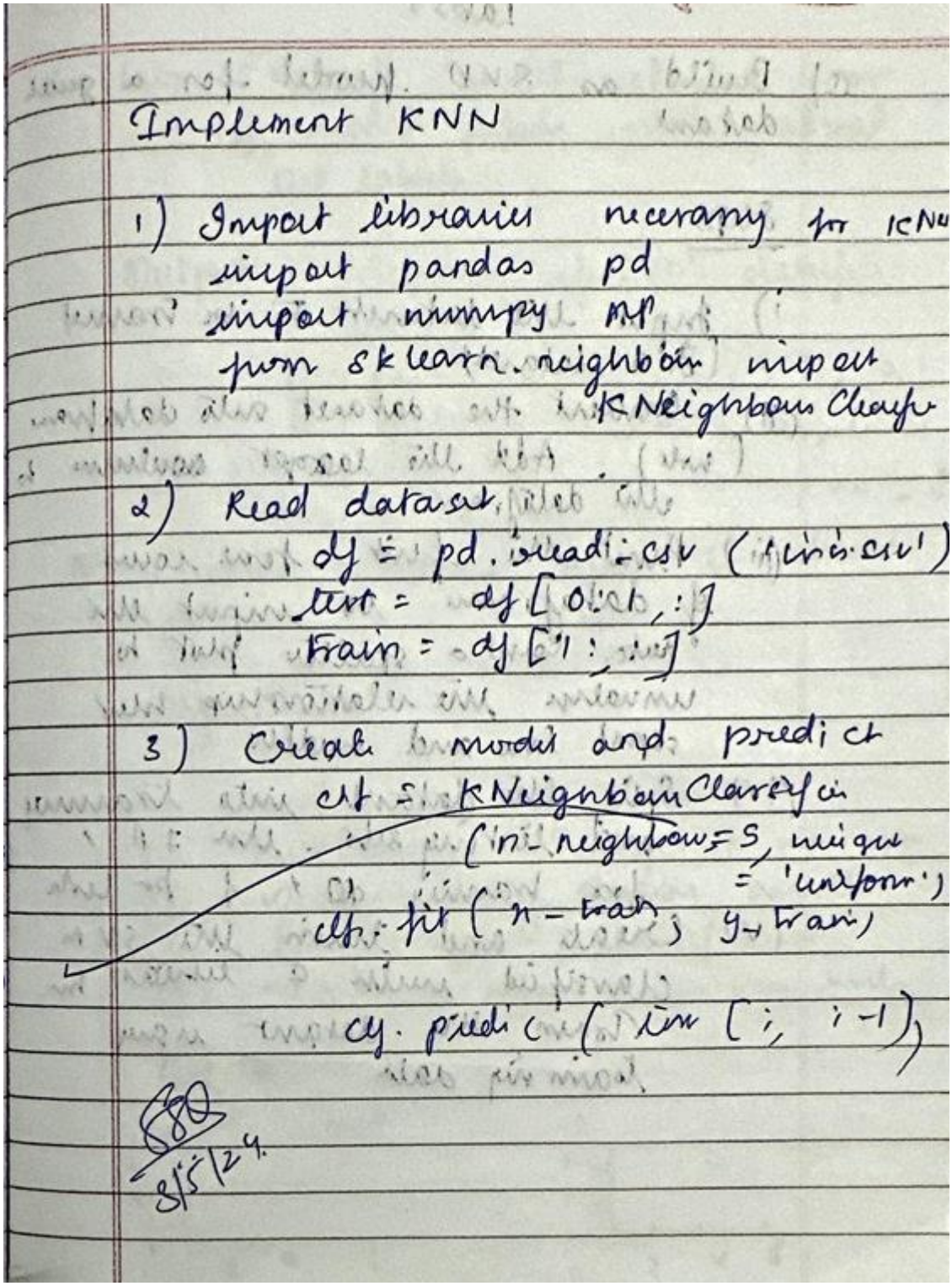
3	True	no
4	True	no

```
Entropy of windy - True = 0.0
Information Gain for windy = 0.971
```

wind has highest information gain

03/05/2024

Build KNN Classification model for a given dataset.:



Implement KNN

1) Import libraries necessary for KNN
import pandas pd
import numpy np
from sklearn.neighbors import KNeighborsClassifier

2) Read dataset
df = pd.read_csv('iris.csv')
test = df[0:1, :]
train = df['1':, :]

3) Create model and predict
clf = KNeighborsClassifier(n_neighbors=5, weights='uniform')
clf.fit(x_train, y_train)
clf.predict(iris[:, :-1])

8/5/24

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization purposes
import seaborn as sns # for data visualization
%matplotlib inline
```

[1] Python

```python
data = '/content/cancer_detector.txt'
df = pd.read_csv(data, header=None)
```

[2] Python

```python
df.shape
```

[3] Python

Outputs are collapsed ...

```python
col_names = ['Id', 'Clump_thickness', 'Uniformity_Cell_Size', 'Uniformity_Cell_Shape', 'Marginal_Adhesion',
             'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin', 'Normal_Nucleoli', 'Mitoses', 'Class']

df.columns = col_names

df.columns
```

[4] Python

Outputs are collapsed ...

```python
df.head()
```

[5] Python

Outputs are collapsed ...

```python
df.drop('Id', axis=1, inplace=True)
```

[6] Python

```python
# view summary of dataset

df.info()
```

[7] Python

Outputs are collapsed ...

```python
for var in df.columns:

    print(df[var].value_counts())
```

[8] Python

Outputs are collapsed ...

```python
df['Bare_Nuclei'] = pd.to_numeric(df['Bare_Nuclei'], errors='coerce')
```

[9] Python

```python
df.dtypes
```

[10] Python

Outputs are collapsed ...

```python
df.isnull().sum()
```

[11] Python

Outputs are collapsed ...

```python
# check `na` values in the dataframe

df.isna().sum()
```

[12] Python

Outputs are collapsed ...

```python
# check frequency distribution of `Bare_Nuclei` column

df['Bare_Nuclei'].value_counts()
```

[13] Python

Outputs are collapsed ...

```python
# check unique values in `Bare_Nuclei` column

df['Bare_Nuclei'].unique()
```

[14] Python

```
array([ 1., 10.,  2.,  4.,  3.,  9.,  7., nan,  5.,  8.,  6.])
```

```python
# check for nan values in `Bare_Nuclei` column ...
```

```
16
```

```python
# view frequency distribution of values in `Class` variable

df['Class'].value_counts()
```

Outputs are collapsed ...

```python
import numpy as np
```

```python
# view summary statistics in numerical variables

print(round(df.describe(),2))
```

Outputs are collapsed ...

```python
X = df.drop(['Class'], axis=1)

y = df['Class']
```

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```python
X_train.shape, X_test.shape
```

Outputs are collapsed ...

```python
for col in X_train.columns:
    if X_train[col].isnull().mean()>0:
        print(col, round(X_train[col].isnull().mean(),4))
```

```
Bare_Nuclei 0.0233
```

...

```python
cols = X_train.columns
```

```python
from sklearn.preprocessing import StandardScaler ...
```

```python
X_train = pd.DataFrame(X_train, columns=[cols])
```

```python
X_test = pd.DataFrame(X_test, columns=[cols])
```

```python
X_train.head() ...
```

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli	Mitoses
0	2.028383	0.299506	0.289573	1.119077	-0.546543	1.058057	-0.577774	0.041241	-0.324258
1	1.669451	2.257680	2.304569	-0.622471	3.106879	1.297589	-0.159953	0.041241	-0.324258
2	-1.202005	-0.679581	-0.717925	0.074148	-1.003220	-0.104329	-0.995595	-0.608165	-0.324258
3	-0.125209	-0.026856	-0.046760	-0.622471	-0.546543	-0.665096	-0.159953	0.041241	-0.324258
4	0.233723	-0.353219	-0.382092	-0.274161	-0.546543	-0.665096	-0.577774	-0.283462	-0.324258

```python
# import KNeighbors ClaSSifier from sklearn
from sklearn.neighbors import KNeighborsClassifier


# instantiate the model
knn = KNeighborsClassifier(n_neighbors=3)


# fit the model to the training set
knn.fit(X_train, y_train)
```

Outputs are collapsed ...

```python
y_pred = knn.predict(X_test)

y_pred
```

Outputs are collapsed ...

```python
knn.predict_proba(X_test)[:,0]
```

Outputs are collapsed ...

```python
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

Outputs are collapsed ...

```python
y_pred_train = knn.predict(X_train) ...
```

```python
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

Outputs are collapsed ...

