VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Prabhanjan Bhat (1BM22CS196)

in partial fulfillment 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
Sep-2024 to Jan-2025

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 (23CS6PCMAL)" carried out by **Prabhanjan Bhat(1BM22CS196),** who is a bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of a Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab Faculty In-Charge Name: **Ms. Geetha N** Assistant Professor Department of CSE, BMSCE	 **Dr. Kavitha Sooda,** Professor & HOD, Department of CSE, BMSCE

Index

Github Link: https://github.com/prabhanjanbhat/MAL

Program 1

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

```
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from sklearn.datasets import load diabetes

# ------------------ Method-1: Directly initialize DataFrame
------------------------------------------------------------------------------------------------------------------------------------------------------
data_method1 = {
    'USN': ['1JS17CS001', '1JS17CS002', '1JS17CS003', '1JS17CS004',
'1JS17CS005'],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Marks': [90, 85, 92, 78, 88]
}
df_method1 = pd.DataFrame(data_method1)
print("Method-1:")
print(df_method1)
print("-" * 40)

# ------------------ Method-2: Load sample dataset from sklearn
------------------------------------------------------------------------------------------------------------------------------------------------------
diabetes_data = load_diabetes()
df_method2 = pd.DataFrame(data=diabetes_data.data,
columns=diabetes_data.feature_names)
df_method2['target'] = diabetes_data.target

print("Method-2:")
print(df_method2.head())
print("-" * 40)

# ------------------------- Method-3: Load from CSV file -------------------------
try:
    df_method3 = pd.read_csv('sample_sales_data.csv')
    print("Method-3:")
    print(df_method3.head())
except FileNotFoundError:
    print("sample sales data.csv not found. Please upload the file.")
print("-" * 40)
```

```python
# ------------------ Stock Data Analysis using yfinance ------------------
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
start_date = "2024-01-01"
end_date = "2024-12-30"

# Download historical stock data
stock_data = yf.download(tickers, start=start_date, end=end_date)

# Extract closing prices and calculate daily returns
closing_prices = stock_data['Close']
daily_returns = closing_prices.pct_change().dropna()

# Plot closing prices
plt.figure(figsize=(12, 6))
closing_prices.plot(title='Closing Prices (2024)')
plt.xlabel('Date')
plt.ylabel('Price (INR)')
plt.grid(True)
plt.tight_layout()
plt.show()

# Plot daily returns
plt.figure(figsize=(12, 6))
daily_returns.plot(title='Daily Returns (2024)')
plt.xlabel('Date')
plt.ylabel('Daily Return')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot

Lab-1

```
import pandas as pd

data = { 'USN' : ['1BM21CS001', '1BM21CS002', '1BM21CS003',
                  '1BM21CS004', '1BM21CS005'],
         'Name' : ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
         'Marks' : [85, 90, 78, 88, 92]
}

df = pd.DataFrame(data)
print ("Sample data :")
print(df.head())
```

Output:

	USN	Name	Marks
0	1BM21CS001	Alice	85
1	1BM21CS002	Bob	90
2	1BM21CS003	Charlie	78
3	1BM21CS004	David	88
4	1BM21CS005	Eve	92

```
from sklearn.datasets import load_diabetes

diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns = diabetes.feature_names)
df ['target'] = diabetes.target

print ("Sample data:")
print(df.head())
```

To do

Using the code given in the above slides, do the exercise of the "Stock Market Data Analysis", considering the following

1. HDFC Bank Ltd , ICIC Bank Ltd , Kotak Mahindra Bank Ltd.
tickers = ["HDFCBANK.NS", "ICICBANK.NS", "KOTAKBANK.NS"]

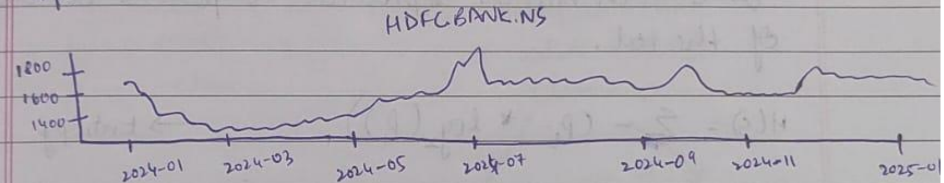
2. Start date : 2024-01-01 , End data : 2024-12-30

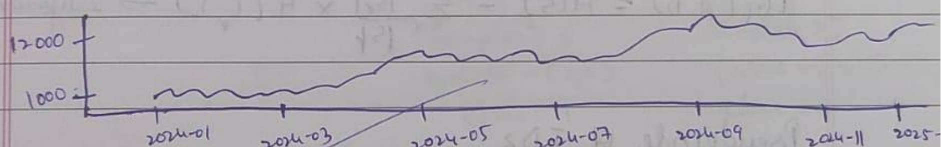
3. Plot the closing price and daily returns for all the 3 banks mentioned

```
import yfinance as yf
Import pandas as pd
import matplotlib.pyplot as plt
  //Define ticker symbols
tickers = ["HDFCBANK.NS", "ICICBANK.NS", "KOTAKBANK.NS"]

data = yf.download(tickers, start = "2024-01-01", end = "2024-12-30",
         group_by = 'ticker')

for ticker in tickers:
    data.loc[:, (ticker , 'Daily Return')] = data[ticker]['Close'] pct_change()

plt.figure(figsize =(12,6))
for i, ticker in enumerate(tickers):
        plt.subplot(3,1, i+1)
        data[ticker]['Close'].plot(title = f"{ticker} - Closing Price")

plt.tight_layout()
plt.show()

plt.figure(figsize =(12,6))
```

```
for i, ticker in enumerate(tickers):
        plt.subplot(3, 1, i+1)
        data[ticker]['Daily Return'].plot(title = f"{ticker} - Daily
                                          Returns", color = 'orange')
plt.tight_layout()
plt.show()
```

Output:





Code:

```
#Set the values to some value (zero, the mean, the median, etc.).
```

```python
# Step 1: Create an instance of SimpleImputer with the median strategy for Age
and mean stratergy for Salary
imputer1 = SimpleImputer(strategy="median")
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column
# Note: SimpleImputer expects a 2D array, so we reshape the column
imputer1.fit(df_copy[["Age"]])
imputer2.fit(df_copy[["Salary"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column
df_copy["Age"] = imputer1.transform(df[["Age"]])
df_copy["Salary"] = imputer2.transform(df[["Salary"]])

# Verify that there are no missing values left
print(df_copy["Age"].isnull().sum())
print(df_copy["Salary"].isnull().sum())
```

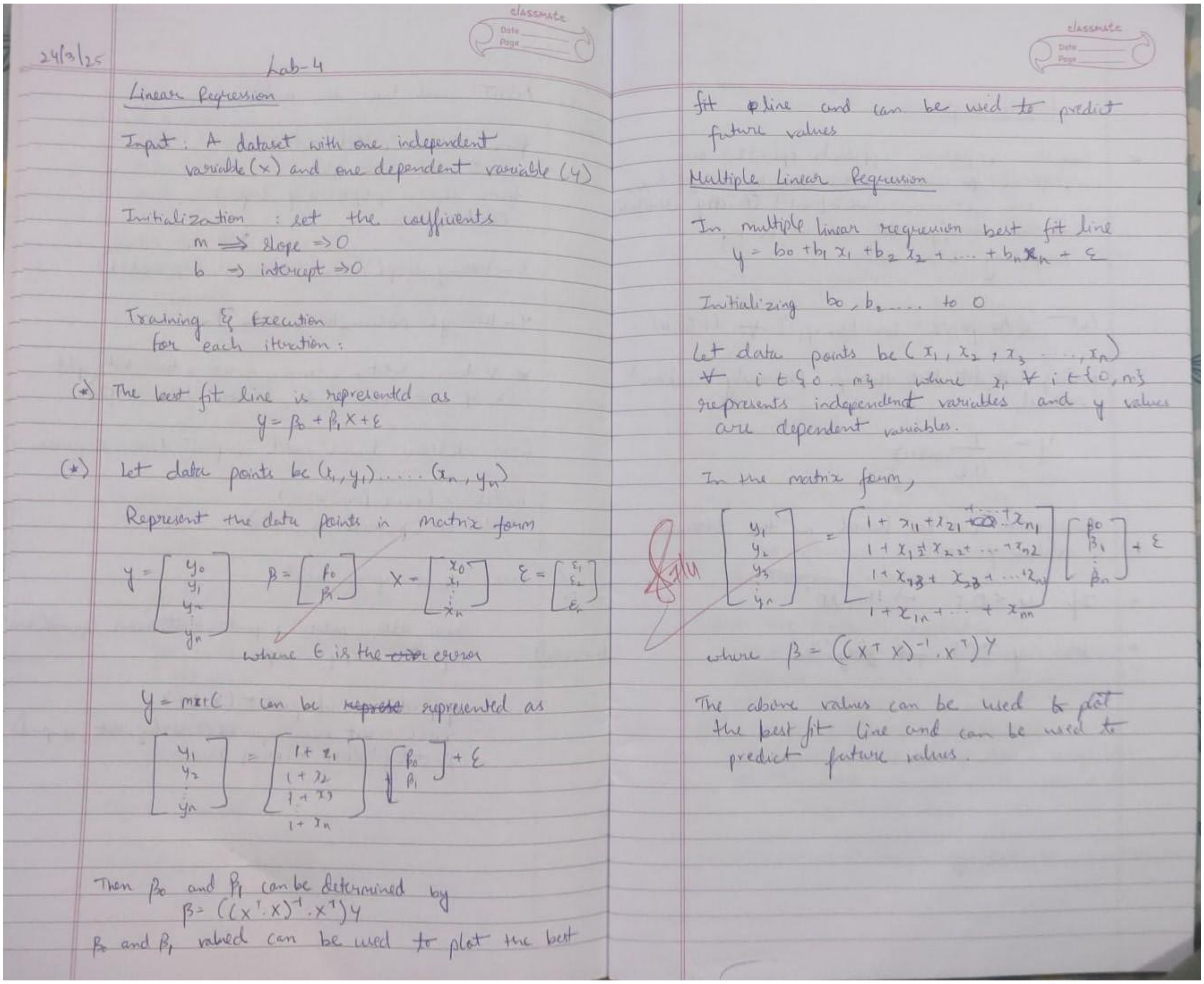
```python
#Handling Categorical Attributes
#Using Ordinal Encoding for gender COlumn and One-Hot Encoding for City Column

# Initialize OrdinalEncoder
ordinal_encoder = OrdinalEncoder(categories=[["Male", "Female"]])
# Fit and transform the data
df_copy["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy[["Gender"]])

# Initialize OneHotEncoder
onehot_encoder = OneHotEncoder()

# Fit and transform the "City" column
encoded_data = onehot_encoder.fit_transform(df[["City"]])

# Convert the sparse matrix to a dense array
encoded_array = encoded_data.toarray()

# Convert to DataFrame for better visualization
encoded_df                    =                    pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["City"]))
df_encoded = pd.concat([df_copy, encoded_df], axis=1)
```

```
df_encoded.drop("Gender", axis=1, inplace=True)
df_encoded.drop("City", axis=1, inplace=True)

print(df_encoded. head())
#Removing Outliers
# Outlier Detection and Treatment using IQR

df_encoded_copy1=df_encoded
df_encoded_copy2=df_encoded
df_encoded_copy3=df_encoded

Q1 = df_encoded_copy1['Salary'].quantile(0.25)
Q3 = df_encoded_copy1['Salary'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_encoded_copy1['Salary'] = np.where(df_encoded_copy1['Salary'] > upper_bound,
upper_bound,
                             np.where(df_encoded_copy1['Salary'] < lower_bound,
lower_bound, df_encoded_copy1['Salary']))

print(df_encoded_copy1.head())