Output:

```
df.shape
```

```
(699, 11)
```

```
col_names = ['Id', 'Clump_thickness', 'Uniformity_Cell_Size', 'Uniformity_Cell_Shape', 'Marginal_Adhesion',
             'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin', 'Normal_Nucleoli', 'Mitoses', 'Class']

df.columns = col_names

df.columns
```

```
Index(['Id', 'Clump_thickness', 'Uniformity_Cell_Size',
       'Uniformity_Cell_Shape', 'Marginal_Adhesion',
       'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin',
       'Normal_Nucleoli', 'Mitoses', 'Class'],
      dtype='object')
```

```
df.head()
```

	Id	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Clump_thickness              699 non-null    int64
 1   Uniformity_Cell_Size         699 non-null    int64
 2   Uniformity_Cell_Shape        699 non-null    int64
 3   Marginal_Adhesion            699 non-null    int64
 4   Single_Epithelial_Cell_Size  699 non-null    int64
 5   Bare_Nuclei                  699 non-null    object
 6   Bland_Chromatin              699 non-null    int64
 7   Normal_Nucleoli              699 non-null    int64
 8   Mitoses                      699 non-null    int64
 9   Class                        699 non-null    int64
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
```

```
for var in df.columns:

    print(df[var].value_counts())
```

```
Clump_thickness
1     145
5     130
3     108
4      80
10     69
2      50
8      46
6      34
7      23
9      14
Name: count, dtype: int64
Uniformity_Cell_Size
1     384
10     67
3      52
2      45
4      40
5      30
8      29
6      27
7      19
9       6
Name: count, dtype: int64
Uniformity_Cell_Shape
...
Class
2    458
4    241
Name: count, dtype: int64
```

```
[9]
df['Bare_Nuclei'] = pd.to_numeric(df['Bare_Nuclei'], errors='coerce')
```
Python

```
[10]
df.dtypes
```
Python

```
Clump_thickness                int64
Uniformity_Cell_Size           int64
Uniformity_Cell_Shape          int64
Marginal_Adhesion              int64
Single_Epithelial_Cell_Size    int64
Bare_Nuclei                  float64
Bland_Chromatin                int64
Normal_Nucleoli                int64
Mitoses                        int64
Class                          int64
dtype: object
```

```
[11]
df.isnull().sum()
```
Python

```
Clump_thickness                0
Uniformity_Cell_Size           0
Uniformity_Cell_Shape          0
Marginal_Adhesion              0
Single_Epithelial_Cell_Size    0
Bare_Nuclei                   16
Bland_Chromatin                0
Normal_Nucleoli                0
Mitoses                        0
Class                          0
dtype: int64
```

```
[12]
# check `na` values in the dataframe

df.isna().sum()
```
Python

```
Clump_thickness                0
Uniformity_Cell_Size           0
Uniformity_Cell_Shape          0
Marginal_Adhesion              0
Single_Epithelial_Cell_Size    0
Bare_Nuclei                   16
Bland_Chromatin                0
Normal_Nucleoli                0
Mitoses                        0
Class                          0
dtype: int64
```

```
[13]
# check frequency distribution of `Bare_Nuclei` column

df['Bare_Nuclei'].value_counts()
```
Python

```
Bare_Nuclei
1.0     402
10.0    132
2.0      30
5.0      30
3.0      28
8.0      21
4.0      19
9.0       9
7.0       8
6.0       4
Name: count, dtype: int64
```

```
[14]
# check unique values in `Bare_Nuclei` column

df['Bare_Nuclei'].unique()
```
Python

```
array([ 1., 10.,  2.,  4.,  3.,  9.,  7., nan,  5.,  8.,  6.])
```

```
# check for nan values in `Bare_Nuclei` column …
```

```
16
```

```
[16]
# view frequency distribution of values in `Class` variable

df['Class'].value_counts()
```
Python

```
Class
2    458
4    241
Name: count, dtype: int64
```

```
[18]
import numpy as np
```
Python

```
[21]
# view summary statistics in numerical variables

print(round(df.describe(),2))
```
Python

```
       Clump_thickness  Uniformity_Cell_Size  Uniformity_Cell_Shape  \
count           699.00                699.00                 699.00
mean              4.42                  3.13                   3.21
std               2.82                  3.05                   2.97
min               1.00                  1.00                   1.00
25%               2.00                  1.00                   1.00
50%               4.00                  1.00                   1.00
75%               6.00                  5.00                   5.00
max              10.00                 10.00                  10.00

       Marginal_Adhesion  Single_Epithelial_Cell_Size  Bare_Nuclei  \
count             699.00                       699.00       683.00
mean                2.81                         3.22         3.54
std                 2.86                         2.21         3.64
min                 1.00                         1.00         1.00
25%                 1.00                         2.00         1.00
50%                 1.00                         2.00         1.00
75%                 4.00                         4.00         6.00
max                10.00                        10.00        10.00

       Bland_Chromatin  Normal_Nucleoli  Mitoses   Class
count           699.00           699.00   699.00  699.00
mean              3.44             2.87     1.59    2.69
std               2.44             3.05     1.72    0.95
min               1.00             1.00     1.00    2.00
25%               2.00             1.00     1.00    2.00
50%               3.00             1.00     1.00    2.00
75%               5.00             4.00     1.00    4.00
max              10.00            10.00    10.00    4.00
```

```
X_test = pd.DataFrame(X_test, columns=[cols])
```

[30] Python

```
X_train.head()
```

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli	Mitoses
0	2.028383	0.299506	0.289573	1.119077	-0.546543	1.858357	-0.577774	0.041241	-0.324258
1	1.669451	2.257680	2.304569	-0.622471	3.106879	1.297589	-0.159953	0.041241	-0.324258
2	-1.202005	-0.679581	-0.717925	0.074148	-1.003220	-0.104329	-0.995595	-0.608165	-0.324258
3	-0.125209	-0.026856	-0.046260	-0.622471	-0.546543	-0.665096	-0.159953	0.041241	-0.324258
4	0.233723	-0.353219	-0.382092	-0.274161	-0.546543	-0.665096	-0.577774	-0.283462	-0.324258

```
# import KNeighbors ClaSSifier from sklearn
from sklearn.neighbors import KNeighborsClassifier


# instantiate the model
knn = KNeighborsClassifier(n_neighbors=3)


# fit the model to the training set
knn.fit(X_train, y_train)
```

[32] Python

```
KNeighborsClassifier(n_neighbors=3)
```

```
y_pred = knn.predict(X_test)

y_pred
```

[33] Python

```
array([2, 2, 4, 2, 4, 2, 4, 2, 4, 2, 2, 2, 4, 4, 4, 2, 2, 4, 4, 2, 4, 4,
       2, 2, 2, 4, 2, 2, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2,
       4, 4, 2, 4, 2, 4, 4, 4, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 4,
       4, 2, 2, 4, 2, 2, 4, 4, 4, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2,
       4, 4, 2, 2, 4, 2, 2, 2, 2, 4, 2, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 2, 4, 2, 2, 4, 4, 4, 4, 4, 2,
       2, 4, 4, 2, 2, 4, 4, 2, 2])
```

```
y_pred = knn.predict(X_test)

y_pred
```

[33] Python

```
array([2, 2, 4, 2, 4, 2, 4, 2, 4, 2, 2, 2, 4, 4, 4, 2, 2, 4, 4, 2, 4, 4,
       2, 2, 2, 4, 2, 2, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2,
       4, 4, 2, 4, 2, 4, 4, 4, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 4,
       4, 2, 2, 4, 2, 2, 4, 4, 4, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2,
       4, 4, 2, 2, 4, 2, 2, 2, 2, 4, 2, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 2, 4, 2, 2, 4, 4, 4, 4, 4, 2,
       2, 4, 4, 2, 2, 4, 4, 2, 2])
```

```
knn.predict_proba(X_test)[:,0]
```

[38] Python

```
array([1.        , 1.        , 0.33333333, 1.        , 0.        ,
       1.        , 0.        , 1.        , 0.        , 0.66666667,
       1.        , 1.        , 0.        , 0.33333333, 0.        ,
       1.        , 1.        , 0.        , 0.        , 1.        ,
       0.        , 0.        , 1.        , 1.        , 1.        ,
       0.        , 1.        , 1.        , 0.        , 0.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       0.66666667, 1.        , 0.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 0.        ,
       0.        , 1.        , 0.        , 1.        , 0.        ,
       0.        , 1.        , 1.        , 0.        , 1.        ,
       1.        , 1.        , 1.        , 0.66666667, 1.        ,
       0.        , 1.        , 1.        , 0.        , 0.        ,
       0.33333333, 0.        , 1.        , 1.        , 0.        ,
       1.        , 1.        , 0.        , 0.        , 1.        ,
       1.        , 1.        , 1.        , 0.        , 1.        ,
       1.        , 1.        , 0.        , 1.        , 1.        ,
       1.        , 0.        , 1.        , 0.        , 0.        ,
       1.        , 1.        , 0.66666667, 0.        , 1.        ,
       1.        , 1.        , 0.        , 1.        , 0.        ,
       0.        , 1.        , 1.        , 1.        , 0.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       0.        , 0.33333333, 0.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 0.        , 0.        ,
       0.        , 0.33333333, 1.        , 0.        , 1.        ,
       1.        , 0.33333333, 0.33333333, 0.        , 0.        ,
       0.        , 1.        , 1.        , 0.33333333, 0.        ,
       1.        , 1.        , 0.        , 1.        , 1.        ])
```

```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

[35] Python

```
Model accuracy score: 0.9714
```

```
y_pred_train = knn.predict(X_train)
```

```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

[37] Python

```
Training-set accuracy score: 0.9821
```

03/05/2024

Linear Regression

Lab - 3

Implement linear regression algo using appropriate dataset.

Algorithm

1. Import necessary libraries
2. Import dataset
3. Visualization of dataset using different plots like heatmap, distribution plot, scatterplot etc.
4. Preprocess the data, convert or encode categorical data
5. Split the dataset into training set and test set from sklearn model_selection import train_test_split

   X_train, X_test, y_train, y_test = train_Test_split(X, y, test_size=0.2, random_state=42)

6. Build Model
   from sklearn.linear_model import LinearRegression
   lin_reg = LinearRegression()

7. Fit the dataset to the model and train it

lin_reg.fit(x_train, y_train)

8. Calculate the accuracy using mean square error.

Output

meansquarederror = 0.9171732469e

Implement multilinear regression

1. Import necessary libraries like linear regression
2. Import dataset
3. Visualise the dataset using matplotlib.pyplot
4. Encode categorical data
   ct = ColumnTransformer(transform
   = [('encode' OneHotEncode
   [3])], remainder = '
   passthrough')
5. split dataset to training and test dataset
6. We can set multiple independent variables
7. create regressor model
   regressor = LinearRegression()
8. Fit the train set
9. Test the model using test set
10. Compare the actual value and predicd value

Code and output:

Linear Regression

```
import pandas as pd
```

```
salary = pd.read_csv('https://github.com/ybifoundation/Dataset/raw/main/Salary%20Data.csv')
```

```
y = salary['Salary']
```

```
X = salary[['Experience Years']]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size=0.7, random_state=2529)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((28, 1), (12, 1), (28,), (12,))
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
model.fit(X_train,y_train)
```

```
LinearRegression()
```

```
model.coef_
```

```
array([9405.62])
```

```
y_pred = model.predict(X_test)
```

```
y_pred
```

```
array([ 90555.15,  59516.62, 106544.7 ,  64219.43,  68922.24, 123474.81,
        84911.78,  63278.87,  65159.99,  61397.74,  37883.7 ,  50111.  ])
```

```
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, mean_squared_error
```

```
import numpy as np
```

```
mean_absolute_error(y_test,y_pred)
```

```
4005.9263101681768
```

Multiple Regression

```
house = pd.read_csv('https://github.com/YBIFoundation/Dataset/raw/main/Boston.csv')
```

```
house.head()
```

	CRIM	ZN	INDUS	CHAS	NX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

```
house.describe()
```

	CRIM	ZN	INDUS	CHAS	NX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

```
house.columns
```

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],
      dtype='object')
```

```
y = house['MEDV']
```

```
X = house.drop(['MEDV'],axis=1)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size=0.7, random_state=2529)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((354, 13), (152, 13), (354,), (152,))
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
# Step 6 : train or fit model
model.fit(X_train,y_train)
```

```
LinearRegression()
```

```
model.intercept_
```

```
34.21916368862993
```

```
model.coef_
```

```
array([-1.29e-01,  3.65e-02,  1.54e-02,  2.35e+00, -2.04e+01,  4.41e+00,
        4.61e-03, -1.59e+00,  2.51e-01, -9.60e-03, -9.64e-01,  1.01e-02,
       -5.43e-01])
```

```
# Step 7 : predict model
y_pred = model.predict(X_test)
```

```
y_pred
```

```
array([31.72, 22.02, 21.17, 39.78, 20.1 , 22.86, 18.36, 14.79, 22.56,
       21.35, 18.38, 27.97, 29.86,  6.45, 10.68, 26.25, 21.89, 25.23,
        3.62, 36.22, 24.08, 22.94, 14.27, 20.79, 24.23, 16.74, 18.75,
       20.97, 28.51, 20.86,  9.23, 17.07, 22.07, 22.23, 39.26, 26.17,
       42.5 , 19.35, 34.52, 14.07, 13.81, 23.28, 11.79,  9.01, 21.65,
       25.55, 18.17, 16.82, 14.66, 14.86, 33.79, 33.27, 15.49, 24.08,
       27.64, 19.58, 45.02, 20.97, 20.07, 27.67, 34.59, 12.71, 23.66,
       31.66, 28.97, 32.46, 13.93, 35.49, 19.36, 19.6 ,  1.44, 24.1 ,
       33.67, 20.62, 26.89, 21.29, 31.95, 29.74, 13.93, 13.82, 19.76,
       21.54, 20.87, 23.63, 28.8 , 23.64,  6.95, 22.2 , -6.82, 16.97,
       16.77, 25.44, 14.95,  3.72, 15.03, 16.91, 21.46, 31.66, 30.72,
       23.73, 22.19, 13.76, 18.47, 18.15, 36.6 , 27.49, 11.  , 17.26,
       22.49, 16.53, 29.49, 22.89, 24.68, 20.38, 19.69, 22.55, 27.32,
       24.86, 20.2 , 29.14,  7.43,  5.85, 25.35, 38.73, 23.94, 25.28,
       20.11, 19.75, 25.07, 35.16, 27.32, 27.26, 31.4 , 16.55, 14.3 ,
       23.77,  7.65, 23.35, 21.37, 26.12, 25.32, 13.12, 17.67, 36.2 ,
       20.5 , 27.95, 22.46, 18.15, 31.24, 20.85, 27.36, 30.53])
```

```
# Step 8 : model accuracy
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, mean_squared_error
```

```
mean_absolute_error(y_test,y_pred)
```

```
3.155030927602485
```

03/05/2024

Logistic Regression

Lab - 4

Implement Logistic Regression

1) Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.linear_model
import LogisticRegression

2) Load the dataset
iris = load_iris
X = iris.data
y = iris.target

3) Create model and train
model = LogisticRegression()
model.fit(df.drop('output', df.out

4) Predict value
model.predict(df.loc['row'])

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os
```

[3] Python

```python
data = pd.read_csv('/content/data.csv')
```

[4] Python

```python
data.drop(['Unnamed: 32',"id"], axis=1, inplace=True)
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]
y = data.diagnosis.values
x_data = data.drop(['diagnosis'], axis=1)
```

[5] Python

```python
# Assuming x_data is a numpy array or pandas DataFrame
x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data))
```

[7] Python

```python
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.15, random_state=42)

x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T

print("x train: ",x_train.shape)
print("x test: ",x_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)
```

[8] Python

```
x train:  (30, 483)
x test:  (30, 86)
y train:  (483,)
y test:  (86,)
```

```python
def initialize_weights_and_bias(dimension):
    w = np.full((dimension,1),0.01)
    b = 0.0
    return w, b
```

[9] Python

```python
def sigmoid(z):
    y_head = 1/(1+np.exp(-z))
    return y_head
```

[10] Python

```python
def forward_backward_propagation(w,b,x_train,y_train):
    # forward propagation
    z = np.dot(w.T,x_train) + b
    y_head = sigmoid(z)
    loss = -y_train*np.log(y_head)-(1-y_train)*np.log(1-y_head)
    cost = (np.sum(loss))/x_train.shape[1]      # x_train.shape[1]  is for scaling
    # backward propagation
    derivative_weight = (np.dot(x_train,((y_head-y_train).T)))/x_train.shape[1] # x_train.shape[1]  is for scaling
    derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]                  # x_train.shape[1]  is for scaling
    gradients = {"derivative_weight": derivative_weight,"derivative_bias": derivative_bias}
    return cost,gradients
```

Python

```python
def update(w, b, x_train, y_train, learning_rate,number_of_iterarion):
    cost_list = []
    cost_list2 = []
    index = []
    # updating(learning) parameters is number_of_iterarion times
    for i in range(number_of_iterarion):
        # make forward and backward propagation and find cost and gradients
        cost,gradients = forward_backward_propagation(w,b,x_train,y_train)
        cost_list.append(cost)
        w = w - learning_rate * gradients["derivative_weight"]
        b = b - learning_rate * gradients["derivative_bias"]
        if i % 10 == 0:
            cost_list2.append(cost)
            index.append(i)
            print ("Cost after iteration %i: %f" %(i, cost))
    # we update(learn) parameters weights and bias
    parameters = {"weight": w,"bias": b}
    plt.plot(index,cost_list2)
    plt.xticks(index,rotation='vertical')
    plt.xlabel("Number of Iterarion")
    plt.ylabel("Cost")
    plt.show()
    return parameters, gradients, cost_list
```

Python

```python
def predict(w,b,x_test):
    # x_test is a input for forward propagation
    z = sigmoid(np.dot(w.T,x_test)+b)
    Y_prediction = np.zeros((1,x_test.shape[1]))
    # if z is bigger than 0.5, our prediction is sign one (y_head=1),
    # if z is smaller than 0.5, our prediction is sign zero (y_head=0),
    for i in range(z.shape[1]):
        if z[0,i]<= 0.5:
            Y_prediction[0,i] = 0
        else:
            Y_prediction[0,i] = 1

    return Y_prediction
```

[11] Python

```
!pip install mymodule
```

[13] Python

```
Collecting mymodule
  Downloading myModule-1.0.0.tar.gz (787 bytes)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: mymodule
  Building wheel for mymodule (setup.py) ... done
  Created wheel for mymodule: filename=myModule-1.0.0-py3-none-any.whl size=1413 sha256=2cdcfae4d015fdc631a533b7e726b48c29061c95d1a6a50b070d71cb04054f3f
  Stored in directory: /root/.cache/pip/wheels/f4/64/ed/4ac2e7514f3e85b8aea309dff4c236817b10c4d0809e5511d8
Successfully built mymodule
Installing collected packages: mymodule
Successfully installed mymodule-1.0.0
```

```python
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def initialize_weights_and_bias(dim):
    w = np.zeros((dim, 1))
    b = 0
    return w, b

def compute_cost(w, b, X, y):
    m = X.shape[1]
    A = sigmoid(np.dot(w.T, X) + b)
    cost = -1 / m * np.sum(y * np.log(A) + (1 - y) * np.log(1 - A))
    return cost

def propagate(w, b, X, y):
    m = X.shape[1]
    A = sigmoid(np.dot(w.T, X) + b)
    dw = 1 / m * np.dot(X, (A - y).T)
    db = 1 / m * np.sum(A - y)
    return dw, db

def logistic_regression(x_train, y_train, x_test, y_test, learning_rate, num_iterations):
    # Initialize
    dimension = x_train.shape[0]  # number of features
    w, b = initialize_weights_and_bias(dimension)
    costs = []

    # Gradient Descent
    for i in range(num_iterations):
        # Forward and Backward Propagation
        dw, db = propagate(w, b, x_train, y_train)

        # Update parameters
        w -= learning_rate * dw
        b -= learning_rate * db

        # Record the costs
        if i % 100 == 0:
            cost = compute_cost(w, b, x_train, y_train)
            costs.append(cost)
            print(f"Cost after iteration {i}: {cost}")