#Removing Outliers
# Z-score method


df_encoded_copy2['Salary_zscore'] = stats.zscore(df_encoded_copy2['Salary'])
df_encoded_copy2['Salary'] = np.where(df_encoded_copy2['Salary_zscore'].abs() >
3, np.nan, df_encoded_copy2['Salary'])  # Replace outliers with NaN
print(df_encoded_copy2.head())
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:



classmate

24/3/25

Lab-4

Linear Regression

Input: A dataset with one independent variable (x) and one dependent variable (y)

Initialization : set the coefficients
$m \Rightarrow$ slope $\Rightarrow 0$
$b \rightarrow$ intercept $\Rightarrow 0$

Training & Execution
for each iteration:

(*) The best fit line is represented as

$$y = \beta_0 + \beta_1 x + \varepsilon$$

(*) Let data points be $(x_1, y_1) \ldots (x_n, y_n)$

Represent the data points in matrix form

$$y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \quad X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_n \end{bmatrix}$$

where ϵ is the ~~err~~ error

$y = mx + c$ can be ~~represe~~ represented as

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 + x_1 \\ 1 + x_2 \\ 1 + x_3 \\ \vdots \\ 1 + x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \varepsilon$$

Then β_0 and β_1 can be determined by

$$\beta = ((x^T \cdot x)^{-1} \cdot x^T) y$$

β_0 and β_1 valued can be used to plot the best fit line and can be used to predict future values

Multiple Linear Regression

In multiple linear regression best fit line

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n + \varepsilon$$

Initializing $b_0, b_1 \ldots$ to 0

Let data points be $(x_1, x_2, x_3 \ldots, x_n)$ $\forall\ i \in \{0 \ldots m\}$ where x_i $\forall\ i \in \{0, n\}$ represents independent variables and y values are dependent variables.

In the matrix form,

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 + x_{11} + x_{21} + \ldots + x_{n1} \\ 1 + x_{12} + x_{22} + \ldots + x_{n2} \\ 1 + x_{13} + x_{23} + \ldots + x_{n3} \\ \vdots \\ 1 + x_{1n} + \ldots + x_{nn} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} + \varepsilon$$

where $\beta = ((X^T X)^{-1} \cdot X^T) Y$

The above values can be used to plot the best fit line and can be used to predict future values.

Code:

```
#canada_per_capita_income.csv
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv('/content/canada_per_capita_income.csv')
print(df.head())
print(df.isnull().sum()
)
df.dropna(inplace=True)


plt.xlabel('Year')
plt.ylabel('Income')
plt.scatter(df.year, df.Income, color='red', marker='+')

# Prepare input feature
X = df[['year']]  # Independent variable
y = df['Income']  # Dependent variable (what we want to predict)

# Train linear regression model
reg = linear_model.LinearRegression()
reg.fit(X, y)

# Predict future incomes
year_2020 = reg.predict([[2020]])[0]
year_2027 = reg.predict([[2027]])[0]

print(f'Year=2020, Predicted Income=${year_2020:,.2f}')
print(f'Year=2027, Predicted Income=${year_2027:,.2f}')
print("------------------------------------------------------------------------
------------------------------")
print(f'Coefficient (m): {reg.coef_[0]:,.2f}')
print("------------------------------------------------------------------------
------------------------------")
print(f'Intercept (b): {reg.intercept_:,.2f}')
print("------------------------------------------------------------------------
------------------------------")

# Plot regression line
```

```
plt.plot(df.year, reg.predict(X), color='blue')
plt.show()
```

```
#salary.csv

import pandas as pd

import numpy as np

from sklearn import linear_model

import matplotlib.pyplot as plt



df = pd.read_csv('/content/salary.csv')

print(df.head())



print(df.isnull().sum())

df.fillna(df['YearsExperience'].mean(),inplace=True)

print(df.isnull().sum())

print(" ------------------------------------------------------------")



plt.xlabel('YearExperience')

plt.ylabel('Salary')

plt.scatter(df.YearsExperience, df.Salary, color='red', marker='+')



# Prepare input feature

X = df[['YearsExperience']]  # Independent variable
```

```python
y = df['Salary']  # Dependent variable


# Train linear regression model

reg = linear_model.LinearRegression()

reg.fit(X, y)


# Predict future incomes

year12= reg.predict([[12]])[0]

# year_2027 = reg.predict([[2027]])[0]

print(f'YearsExperience=12, Predicted Income=${year12:,.2f}')


print(f'Coefficient (m): {reg.coef_[0]:,.2f}')

print("------------------------------------------------------------------------
-------------------")

print(f'Intercept (b): {reg.intercept_:,.2f}')

print("------------------------------------------------------------------------
-------------------")


# Plot regression line

plt.plot(df.YearsExperience, reg.predict(X), color='blue')

plt.show()

#homeprices_Multiple_LR.csv

import pandas as pd
```

```
import numpy as np
from sklearn import linear_model


# Load dataset
df = pd.read_csv('/content/homeprices_Multiple_LR.csv')


# Handle missing values (Fill NA in 'bedrooms' with median)
df['bedrooms'] = df['bedrooms'].fillna(df['bedrooms'].median())


# Prepare training data
X = df.drop('price', axis='columns')  # Features: Area, Bedrooms, Age
y = df['price']  # Target: Price


# Train linear regression model
reg = linear_model.LinearRegression()
reg.fit(X, y)


# Display model coefficients
print(f'Coefficients: {reg.coef_}')
print(f'Intercept: {reg.intercept_}\n')


# Predict price of a home with given features
```

```
area = 3000

bedrooms = 3

age = 40


predicted_price = (

    reg.coef_[0] * area +

    reg.coef_[1] * bedrooms +

    reg.coef_[2] * age +

    reg.intercept_

)


print(f'Predicted price for a {area} sq. ft home, {bedrooms} bedrooms, {age} years old: ${predicted_price:,.2f}')

#Hiring

import pandas as pd

import numpy as np

from sklearn import linear_model


# Load dataset

df = pd.read_csv('/content/hiring.csv')

print(df.isnull().sum())

# Fill missing values

df['test_score(out of 10)'].fillna(df['test_score(out of 10)'].median(),
```

```
inplace=True)


# Convert experience to string (for one-hot encoding)

df['experience'] = df['experience'].astype(str)


# Apply one-hot encoding to 'experience'

df_encoded = pd.get_dummies(df, columns=['experience'], drop_first=True)


# Separate features and target variable

X = df_encoded.drop('salary($)', axis='columns')

y = df_encoded['salary($)']


# Train the regression model

reg = linear_model.LinearRegression()

reg.fit(X, y)


# Function to predict salary

def predict_salary(exp, test_score, interview_score):

    # Convert experience to one-hot encoding

    exp_col = f'experience_{exp}'

    input_data = {col: 0 for col in X.columns}  # Initialize all columns to 0

    if exp_col in input_data:
```

```
        input_data[exp_col] = 1  # Set the correct experience column

    input_data['test_score(out of 10)'] = test_score

    input_data['interview_score(out of 10)'] = interview_score


    # Convert to DataFrame and predict

    input_df = pd.DataFrame([input_data])

    predicted_salary = reg.predict(input_df)[0]


    return f'Predicted Salary: ${predicted_salary:,.2f}'


# Example Prediction

print(predict_salary('twelve', 10, 10))

print(predict_salary('two', 9, 6))

import pandas as pd

import numpy as np

from sklearn.linear_model import LinearRegression


# Load dataset

df = pd.read_csv('/content/1000_Companies.csv')


# One-hot encode the 'State' column

df_encoded = pd.get_dummies(df, columns=['State'], drop_first=True)
```

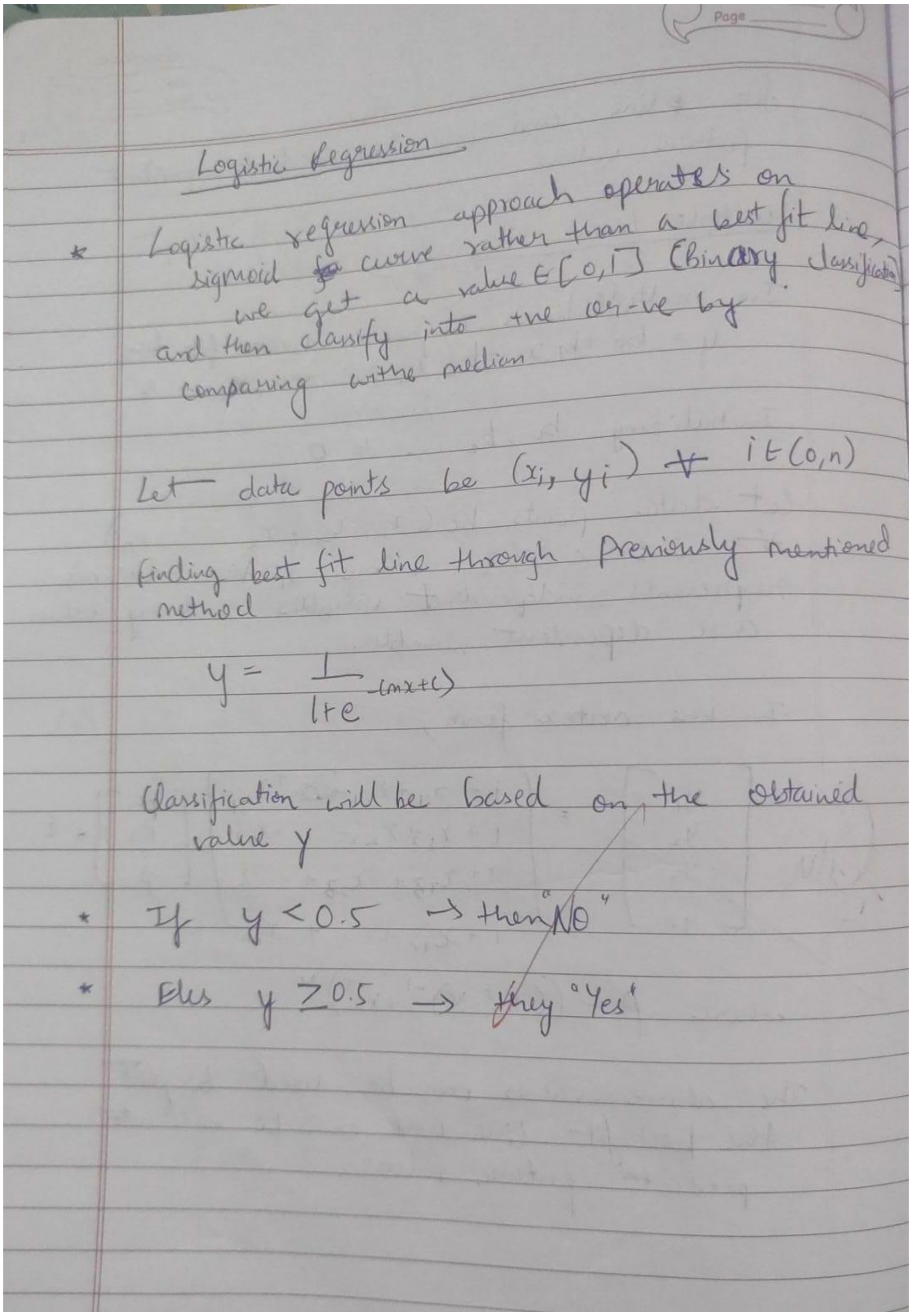
```python
# Separate features and target variable

X = df_encoded.drop('Profit', axis='columns')  # Features: R&D Spend,
Administration, Marketing Spend, State

y = df_encoded['Profit']  # Target: Profit


# Train linear regression model

reg = LinearRegression()

reg.fit(X, y)


# Function to predict profit

def predict_profit(rnd_spend, admin_spend, marketing_spend, state):

    # Initialize input data dictionary with all zeros

    input_data = {col: 0 for col in X.columns}


    # Assign provided values

    input_data['R&D Spend'] = rnd_spend

    input_data['Administration'] = admin_spend

    input_data['Marketing Spend'] = marketing_spend


    # One-hot encode 'State'

    state_col = f'State_{state}'

    if state_col in input_data:
```

```
        input_data[state_col] = 1

    # Convert to DataFrame and predict
    input_df = pd.DataFrame([input_data])
    predicted_profit = reg.predict(input_df)[0]

    return f'Predicted Profit: ${predicted_profit:,.2f}'

# Example Prediction
print(predict_profit(91694.48, 515841.3, 11931.24, 'Florida'))
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:



Logistic Regression

* Logistic regression approach operates on sigmoid curve rather than a best fit line, we get a value $\in [0,1]$ (Binary classification) and then classify into the +ve or -ve by comparing with the median

Let data points be $(x_i, y_i)\ \forall\ i \in (0,n)$

finding best fit line through previously mentioned method

$$y = \frac{1}{1+e^{-(mx+c)}}$$

Classification will be based on the obtained value y

* If $y < 0.5$ → then "NO"
* Else $y \geq 0.5$ → they "Yes"

Code:

```
#INSURANCE DATA

 Commented out IPython magic to ensure Python compatibility.

import pandas as pd

from matplotlib import pyplot as plt

# %matplotlib inline

#"%matplotlib inline" will make your plot outputs appear and be stored within
the notebook.



df = pd.read_csv("/content/insurance_data.csv")

df.head()



plt.scatter(df.age,df.bought_insurance,marker='+',color='red')



from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(df[['age']],df.bought_insurance,train_size=0.9,random_state=10
)

X_train.shape



X_test



from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
```

```
model.fit(X_train, y_train)

X_test

y_test

y_predicted = model.predict(X_test)

y_predicted

model.score(X_test,y_test)

model.predict_proba(X_test)

y_predicted = model.predict([[60]])

y_predicted

#model.coef_ indicates value of m in y=m*x + b equation

model.coef_

#model.intercept_ indicates value of b in y=m*x + b equation

model.intercept_

#Lets defined sigmoid function now and do the math with hand

import math

def sigmoid(x):

  return 1 / (1 + math.exp(-x))



def prediction_function(age):

   z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97

   y = sigmoid(z)
```

```
    return y


age = 35

prediction_function(age)


"""0.37 is less than 0.5 which means person with 35 will not buy the
insurance"""

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

# Load the dataset

df = pd.read_csv("/content/HR_comma_sep.csv")

# Display basic information about the dataset

df.info()

print(df.head())

# Exploratory Data Analysis

plt.figure(figsize=(8, 6))

sns.countplot(x='salary', hue='left', data=df)

plt.title("Impact of Salary on Employee Retention")

plt.show()
```

```
plt.figure(figsize=(10, 6))

sns.countplot(x='Department', hue='left', data=df)

plt.xticks(rotation=45)

plt.title("Correlation Between Department and Employee Retention")

plt.show()

# Selecting relevant features

X = df[['satisfaction_level', 'last_evaluation', 'number_project',
'average_montly_hours', 'time_spend_company', 'Work_accident',
'promotion_last_5years']]

y = df['left']

# Splitting data

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.9,
random_state=10)

# Building Logistic Regression Model

model = LogisticRegression()

model.fit(X_train, y_train)

y_predicted = model.predict(X_test)

accuracy = accuracy_score(y_test, y_predicted)

print(f"Model Accuracy: {accuracy:.2f}")

# Model Coefficients

print("Model Coefficients:", model.coef_)

print("Model Intercept:", model.intercept_)

# -*- coding: utf-8 -*-

"""LogisticRegression_Multiclass.ipynb
```

```
Automatically generated by Colab.

# Import necessary libraries

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

from sklearn import metrics

import matplotlib.pyplot as plt

# Load the Iris dataset

iris =  pd.read_csv("/content/iris.csv")

iris.head()

X=iris.drop('species',axis='columns')# Features (sepal length, sepal width,
petal length, petal width)

y = iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)



# Split the dataset into 80% training and 20% testing

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



# Initialize the Multinomial Logistic Regression model

# Use 'multinomial' for multi-class classification and 'lbfgs' solver

model = LogisticRegression(multi_class='multinomial')
```

```
# Train the model on the training data

model.fit(X_train, y_train)



# Make predictions on the test data

y_pred = model.predict(X_test)



# Calculate the accuracy of the model on the test data

accuracy = accuracy_score(y_test, y_pred)



# Display the accuracy

print(f"Accuracy of the Multinomial Logistic Regression model on the test set:
{accuracy:.2f}")

confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
confusion_matrix, display_labels = ["Setosa", "Versicolor", "Virginica"])

cm_display.plot()

plt.show()

#Implementation - Logistic Regression (Multiclass Classification)

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay

# Load datasets

zoo_data_path = "/content/zoo-data.csv"

class_type_path = '/content/zoo-class-type.csv'

df = pd.read_csv(zoo_data_path)

class_df = pd.read_csv(class_type_path)

print(df.info())

print(df.describe())

print(df.isnull().sum())

print("-------------------------------------------------------------------------
-------")

print(class_df.info())

print(class_df.describe())

print(class_df.isnull().sum())

print("-------------------------------------------------------------------------
-------")

# Merge datasets on class_type

if 'class_type' in df.columns and 'class_type' in class_df.columns:

    df = df.merge(class_df, on='class_type', how='left')

# Drop unnecessary columns (if any)

if 'animal_name' in df.columns:

    df.drop(columns=['animal_name'], inplace=True)
```

```
# Separate features and target

X = df.drop(columns=['class_type'])

y = df['class_type']

# Split data into training and testing sets

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

# Build logistic regression model

model = LogisticRegression(max_iter=200)

model.fit(X_train, y_train)



# Predict and evaluate accuracy

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.4f}")