    # Evaluate model
    y_prediction_train = predict(w, b, x_train)
    y_prediction_test = predict(w, b, x_test)

    train_accuracy = 100 - np.mean(np.abs(y_prediction_train - y_train)) * 100
    test_accuracy = 100 - np.mean(np.abs(y_prediction_test - y_test)) * 100

    print("Train accuracy: {} %".format(train_accuracy))
    print("Test accuracy: {} %".format(test_accuracy))

    return w, b

# assuming you have defined the predict function
# def predict(w, b, x):
#     ...

# assuming you have defined x_train, y_train, x_test, y_test, learning_rate, and num_iterations
logistic_regression(x_train, y_train, x_test, y_test, learning_rate=1, num_iterations=100)
```

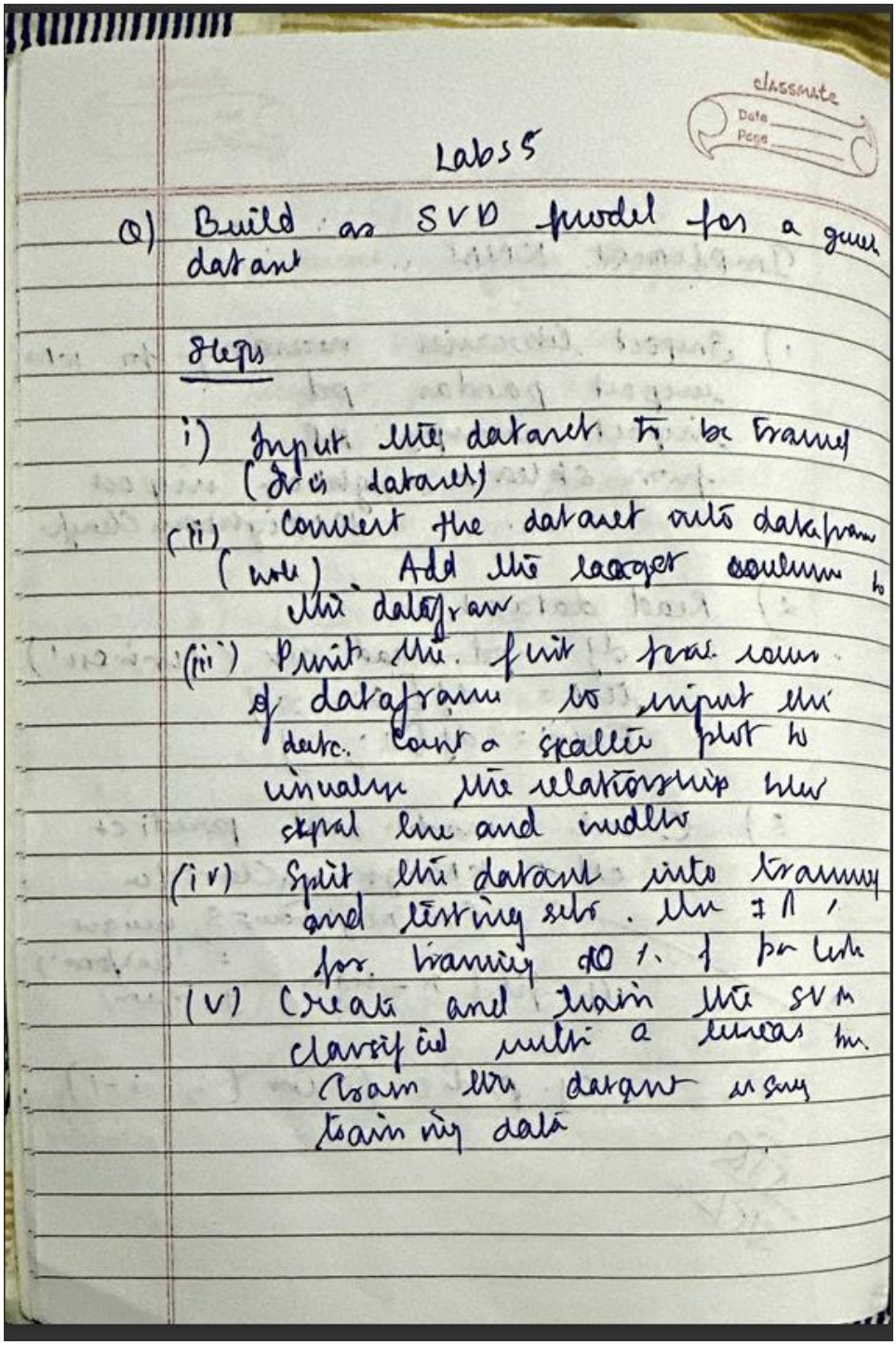
```
Cost after iteration 0: 0.6782740160052536
Train accuracy: 80.74534161490683 %
Test accuracy: 81.3953488372093 %

(array([[ 1.77806654e-02],
        [ 1.10160388e-02],
        [ 1.27806976e-01],
        [ 1.95749649e+00],
        [ 1.85931875e-05],
        [ 2.68863405e-04],
        [ 4.89020048e-04],
        [ 2.63106803e-04],
        [ 3.49357933e-05],
        [-2.02145931e-05],
        [ 1.25690784e-03],
        [-3.98285024e-04],
        [ 8.96937014e-03],
        [ 2.02426962e-01],
        [-3.60718647e-06],
        [ 4.19150446e-05],
        [ 6.03411729e-05],
        [ 2.00740406e-05],
        [-6.24803672e-06],
        [ 6.24944780e-07],
        [ 2.79506973e-02],
        [ 1.99326360e-02],
        [ 1.98774929e-01],
        [ 3.39188908e+00],
        [ 5.79135019e-05],
 ...
        [ 1.25862280e-03],
        [ 4.60695564e-04],
        [ 1.89671301e-04],
        [ 3.52490835e-05]]),
 -1.5161875221606185)
```

24/05/2024

Build Support vector machine model for a given dataset

Algorithm:



Lab 5

Q) Build an SVD model for a given dataset

Steps

i) Import the dataset to be trained (Iris dataset)

(ii) Convert the dataset into dataframe (note) Add the target column to the dataframe

(iii) Print the first few rows of dataframe to input the data. Create a scatter plot to visualize the relationship between sepal line and width

(iv) Split the dataset into training and testing sets. Use 7:1, for training 80% of the data

(v) Create and train the SVM classifier with a linear kernel. Train the dataset using training data

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
```

```python
df = pd.read_csv('/content/drive/MyDrive/breast-cancer.csv')
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

5 rows × 32 columns

```python
df.drop('id', axis=1, inplace=True) #drop redundant columns
```


```python
df.drop('id', axis=1, inplace=True) #drop redundant columns
```


```python
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
radius_mean	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.11000
texture_mean	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.28000
perimeter_mean	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.50000
area_mean	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.00000
smoothness_mean	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.16340
compactness_mean	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.34540
concavity_mean	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.42680
concave points_mean	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.20120
symmetry_mean	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.195700	0.30400
fractal_dimension_mean	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.066120	0.09744
radius_se	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.478900	2.87300
texture_se	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.474000	4.88500
perimeter_se	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.357000	21.98000
area_se	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.20000
smoothness_se	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.008146	0.03113
compactness_se	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.032450	0.13540
concavity_se	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.042050	0.39600
concave points_se	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.014710	0.05279
symmetry_se	569.0	0.020542	0.008266	0.007882	0.015160	0.018730	0.023480	0.07895
fractal_dimension_se	569.0	0.003795	0.002646	0.000895	0.002248	0.003187	0.004558	0.02984
radius_worst	569.0	16.269190	4.833242	7.930000	13.010000	14.970000	18.790000	36.04000
texture_worst	569.0	25.677223	6.146258	12.020000	21.080000	25.410000	29.720000	49.54000
perimeter_worst	569.0	107.261213	33.602542	50.410000	84.110000	97.660000	125.400000	251.20000
area_worst	569.0	880.583128	569.356993	185.200000	515.300000	686.500000	1084.000000	4254.00000
smoothness_worst	569.0	0.132369	0.022832	0.071170	0.116900	0.131300	0.146000	0.22260
compactness_worst	569.0	0.254265	0.157336	0.027290	0.147200	0.211900	0.339100	1.05800
concavity_worst	569.0	0.272188	0.208624	0.000000	0.114500	0.226700	0.382900	1.25200
concave points_worst	569.0	0.114606	0.065732	0.000000	0.064930	0.099930	0.161400	0.29100
symmetry_worst	569.0	0.290076	0.061867	0.156500	0.250400	0.282200	0.317900	0.66380
fractal_dimension_worst	569.0	0.083946	0.018061	0.055040	0.071460	0.080040	0.092080	0.20750

```python
df['diagnosis'] = (df['diagnosis'] == 'M').astype(int) #encode the label into 1/0
```

```python
corr = df.corr()
```

```python
# Get the absolute value of the correlation
cor_target = abs(corr['diagnosis'])

# Select highly correlated features (threshold = 0.2)
relevant_features = cor_target[cor_target>0.2]

# Collect the names of the features
names = [index for index, value in relevant_features.items()]

# Drop the target variable from the results
names.remove('diagnosis')

# Display the results
print(names)
```

```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', 'radius_se', 'perimeter_se', 'area_se', 'compactness_se', 'concavity_se', 'concave points_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst', '
```

```python
X = df[names].values
y = df['diagnosis']
```

```python
def scale(X):
    """
    Standardizes the data in the array X.

    Parameters:
        X (numpy.ndarray): Features array of shape (n_samples, n_features).

    Returns:
        numpy.ndarray: The standardized features array.
    """
    # Calculate the mean and standard deviation of each feature
    mean = np.mean(X, axis=0)
    std = np.std(X, axis=0)

    # Standardize the data
    X = (X - mean) / std
    return X
```

```python
X = scale(X)
```


```python
X = scale(X)
```


```python
def train_test_split(X, y, random_state=42, test_size=0.2):
    """
    Splits the data into training and testing sets.

    Parameters:
        X (numpy.ndarray): Features array of shape (n_samples, n_features).
        y (numpy.ndarray): Target array of shape (n_samples,).
        random_state (int): Seed for the random number generator. Default is 42.
        test_size (float): Proportion of samples to include in the test set. Default is 0.2.

    Returns:
        tuple[numpy.ndarray]: A tuple containing X_train, X_test, y_train, y_test.
    """
    # Get number of samples
    n_samples = X.shape[0]

    # Set the seed for the random number generator
    np.random.seed(random_state)

    # Shuffle the indices
    shuffled_indices = np.random.permutation(np.arange(n_samples))

    # Determine the size of the test set
    test_size = int(n_samples * test_size)

    # Split the indices into test and train
    test_indices = shuffled_indices[:test_size]
    train_indices = shuffled_indices[test_size:]

    # Split the features and target arrays into test and train
    X_train, X_test = X[train_indices], X[test_indices]
    y_train, y_test = y[train_indices], y[test_indices]

    return X_train, X_test, y_train, y_test
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42) #split the  data into trasing and validatin
```

```python
class SVM
    """
    A Support Vector Machine (SVM) implementation using gradient descent.

    Parameters:
    -----------
    iterations : int, default=1000
        The number of iterations for gradient descent.
    lr : float, default=0.01
        The learning rate for gradient descent.
    lambdaa : float, default=0.01
        The regularization parameter.

    Attributes:
    -----------
    lambdaa : float
        The regularization parameter.
    iterations : int
        The number of iterations for gradient descent.
    lr : float
        The learning rate for gradient descent.
    w : numpy array
        The weights.
    b : float
        The bias.

    Methods:
    --------
    initialize_parameters(X)
        Initializes the weights and bias.
    gradient_descent(X, y)
        Updates the weights and bias using gradient descent.
    update_parameters(dw, db)
        Updates the weights and bias.
    fit(X, y)
        Fits the SVM to the data.
    predict(X)
        Predicts the labels for the given data.

    """

    def __init__(self, iterations=1000, lr=0.01, lambdaa=0.01):
        """
        Initializes the SVM model.

        Parameters:
        -----------
        iterations : int, default=1000
            The number of iterations for gradient descent.
        lr : float, default=0.01
            The learning rate for gradient descent.
        lambdaa : float, default=0.01
            The regularization parameter.
        """
        self.lambdaa = lambdaa
        self.iterations = iterations
        self.lr = lr
        self.w = None
        self.b = None
    def initialize_parameters(self, X):
        """
        Initializes the weights and bias.

        Parameters:
        ---------
        X : numpy array
            The input data.
        """
```

```
        n, n = X.shape
        self.w = np.zeros(n)
        self.b = 0

    def gradient_descent(self, X, y):
        """
        Updates the weights and bias using gradient descent.

        Parameters:
        -----------
        X : numpy array
            The input data.
        y : numpy array
            The target values.
        """
        y_ = np.where(y <= 0, -1, 1)
        for i, x in enumerate(X):
            if y_[i] * (np.dot(x, self.w) - self.b) >= 1:
                dw = 2 * self.lambdaa * self.w
                db = 0
            else:
                dw = 2 * self.lambdaa * self.w - np.dot(x, y_[i])
                db = y_[i]
            self.update_parameters(dw, db)

    def update_parameters(self, dw, db):
        """
        Updates the weights and bias.

        Parameters:
        -----------
        dw : numpy array
            The change in weights.
        db : float
            The change in bias.
        """
        self.w = self.w - self.lr * dw
        self.b = self.b - self.lr * db
    def fit(self, X, y):
        """
        Fits the SVM to the data.

        Parameters:
        -----------
        X : numpy array
            The input data.
        y : numpy array
            The target values.
        """
        self.initialize_parameters(X)
        for i in range(self.iterations):
            self.gradient_descent(X, y)


    def predict(self, X):
        """
        Predicts the class labels for the test data.

        Parameters
        ----------
        X : array-like, shape (n_samples, n_features)
            The input data.

        Returns
        -------
        y_pred : array-like, shape (n_samples,)
            The predicted class labels.

        """
        # get the output
        output = np.dot(X, self.w) - self.b
        # get the signs of the labels depending on if it's greater/less than zero
        label_signs = np.sign(output)
        #set predictions to 0 if they are less than or equal to -1 else set them to 1
        predictions = np.where(label_signs <= -1, 0, 1)
        return predictions
```

Python

```
def accuracy(y_true, y_pred):
    """
    Computes the accuracy of a classification model.

    Parameters:
    ----------
        y_true (numpy array): A numpy array of true labels for each data point.
        y_pred (numpy array): A numpy array of predicted labels for each data point.

    Returns:
    -------
        float: The accuracy of the model
    """
    total_samples = len(y_true)
    correct_predictions = np.sum(y_true == y_pred)
    return (correct_predictions / total_samples)
```

Python

```
model = SVM()
model.fit(X_train,y_train)
predictions = model.predict(X_test)

accuracy(y_test, predictions)
```

Python

```
0.9823008849557522
```

```
model.predict([-0.47054438, -0.15948554, -0.44038956, -0.48339875,  0.23411425,
       -0.02765851, -0.38884741, 0.27623152,  0.41394897, -0.0327429 ,
       -0.18269561, 0.22385292, 0.35591235, 0.16292349, -0.23133322,
       0.25963951, -0.16896536, -0.22393537, 0.35629925,  0.0485828 ,
       -0.18478865, -0.02441212, -0.25056328,  0.18328441,  0.10585704])
```

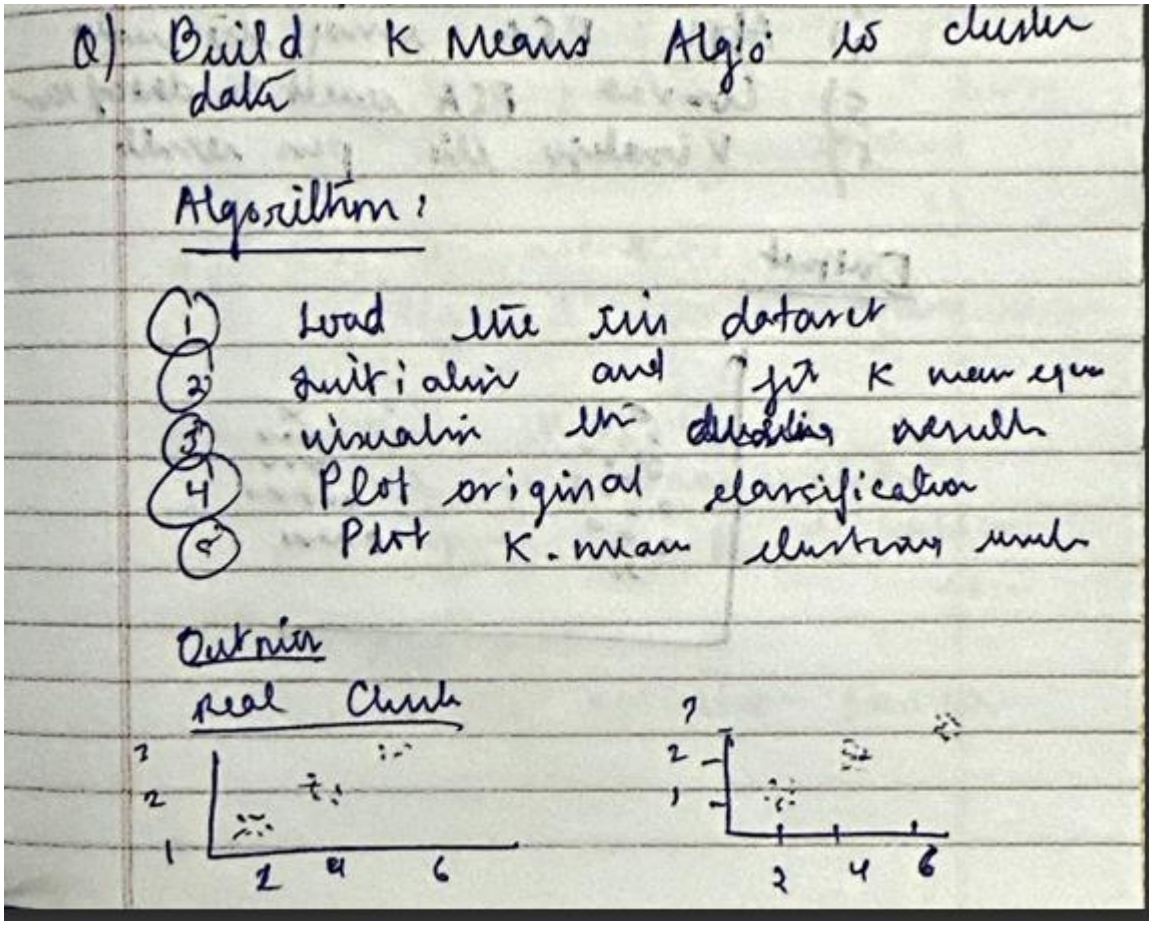
Python

```
array(0)
```

24/05/2024

Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Algorithm:



```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
import plotly.graph_objects as go
```

```python
iris = pd.read_csv("/content/drive/MyDrive/Iris.csv") #Load Data
iris.drop('Id',inplace=True,axis=1) #Drop id column
```

```python
X = iris.iloc[:,:-1] #Set our training data

y = iris.iloc[:,-1] #We'll use this just for visualization as clustering doesn't require labels
```

```python
class Kmeans:
    """
    K-Means clustering algorithm implementation.

    Parameters:
        K (int): Number of clusters

    Attributes:
        K (int): Number of clusters
        centroids (numpy.ndarray): Array containing the centroids of each cluster

    Methods:
        __init__(self, K): Initializes the Kmeans instance with the specified number of clusters.
        initialize_centroids(self, X): Initializes the centroids for each cluster by selecting K random points from the dataset.
        assign_points_centroids(self, X): Assigns each point in the dataset to the nearest centroid.
        compute_mean(self, X, points): Computes the mean of the points assigned to each centroid.
        fit(self, X, iterations=10): Clusters the dataset using the K-Means algorithm.
    """

    def __init__(self, K):
        assert K > 0, "K should be a positive integer."
        self.K = K

    def initialize_centroids(self, X):
        assert X.shape[0] >= self.K, "Number of data points should be greater than or equal to K."

        randomized_X = np.random.permutation(X.shape[0])
        centroid_idx = randomized_X[:self.K] # get the indices for the centroids
        self.centroids = X[centroid_idx] # assign the centroids to the selected points

    def assign_points_centroids(self, X):
        """
        Assign each point in the dataset to the nearest centroid.

        Parameters:
        X (numpy.ndarray): dataset to cluster

        Returns:
        numpy.ndarray: array containing the index of the centroid for each point
        """
        X = np.expand_dims(X, axis=1) # expand dimensions to match shape of centroids
        distance = np.linalg.norm(X - self.centroids, axis=-1) # calculate Euclidean distance between each point and each centroid
        points = np.argmin(distance, axis=1) # assign each point to the closest centroid
        assert len(points) == X.shape[0], "Number of assigned points should equal the number of data points."
        return points

    def compute_mean(self, X, points):
        """
        Compute the mean of the points assigned to each centroid.

        Parameters:
        X (numpy.ndarray): dataset to cluster
        points (numpy.ndarray): array containing the index of the centroid for each point

        Returns:
        numpy.ndarray: array containing the new centroids for each cluster
        """
        centroids = np.zeros((self.K, X.shape[1])) # initialize array to store centroids
        for i in range(self.K):
            centroid_mean = X[points == i].mean(axis=0) # calculate mean of the points assigned to the current centroid
            centroids[i] = centroid_mean # assign the new centroid to the mean of its points
        return centroids

    def fit(self, X, iterations=10):
        """
        Cluster the dataset using the K-Means algorithm.

        Parameters:
        X (numpy.ndarray): dataset to cluster
        iterations (int): number of iterations to perform (default=10)

        Returns:
        numpy.ndarray: array containing the final centroids for each cluster
        numpy.ndarray: array containing the index of the centroid for each point
        """
        self.initialize_centroids(X) # initialize the centroids
        for i in range(iterations):
            points = self.assign_points_centroids(X) # assign each point to the nearest centroid
            self.centroids = self.compute_mean(X, points) # compute the new centroids based on the mean of their points

            # assertions for debugging and validation
            assert len(self.centroids) == self.K, "Number of centroids should equal K."
            assert X.shape[1] == self.centroids.shape[1], "Dimensionality of centroids should match input data."
            assert max(points) < self.K, "Cluster index should be less than K."
            assert min(points) >= 0, "Cluster index should be non-negative."

        return self.centroids, points
```

```python
X = X.values
```

```python
kmeans = Kmeans(3)

centroids, points = kmeans.fit(X, 1000)
```

```python
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=X[points == 0, 0], y=X[points == 0, 1],
    mode='markers',marker_color='#DB4C82',name='Iris-setosa'
))

fig.add_trace(go.Scatter(
    x=X[points == 1, 0], y=X[points == 1, 1],
    mode='markers',marker_color='#43B5F6',name='Iris-versicolour'
))

fig.add_trace(go.Scatter(