# Plot confusion matrix

cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

disp.plot(cmap='Blues')

plt.title("Confusion Matrix")

plt.show()
```

Program 5

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample

Screenshot:

classmate
Date 10/3/25
Page

ID 3 algorithm.

ID3 algorithm is a popular decision tree algorithm used in machine learning. It aims to build decision tree by iteratively selecting the best attribute to split the data based on information gain.

Each node represents a test on an attribute and each branch represents a possible outcome of the test.

$H(S) = \sum - (P_i * \log_2(P_i))$ → Entropy

$IG(A,D) = H(S) - \sum_v \frac{|S_v|}{|S|} \times H(S_v)]$ → Information Gain

Pseudocode of ID3

```
def ID3(D,A):
    if D is pure or A is empty:
        return a leaf node with the majority class in D
    else:
        A_best = argmax( InformationGain(D, A))
        root = Node(A_best)
        for v in values(A_best):
            Dv = subset(D, A_best, v)
            child = ID3(Dv , A - {A_best})
            root.add_child (v, child)

    return root
```

Outlook <= 0.5
entropy = 0.94
samples = 14
value = [5,9]
class = 1

True

entropy = 0.0
samples = 4
value = [0,4]
class = 1

False

Outlook <= 1.5
entropy = 1.0
samples = 10
value = [5,5]
class = 0

entropy = 0.971
samples = 5
value = [2,3]
class = 1

entropy = 0.971
samples = 5
value = [3,2]
class = 0

10/3/25

Code:

```
import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, classification_report


# Create the dataset

data =pd.read_csv("/content/iris.csv")


data
```

```
# Convert to DataFrame
df = pd.DataFrame(data)

# Convert categorical data to numerical data
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Split the dataset into features and target
X = df.drop('species', axis=1)
y = df['species']

# Split the data into training and testing sets
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,  test_size=0.2,
random_state=42)

# Initialize the Decision Tree Classifier with entropy as the criterion
clf = DecisionTreeClassifier(criterion='entropy')

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test,      y_pred,     target_names=['Iris-setosa',
```

```
'Iris-versicolor','Iris-virginica']))

# Optionally, visualize the decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
plot_tree(clf,               filled=True,               feature_names=X.columns,
class_names=['Iris-setosa', 'Iris-versicolor','Iris-virginica'])
plt.show()

import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Create the dataset
data =pd.read_csv("/content/drug.csv")

data

# Convert to DataFrame
df = pd.DataFrame(data)

# Convert categorical data to numerical data
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
```

```
# Split the dataset into features and target
X = df.drop('Drug', axis=1)
y = df['Drug']

# Split the data into training and testing sets
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,  test_size=0.2,
random_state=42)

# Initialize the Decision Tree Classifier with entropy as the criterion
clf = DecisionTreeClassifier(criterion='entropy')

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test,        y_pred,        target_names=['drugA',
'drugB','drugC','drugX','drugY']))

# Optionally, visualize the decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
plot_tree(clf,   filled=True,   feature_names=X.columns,   class_names=['drugA',
'drugB','drugC','drugX','drugY'])
plt.show()
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load dataset
df = pd.read_csv("/content/petrol_consumption.csv")

# Separate features (X) and target (y)
X = df.drop(columns=['Petrol_Consumption'])  # Assuming 'Petrol_Consumption' is the target variable
y = df['Petrol_Consumption']

# Split the dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)

# Make predictions
y_pred = regressor.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
```

```python
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

# Optionally, visualize the decision tree
from sklearn.tree import plot_tree

plt.figure(figsize=(12, 8))
plot_tree(regressor, filled=True, feature_names=X.columns)
plt.show()
```

Program 6

Build KNN Classification model for a given dataset

Screenshot:

7/4/24 Lab- 5

KNN algorithm

1. Choose the number of neighbors(k)
   set k = 3

2. Calculate the distance from the test point to all training points
   → Use the Euclid's formula

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

$$d_1 = \sqrt{(x_T - x_1)^2 + (y_T - y_1)^2}$$

$$d_2 = \sqrt{(x_T - x_2)^2 + (y_T - y_2)^2}$$

$$\vdots$$

$$d_n = \sqrt{(x_T - x_n)^2 + (y_T - y_n)^2}$$

3. Sort the distances in ascending order of the Euclidean's distance

4. Select the k nearest neighbors
   → Pick the top k points with the smallest distances

5. Do a majority vote (for classification)
   → Count the labels of the k neighbors
   → The label with the most votes is the predicted class

For regression: Average of the k nearest neighbors

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load datasets
iris_df = pd.read_csv('/content/iris (1).csv')
diabetes_df = pd.read_csv('/content/diabetes.csv')

print("=== IRIS DATASET ===")

# Split features and target
X_iris = iris_df.drop('species', axis=1)
y_iris = iris_df['species']

# Train-test split
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)

# Choose best k
k_range = range(1, 21)
accuracies = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_iris, y_train_iris)
    accuracies.append(knn.score(X_test_iris, y_test_iris))

optimal_k_iris = k_range[accuracies.index(max(accuracies))]
```

```python
print(f"Optimal K for Iris: {optimal_k_iris}")

# Train and evaluate
knn_iris = KNeighborsClassifier(n_neighbors=optimal_k_iris)
knn_iris.fit(X_train_iris, y_train_iris)
y_pred_iris = knn_iris.predict(X_test_iris)

print("Accuracy:", accuracy_score(y_test_iris, y_pred_iris))
print("Confusion Matrix:\n", confusion_matrix(y_test_iris, y_pred_iris))
print("Classification Report:\n", classification_report(y_test_iris,
y_pred_iris))

# Optional: Plot accuracy vs. k for Iris
plt.figure(figsize=(8, 4))
plt.plot(k_range, accuracies, marker='o')
plt.title('Iris Dataset - Accuracy vs K')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()

### ---------- DIABETES DATASET -------------------- ###
print("\n=== DIABETES DATASET ===")

# Split features and target
X_diabetes = diabetes_df.drop('Outcome', axis=1)
y_diabetes = diabetes_df['Outcome']

# Feature scaling
scaler = StandardScaler()
X_diabetes_scaled = scaler.fit_transform(X_diabetes)
```

```python
# Train-test split
X_train_diab, X_test_diab, y_train_diab, y_test_diab = train_test_split(X_diabetes_scaled, y_diabetes, test_size=0.2, random_state=42)

# Choose best k
k_range_diab = range(1, 21)
accuracies_diab = []
for k in k_range_diab:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_diab, y_train_diab)
    accuracies_diab.append(knn.score(X_test_diab, y_test_diab))

optimal_k_diab = k_range_diab[accuracies_diab.index(max(accuracies_diab))]
print(f"Optimal K for Diabetes: {optimal_k_diab}")

# Train and evaluate
knn_diab = KNeighborsClassifier(n_neighbors=optimal_k_diab)
knn_diab.fit(X_train_diab, y_train_diab)
y_pred_diab = knn_diab.predict(X_test_diab)

print("Accuracy:", accuracy_score(y_test_diab, y_pred_diab))
print("Confusion Matrix:\n", confusion_matrix(y_test_diab, y_pred_diab))

# Optional: Plot accuracy vs. k for Diabetes
plt.figure(figsize=(8, 4))
plt.plot(k_range_diab, accuracies_diab, marker='o', color='green')
plt.title('Diabetes Dataset - Accuracy vs K')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
heart_df = pd.read_csv("heart.csv")

# Features and target
X = heart_df.drop('target', axis=1)
y = heart_df['target']

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Find optimal k
k_range = range(1, 21)
accuracies = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    acc = knn.score(X_test, y_test)
```

```
    accuracies.append(acc)
optimal_k = k_range[accuracies.index(max(accuracies))]
print(f"Optimal K value: {optimal_k}")
# Train with optimal k
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
# Classification report
report = classification_report(y_test, y_pred, output_dict=False)
print("Classification Report:\n", report)
# Optional: Accuracy vs. k plot
plt.figure(figsize=(8, 4))
plt.plot(k_range, accuracies, marker='o')
plt.title('Heart Dataset - Accuracy vs K')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```

Program 7

Build Support vector machine model for a given dataset

Screenshot:

classmate
Date
Page

SVM algorithm

(1) Initialize : Dataset $D = \{(x1, y1), (x2, y2), \ldots, (x_n y_n)\}$
Regularization parameter C
~~Maximize numb~~
Maximum number of iterations max_iter

(1) Data Loading & Preprocessing
→ Load the Iris Dataset
→ Apply z-score normalization to standardize features

$x' = \sigma x - \mu$

(2) ~~B~~ Split the Dataset into
→ Training data set (70%)
→ Test data set (30%)

(3) Initialize the SVM Classifier
→ C : Regularization constant
→ max_iter
→ kernel : Linear

(4) One vs Rest Training Strategy

For each class C in the set of unique classes:
→ Convert labels into binary format

$$y = \begin{cases} 0 & \text{if } y = c \\ 1 & \text{if otherwise} \end{cases}$$

→ Train a binary SVM classifier using the simplified SMO algorithm

classmate
Date
Page

(5) Binary SVM Training
Initialize :
$\alpha = 0$: Lagrange multiplier
$b = 0$: Bias term

Repeat for max_iteration:
For each training sample i:
1. Randomly select another index $j = i$
2. Compute prediction errors E_i & E_j
3. Save old values α_i, α_j
4. compute bounds L, H
   if $L = H$, continue
5. Compute:
   $\eta = 2K(x_i, x_j) - K(x_i, x_i) - K(x_j, x_j)$
   if $\eta \geq 0$, skip update
6. Update α_j:
   $\alpha_j = \alpha_j + \eta y_j (E_i - E_j)$
7. Update α_i
8. ~~Compte~~ Compute
   $b_1 = b - E_i - y_i(\alpha_i - \alpha_i^{d})$
9. Update the bias term b:
   → If α_i in bounds: $b = b_1$
   → Else if α_j in bounds: $b = b_2$
   → Else : $b = 2b_1 + b_2$

(6) Prediction Phase
For each test sample x:
Compute decision score:
$f(x) = \sum \alpha_j y_j K(x, x_j) + b$

• Predict the class with maximum decision score

Date
Page

(7) Evaluate the accuracy with the unseen data.

21

Code:

```
# Load Letter dataset
letter_df = pd.read_csv("/content/letter-recognition.csv")

# Assuming the first column is label
X_letter = letter_df.iloc[:, 1:]
y_letter = letter_df.iloc[:, 0]

# Train-test split
X_train_letter, X_test_letter, y_train_letter, y_test_letter = train_test_split(X_letter, y_letter, test_size=0.2, random_state=42)

# SVM Classifier
model_letter = SVC(kernel='rbf', probability=True)
model_letter.fit(X_train_letter, y_train_letter)
y_pred_letter = model_letter.predict(X_test_letter)

print("\n□ Letter Dataset")
print("Accuracy:", accuracy_score(y_test_letter, y_pred_letter))
print("Confusion Matrix:\n", confusion_matrix(y_test_letter, y_pred_letter))

# ROC Curve and AUC Score
lb = LabelBinarizer()
y_test_bin = lb.fit_transform(y_test_letter)
y_score = model_letter.predict_proba(X_test_letter)

# Compute ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(len(lb.classes_)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = roc_auc_score(y_test_bin[:, i], y_score[:, i])
```

```python
# Plot ROC curve for first 3 classes
plt.figure(figsize=(10, 6))
for i in range(3):
    plt.plot(fpr[i], tpr[i], label=f'Class {lb.classes_[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Chance')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Letter Dataset (First 3 Classes)')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelBinarizer

# ------------------------------ Part 1: IRIS Dataset ------------------------------

# Load Iris dataset
iris_df = pd.read_csv("/content/iris (2).csv")

# Features and labels
X_iris = iris_df.iloc[:, :-1]
```

```
y_iris = iris_df.iloc[:, -1]

# Train-test split
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, 
y_iris, test_size=0.2, random_state=42)

# SVM with Linear Kernel
model_linear = SVC(kernel='linear')
model_linear.fit(X_train_iris, y_train_iris)
y_pred_linear = model_linear.predict(X_test_iris)

print("IRIS Dataset - Linear Kernel")
print("Accuracy:", accuracy_score(y_test_iris, y_pred_linear))
print("Confusion Matrix:\n", confusion_matrix(y_test_iris, y_pred_linear))