    x=X[points == 2, 0], y=X[points == 2, 1],
    mode='markers',marker_color='#7D5AC1',name='Iris-virginica'
))

fig.add_trace(go.Scatter(
    x=centroids[:, 0], y=centroids[:,1],
    mode='markers',marker_color='#CACBCD',marker_symbol=4,marker_size=13,name='Centroids'
))
fig.update_layout(template='plotly_dark',width=1000, height=500,)
```

24/05/2024

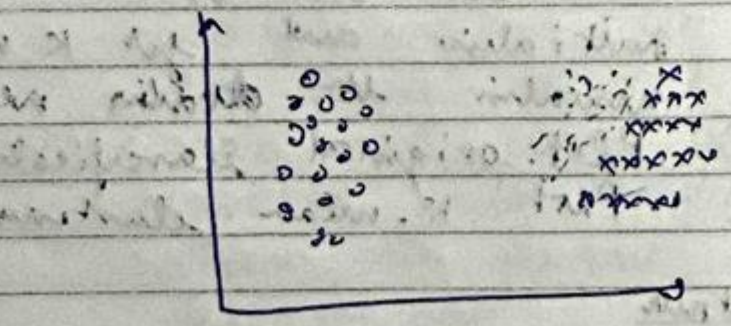
Implement Dimensionality reduction using Principle Component Analysis (PCA) Algorithm:

Q). Implement dimensionality reduction using principle component analysis method

Algorithm:

1) Import necessary libraries for data handling
2) Load the iris dataset
3) Standardize the data
4) Apply PCA : using the model
5) Convert PCA result to dataframe
6) Visualize the pca result

Output



```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```python
df = pd.read_csv('/content/drive/myDrive/breast-cancer.csv')
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

5 rows × 32 columns

```python
df.drop('id', axis=1, inplace=True) #drop redundant columns
```

```python
df['diagnosis'] = (df['diagnosis'] == 'M').astype(int) #encode the label into 1/0
```

```python
corr = df.corr()
```

```python
# Get the absolute value of the correlation
cor_target = abs(corr["diagnosis"])

# Select highly correlated features (threshold = 0.2)
relevant_features = cor_target[cor_target>0.2]

# Collect the names of the features
names = [index for index, value in relevant_features.items()]

# Drop the target variable from the results
names.remove('diagnosis')

# Display the results
print(names)
```

```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', 'radius_se', 'perimeter_se', 'area_se', 'compactness_se', 'concavity_se', 'concave points_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst',
```

```python
X = df[names].values
```

```python
class PCA:
    """
    Principal Component Analysis (PCA) class for dimensionality reduction.
    """

    def __init__(self, n_components):
        """
        Constructor method that initializes the PCA object with the number of components to retain.

        Args:
        - n_components (int): Number of principal components to retain.
        """
        self.n_components = n_components
    def fit(self, X):
        """
        Fits the PCA model to the input data and computes the principal components.

        Args:
        - X (numpy.ndarray): Input data matrix with shape (n_samples, n_features).
        """
        # Compute the mean of the input data along each feature dimension.
        mean = np.mean(X, axis=0)

        # Subtract the mean from the input data to center it around zero.
        X = X - mean

        # Compute the covariance matrix of the centered input data.
        cov = np.cov(X.T)

        # Compute the eigenvectors and eigenvalues of the covariance matrix.
        eigenvalues, eigenvectors = np.linalg.eigh(cov)
        # Reverse the order of the eigenvalues and eigenvectors.
        eigenvalues = eigenvalues[::-1]
        eigenvectors = eigenvectors[:,::-1]

        # Keep only the first n_components eigenvectors as the principal components.
        self.components = eigenvectors[:,:self.n_components]

        # Compute the explained variance ratio for each principal component.
        # Compute the total variance of the input data
        total_variance = np.sum(np.var(X, axis=0))

        # Compute the variance explained by each principal component
        self.explained_variances = eigenvalues[:self.n_components]

        # Compute the explained variance ratio for each principal component
        self.explained_variance_ratio_ = self.explained_variances / total_variance
    def transform(self, X):
        """
        Transforms the input data by projecting it onto the principal components.

        Args:
        - X (numpy.ndarray): Input data matrix with shape (n_samples, n_features).

        Returns:
        - transformed_data (numpy.ndarray): Transformed data matrix with shape (n_samples, n_components).
        """
        # Center the input data around zero using the mean computed during the fit step.
        X = X - np.mean(X, axis=0)

        # Project the centered input data onto the principal components.
        transformed_data = np.dot(X, self.components)

        return transformed_data

    def fit_transform(self, X):
        """
        Fits the PCA model to the input data and computes the principal components then
        transforms the input data by projecting it onto the principal components.

        Args:
        - X (numpy.ndarray): Input data matrix with shape (n_samples, n_features).
        """
        self.fit(X)
        transformed_data = self.transform(X)
        return transformed_data
```

```python
pca = PCA(2)
```

```python
pca.fit(X)
```

```python
pca.explained_variance_ratio_
```

```
array([0.98377428, 0.01630498])
```

```python
X_transformed = pca.transform(X)
```

```python
X_transformed[:,1].shape
```

```
(569,)
```

```python
fig = px.scatter(x=X_transformed[:,0], y=X_transformed[:,1])
fig.update_layout(
    title="PCA transformed data for breast cancer dataset",
    xaxis_title="PC1",
    yaxis_title="PC2"
)
fig.show()
```

31.05.2024

Build Artificial Neural Network model with back propagation on a given dataset

Algorithm:

Lab - 6

a) Build an ANN model with back propagation.

Algorithm :-

1) Initialize parameters
   - Normalize input feature matrix 'x'
   - Normalize the output 'y'
   - Set hyperparameters : no of epochs, no of neurons

2) Define activation func
   - sigmoid func adjustments

3) Training the network
   - Forward propagation
   - Compute i/p to hidden layer
   - Add bias
   - apply activation function

4) Back Propagation
→ compute error
→ compute gradients
→ compute delta

5) Update weights and bias

Output :- I/P [[0.6667, 1]
[0.333, 0.556]
[0.1, 0.667]]

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

Predicted Output : [[0.8078...]
[0.7929...]
[0.7011...]]

```
import numpy as np
from sklearn.model_selection import train_test_split

db = np.loadtxt("/content/duke-breast-cancer.txt")
print('Database raw shape (%s,%s)' % np.shape(db))
```

```
Database raw shape (86,7130)
```

```
np.random.shuffle(db)
y = db[:, 0]
x = np.delete(db, [0], axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)
print(np.shape(x_train),np.shape(x_test))
```

```
(77, 7129) (9, 7129)
```

```
hidden_layer = np.zeros(72)
weights = np.random.random((len(x[0]), 72))
output_layer = np.zeros(2)
hidden_weights = np.random.random((72, 2))
```

```
def sum_function(weights, index_locked_col, x):
    result = 0
    for i in range(0, len(x)):
        result += x[i] * weights[i][index_locked_col]
    return result
```

```
def activate_layer(layer, weights, x):
    for i in range(0, len(layer)):
        layer[i] = 1.7159 * np.tanh(2.0 * sum_function(weights, i, x) / 3.0)
```

```
def soft_max(layer):
    soft_max_output_layer = np.zeros(len(layer))
    for i in range(0, len(layer)):
        denominator = 0
        for j in range(0, len(layer)):
            denominator += np.exp(layer[j] - np.max(layer))
        soft_max_output_layer[i] = np.exp(layer[i] - np.max(layer)) / denominator
    return soft_max_output_layer
```

```
def recalculate_weights(learning_rate, weights, gradient, activation):
    for i in range(0, len(weights)):  # Add to CodeGPT Chat (Ctrl + Shift + E).        Add to chat (Ctrl+L) | Edit highlighted code (Ctrl+I).
        for j in range(0, len(weights[i])):
            weights[i][j] = (learning_rate * gradient[j] * activation[i]) + weights[i][j]
```

```
def back_propagation(hidden_layer, output_layer, one_hot_encoding, learning_rate, x):
    output_derivative = np.zeros(2)
    output_gradient = np.zeros(2)
    for i in range(0, len(output_layer)):
        output_derivative[i] = (1.0 - output_layer[i]) * output_layer[i]
    for i in range(0, len(output_layer)):
        output_gradient[i] = output_derivative[i] * (one_hot_encoding[i] - output_layer[i])
    hidden_derivative = np.zeros(72)
    hidden_gradient = np.zeros(72)
    for i in range(0, len(hidden_layer)):
        hidden_derivative[i] = (1.0 - hidden_layer[i]) * (1.0 + hidden_layer[i])
    for i in range(0, len(hidden_layer)):
        sum_ = 0
        for j in range(0, len(output_gradient)):
            sum_ += output_gradient[j] * hidden_weights[i][j]
        hidden_gradient[i] = sum_ * hidden_derivative[i]
    recalculate_weights(learning_rate, hidden_weights, output_gradient, hidden_layer)
    recalculate_weights(learning_rate, weights, hidden_gradient, x)
```

```
one_hot_encoding = np.zeros((2,2))
for i in range(0, len(one_hot_encoding)):
    one_hot_encoding[i][i] = 1
training_correct_answers = 0
for i in range(0, len(x_train)):
    activate_layer(hidden_layer, weights, x_train[i])
    activate_layer(output_layer, hidden_weights, hidden_layer)
    output_layer = soft_max(output_layer)
    training_correct_answers += 1 if y_train[i] == np.argmax(output_layer) else 0
    back_propagation(hidden_layer, output_layer, one_hot_encoding[int(y_train[i])], -1, x_train[i])
print('MLP Correct answers while learning: %s / %s (Accuracy = %s) on %s database.' % (training_correct_answers, len(x_train),
                                                                                       training_correct_answers/len(x_train),"Duke breast cancer"))
```

```
MLP Correct answers while learning: 44 / 77 (Accuracy = 0.5714285714285714) on Duke breast cancer database.
```

```
testing_correct_answers = 0
for i in range(0, len(x_test)):
    activate_layer(hidden_layer, weights, x_test[i])
    activate_layer(output_layer, hidden_weights, hidden_layer)
    output_layer = soft_max(output_layer)
    testing_correct_answers += 1 if y_test[i] == np.argmax(output_layer) else 0
print('MLP Correct answers while testing: %s / %s (Accuracy = %s) on %s database' % (testing_correct_answers, len(x_test),
                                                                                     testing_correct_answers/len(x_test), "Duke breast cancer"))
```

```
MLP Correct answers while testing: 0 / 9 (Accuracy = 0.0000000000000000) on Duke breast cancer database
```

31.05.2024
Implement Random forest ensemble method on a given dataset.

Algorithm:

6) Implement random forest ensemble method

Algorithm:

1) Import necessary libraries
2) Load and inspect data
3) Pre process the data
4) Split data to train and test
5) Initialize RF classifier and train using fit method
6) Make predictions on the sampling using method predict
7) Evaluate the model

Output: Accuracy: 0.98

Confusion Matrix [[23 0 0
0 19 0
0 1 12]]

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

```python
iris = pd.read_csv("/content/drive/MyDrive/Iris.csv") #Load Data
iris.drop('Id',inplace=True,axis=1) #Drop Id column
```

```python
iris.head().style.background_gradient(cmap =sns.light_palette("seagreen", as_cmap=True)
)
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5.000000	3.600000	1.400000	0.200000	Iris-setosa

```python
X_df = iris.iloc[:,:-1] #Set our training dataframe

y_df = iris.iloc[:,-1] # Set our training labels dataframe
```

```python
fig = px.pie(iris, 'Species',color_discrete_sequence=['#3dec84 ','#A69E88 ','#2E8B57 '],title='Data Distribution',template='plotly')

fig.show()
```

```python
iris['Species'] = iris['Species'].astype("category")
codes = iris['Species'].cat.codes
```

```python
def train_test_split(X, y, random_state=42, test_size=0.2):
    """
    Splits the data into training and testing sets.

    Parameters:
        X (numpy.ndarray): Features array of shape (n_samples, n_features).
        y (numpy.ndarray): Target array of shape (n_samples,).
        random_state (int): Seed for the random number generator. Default is 42.
        test_size (float): Proportion of samples to include in the test set. Default is 0.2.

    Returns:
        Tuple[numpy.ndarray]: A tuple containing X_train, X_test, y_train, y_test.
    """
    # Get number of samples
    n_samples = X.shape[0]

    # Set the seed for the random number generator
    np.random.seed(random_state)

    # Shuffle the indices
    shuffled_indices = np.random.permutation(np.arange(n_samples))

    # Determine the size of the test set
    test_size = int(n_samples * test_size)

    # Split the indices into test and train
    test_indices = shuffled_indices[:test_size]
    train_indices = shuffled_indices[test_size:]

    # Split the features and target arrays into test and train
    X_train, X_test = X[train_indices], X[test_indices]
    y_train, y_test = y[train_indices], y[test_indices]

    return X_train, X_test, y_train, y_test
```

```python
X = iris.iloc[:, :-1].values
y = iris.iloc[:, -1].values.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=41)
```

```python
from sklearn.tree import DecisionTreeClassifier
m = DecisionTreeClassifier()
```

```python
class RandomForest:
    """
    A random forest classifier.

    Parameters
    ----------
    n_trees : int, default=7
        The number of trees in the random forest.
    max_depth : int, default=7
        The maximum depth of each decision tree in the random forest.
    min_samples : int, default=2
        The minimum number of samples required to split an internal node
        of each decision tree in the random forest.

    Attributes
    ----------
    n_trees : int
        The number of trees in the random forest.
    max_depth : int
        The maximum depth of each decision tree in the random forest.
    min_samples : int
        The minimum number of samples required to split an internal node
        of each decision tree in the random forest.
    trees : list of DecisionTreeClassifier
        The decision trees in the random forest.
    """

    def __init__(self, n_trees=7, max_depth=7, min_samples=2):
        """
        Initialize the random forest classifier.

        Parameters
        ----------
        n_trees : int, default=7
            The number of trees in the random forest.
        max_depth : int, default=7
            The maximum depth of each decision tree in the random forest.
        min_samples : int, default=2
            The minimum number of samples required to split an internal node
            of each decision tree in the random forest.
        """
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.min_samples = min_samples
        self.trees = []

    def fit(self, X, y):
        """
        Build a random forest classifier from the training set (X, y).

        Parameters
        ----------
        X : array-like of shape (n_samples, n_features)
            The training input samples.
        y : array-like of shape (n_samples,)
            The target values.

        Returns
        -------
```

```
        Returns
        -------
        self : object
            Returns self.
        """
        # Create an empty list to store the trees.
        self.trees = []
        # Concatenate X and y into a single dataset.
        dataset = np.concatenate((X, y.reshape(-1, 1)), axis=1)
        # Loop over the number of trees.
        for _ in range(self.n_trees):
            # Create a decision tree instance.
            tree = DecisionTreeClassifier(max_depth=self.max_depth, min_samples_split=self.min_samples)
            # Sample from the dataset with replacement (bootstrapping).
            dataset_sample = self.bootstrap_samples(dataset)
            # Get the X and y samples from the dataset sample.
            X_sample, y_sample = dataset_sample[:, :-1], dataset_sample[:, -1]
            # Fit the tree to the X and y samples.
            tree.fit(X_sample, y_sample)
            # Store the tree in the list of trees.
            self.trees.append(tree)
        return self

    def bootstrap_samples(self, dataset):
        """
        Bootstrap the dataset by sampling from it with replacement.

        Parameters
        ----------
        dataset : array-like of shape (n_samples, n_features + 1)
            The dataset to bootstrap.

        Returns
        -------
        dataset_sample : array-like of shape (n_samples, n_features + 1)
            The bootstrapped dataset sample.
        """
        # Get the number of samples in the dataset.
        n_samples = dataset.shape[0]
        # Generate random indices to index into the dataset with replacement.
        np.random.seed(1)
        indices = np.random.choice(n_samples, n_samples, replace=True)
        # Return the bootstrapped dataset sample using the generated indices.
        dataset_sample = dataset[indices]
        return dataset_sample

    def most_common_label(self, y):
        """
        Return the most common label in an array of labels.

        Parameters
        ----------
        y : array-like of shape (n_samples,)
            The array of labels.

        Returns
        -------
        most_occuring_value : int or float
            The most common label in the array.
        """
        y = list(y)
        # get the highest present class in the array
        most_occuring_value = max(y, key=y.count)
        return most_occuring_value

    def predict(self, X):
        """
        Predict class for X.
```

```
        Parameters
        ----------
        X : array-like of shape (n_samples, n_features)
            The input samples.

        Returns
        -------
        majority_predictions : array-like of shape (n_samples,)
            The predicted classes.
        """
        #get prediction from each tree in the tree list on the test data
        predictions = np.array([tree.predict(X) for tree in self.trees])
        # get prediction for the same sample from all trees for each sample in the test data
        preds = np.swapaxes(predictions, 0, 1)
        #get the most voted value by the trees and store it in the final predictions array
        majority_predictions = np.array([self.most_common_label(pred) for pred in preds])
        return majority_predictions
```

[21] Python

```
def accuracy(y_true, y_pred):
    """
    Computes the accuracy of a classification model.

    Parameters:
    y_true (numpy array): A numpy array of true labels for each data point.
    y_pred (numpy array): A numpy array of predicted labels for each data point.

    Returns:
    float: The accuracy of the model, expressed as a percentage.
    """
    y_true = y_true.flatten()
    total_samples = len(y_true)
    correct_predictions = np.sum(y_true == y_pred)
    return (correct_predictions / total_samples)
```

[23] Python

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

[24] Python

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:116: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().

/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:134: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
```

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train.ravel())
y_test_encoded = label_encoder.transform(y_test.ravel())
model = RandomForest(20, 10, 2)
model.fit(X_train, y_train_encoded)

predictions = model.predict(X_test)
accuracy(y_test_encoded, predictions)
```

[31] Python

```
0.9333333333333333
```

```
from sklearn.tree import DecisionTreeClassifier

# Create and train the decision tree model
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train_encoded)

# Make predictions on the test data
predictions = dt.predict(X_test)

# Calculate accuracy
accuracy(y_test_encoded, predictions)
```

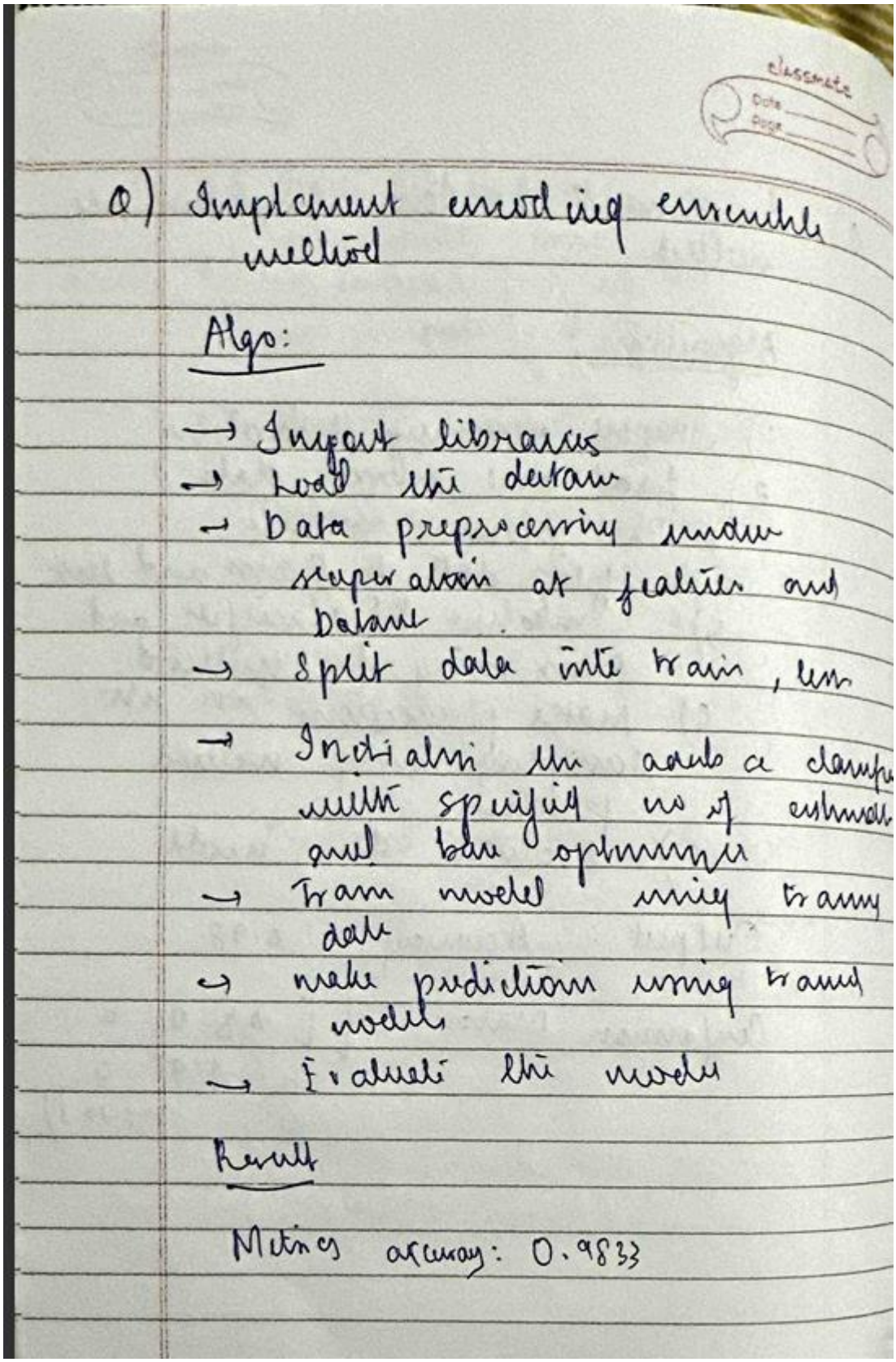
[32] Python

```
0.9
```

31.05.2024

Implement Boosting ensemble method on a given dataset.

Algorithm:



a) Implement ensembling ensemble method

Algo:

→ Import libraries
→ Load the dataset
→ Data preprocessing include separation of features and Dataset
→ Split data into train, test
→ Initialize the model a classifier with specific no of estimators and base optimizer
→ Train model using train data
→ make predictions using trained model
→ Evaluate the model

Result

Metrics accuracy: 0.9833

```python
# Compute error rate, alpha and w
def compute_error(y, y_pred, w_i):
    '''
    Calculate the error rate of a weak classifier m. Arguments:
    y: actual target value
    y_pred: predicted value by weak classifier
    w_i: individual weights for each observation

    Note that all arrays should be the same length
    '''
    return (sum(w_i * (np.not_equal(y, y_pred)).astype(int)))/sum(w_i)

def compute_alpha(error):
    '''
    Calculate the weight of a weak classifier m in the majority vote of the final classifier. This is called
    alpha in chapter 10.1 of The Elements of Statistical Learning. Arguments:
    error: error rate from weak classifier m
    '''
    return np.log((1 - error) / error)

def update_weights(w_i, alpha, y, y_pred):
    '''
    Update individual weights w_i after a boosting iteration. Arguments:
    w_i: individual weights for each observation
    y: actual target value
    y_pred: predicted value by weak classifier
    alpha: weight of weak classifier used to estimate y_pred
    '''
    return w_i * np.exp(alpha * (np.not_equal(y, y_pred)).astype(int))
```

```python
# Define AdaBoost class
class AdaBoost:

    def __init__(self):
        self.alphas = []
        self.G_m = []
        self.M = None
        self.training_errors = []
        self.prediction_errors = []

    def fit(self, X, y, M = 100):
        '''
        Fit model. Arguments:
        X: independent variables - array-like matrix
        y: target variable - array-like vector
        M: number of boosting rounds. Default is 100 - integer
        '''

        # Clear before calling
        self.alphas = []
        self.training_errors = []
        self.M = M

        # Iterate over M weak classifiers
        for m in range(0, M):

            # Set weights for current boosting iteration
            if m == 0:
                w_i = np.ones(len(y)) * 1 / len(y)  # At m = 0, weights are all the same and equal to 1 / N
            else:
                # (d) Update w_i
                w_i = update_weights(w_i, alpha_m, y, y_pred)
```

```python
            # Set weights for current boosting iteration
            if m == 0:
                w_i = np.ones(len(y)) * 1 / len(y)  # At m = 0, weights are all the same and equal to 1 / N
            else:
                # (d) Update w_i
                w_i = update_weights(w_i, alpha_m, y, y_pred)

            # (a) Fit weak classifier and predict labels
            G_m = DecisionTreeClassifier(max_depth = 1)     # Stump: Two terminal-node classification tree
            G_m.fit(X, y, sample_weight = w_i)
            y_pred = G_m.predict(X)

            self.G_m.append(G_m) # Save to list of weak classifiers

            # (b) Compute error
            error_m = compute_error(y, y_pred, w_i)
            self.training_errors.append(error_m)

            # (c) Compute alpha
            alpha_m = compute_alpha(error_m)
            self.alphas.append(alpha_m)

        assert len(self.G_m) == len(self.alphas)

    def predict(self, X):
        '''
        Predict using fitted model. Arguments:
        X: independent variables - array-like
        '''

        # Initialise dataframe with weak predictions for each observation
        weak_preds = pd.DataFrame(index = range(len(X)), columns = range(self.M))

        # Predict class label for each weak classifier, weighted by alpha_m
        for m in range(self.M):
            y_pred_m = self.G_m[m].predict(X) * self.alphas[m]
            weak_preds.iloc[:,m] = y_pred_m

        # Calculate final predictions
        y_pred = (1 * np.sign(weak_preds.T.sum())).astype(int)

        return y_pred
```

```python

```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

# Dataset
df = pd.read_csv('/content/spambase.data', header = None)

# Column names
names = pd.read_csv('/content/spambase.names', sep = ':', skiprows=range(0, 33), header = None)
col_names = list(names[0])
col_names.append('Spam')

# Rename df columns
df.columns = col_names
```

```python
# Column names
names = pd.read_csv('/content/spambase.names', sep = ':', skiprows=range(0, 33), header = None)
col_names = list(names[0])
col_names.append('Spam')

# Rename df columns
df.columns = col_names

# Convert classes in target variable to [-1, 1]
df['Spam'] = df['Spam'] * 2 - 1

# Train - test split
X_train, X_test, y_train, y_test = train_test_split(df.drop(columns = 'Spam').values,
                                                    df['Spam'].values,
                                                    train_size = 3065,
                                                    random_state = 2)
```

```python

# Fit model
ab = AdaBoost()
ab.fit(X_train, y_train, M = 400)

# Predict on test set
y_pred = ab.predict(X_test)

from sklearn.metrics import accuracy_score

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy', accuracy)
```

```
Accuracy 0.9449304174950666
```