# SVM with RBF Kernel
model_rbf = SVC(kernel='rbf')
model_rbf.fit(X_train_iris, y_train_iris)
y_pred_rbf = model_rbf.predict(X_test_iris)

print("\nIRIS Dataset - RBF Kernel")
print("Accuracy:", accuracy_score(y_test_iris, y_pred_rbf))
print("Confusion Matrix:\n", confusion_matrix(y_test_iris, y_pred_rbf))
```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:

21/4/25 Lab - 6

Random Forest Algorithm:

1. Collect Data
   Start with a labeled dataset

2. Create Subsets of the Data
   Randomly sample the training data with replacement to create multiple subsets. This process is known as bootstrapping

3. Build Decision Trees
   For each data subset, build a decision tree. When splitting a node, randomly select a subset of features and use that subset to determine the best split.

4. Repeat for multiple Trees
   Repeat steps 2 and 3 to generate a forest of decision trees.
   Each tree is built on a different subset of the data and uses a random subset of features at each split.

5. Make predictions with each tree:
   After all the trees are built, use each tree to make a prediction for new, unseen data

   For classification, each tree provides a class prediction
   For regression, each tree provides a numerical value

6. Aggregate the results
Classification : Use majority voting to determine the final class label. Each tree "votes", for a class and the class with the most votes is selected as the final prediction

Predicted class = Mode (most frequent vote from all trees)

Regression : Take the average of all the tree's predicted values.

$$\text{Predicted value} = \frac{1}{N} \sum_{i=1}^{N} \text{Prediction from Tree}_i$$

N = No. of trees in the forest

Code:

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("iris_.csv")   # Use the correct path if needed

# Prepare data
X = df.iloc[:, :-1]  # All features
y = df.iloc[:, -1]   # Class label

# Split into train and test
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,  test_size=0.2,
random_state=42)

# 1. Default model with n_estimators = 10
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default n_estimators=10 accuracy: {default_score:.4f}")

# 2. Tune number of trees
best_score = 0
best_n = 0
scores = []

for n in range(1, 101):  # Try from 1 to 100 trees
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    score = rf.score(X_test, y_test)
    scores.append(score)
    if score > best_score:
        best_score = score
        best_n = n

print(f"Best accuracy: {best_score:.4f} with n_estimators={best_n}")
from sklearn.metrics import confusion_matrix
```

```
# Train the best model
rf_best = RandomForestClassifier(n_estimators=best_n, random_state=42)
rf_best.fit(X_train, y_train)
y_pred_best = rf_best.predict(X_test)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_best)
print("Confusion Matrix for Best Model:")
print(cm)

# Optional: Plot the scores
plt.plot(range(1, 101), scores)
plt.xlabel("Number of Trees (n_estimators)")
plt.ylabel("Accuracy")
plt.title("Random Forest Accuracy vs Number of Trees")
plt.grid(True)
plt.show()
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot:

21/4/25

Lab-7

AdaBoost Classifier Algorithm :

Input :

⇒ Training data : $D = \{(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)\}$
where $y_i \in \{-1, +1\}$

⇒ Number of boosting : T

Output :

⇒ Final strong classifier $H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x))$

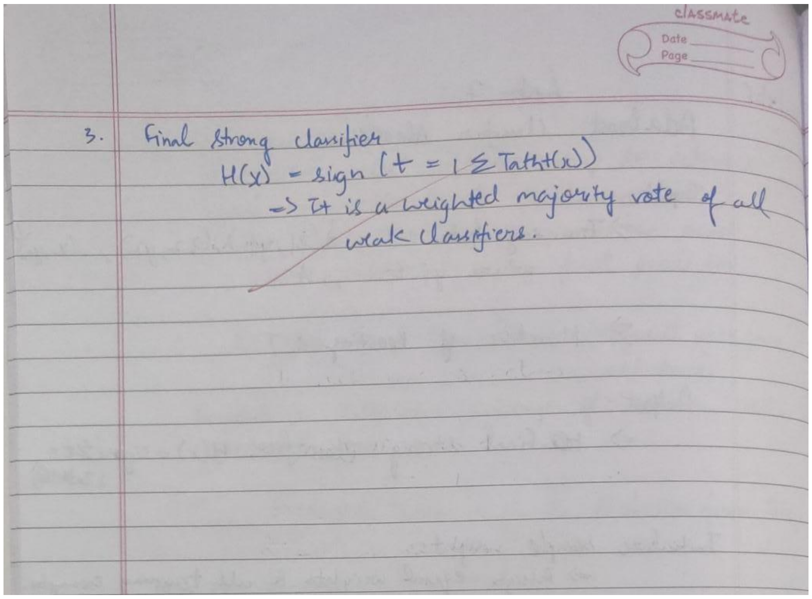
1. Initialize sample weights:
⇒ Assign equal weights to all training examples.
2. For each boosting round t=1 to T:

(a) Train a weak classifier $h_t(x)$:
⇒ train using the current weights $w_t(i)$
⇒ Weak classifiers are usually decision stumps (trees with 1 split)

(b) compute the weighted classification error
$\epsilon_t = \sum_{i=1}^{n} w_t(i) \cdot I(h_t(x_i) \neq y_i)$
⇒ Here $I(\cdot)$ is an indicator function
⇒ ϵ_t measures how badly $h_t(x)$ performed

(c) Compute the importance (weight) of the weak classifier
$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

(d) Update sample weights :
$w_{t+1}(i) = w_t(i) \cdot \exp(-\alpha_t y_i h_t(x_i))$
· Misclassified samples will have weights increased
· Normalize the weights so they sum to 1



3. Final Strong classifier

$H(x) = \text{sign}\,(t = 1 \sum T \alpha_t h_t(x))$

-> It is a weighted majority vote of all weak classifiers.

Code:

```
import pandas as pd
```

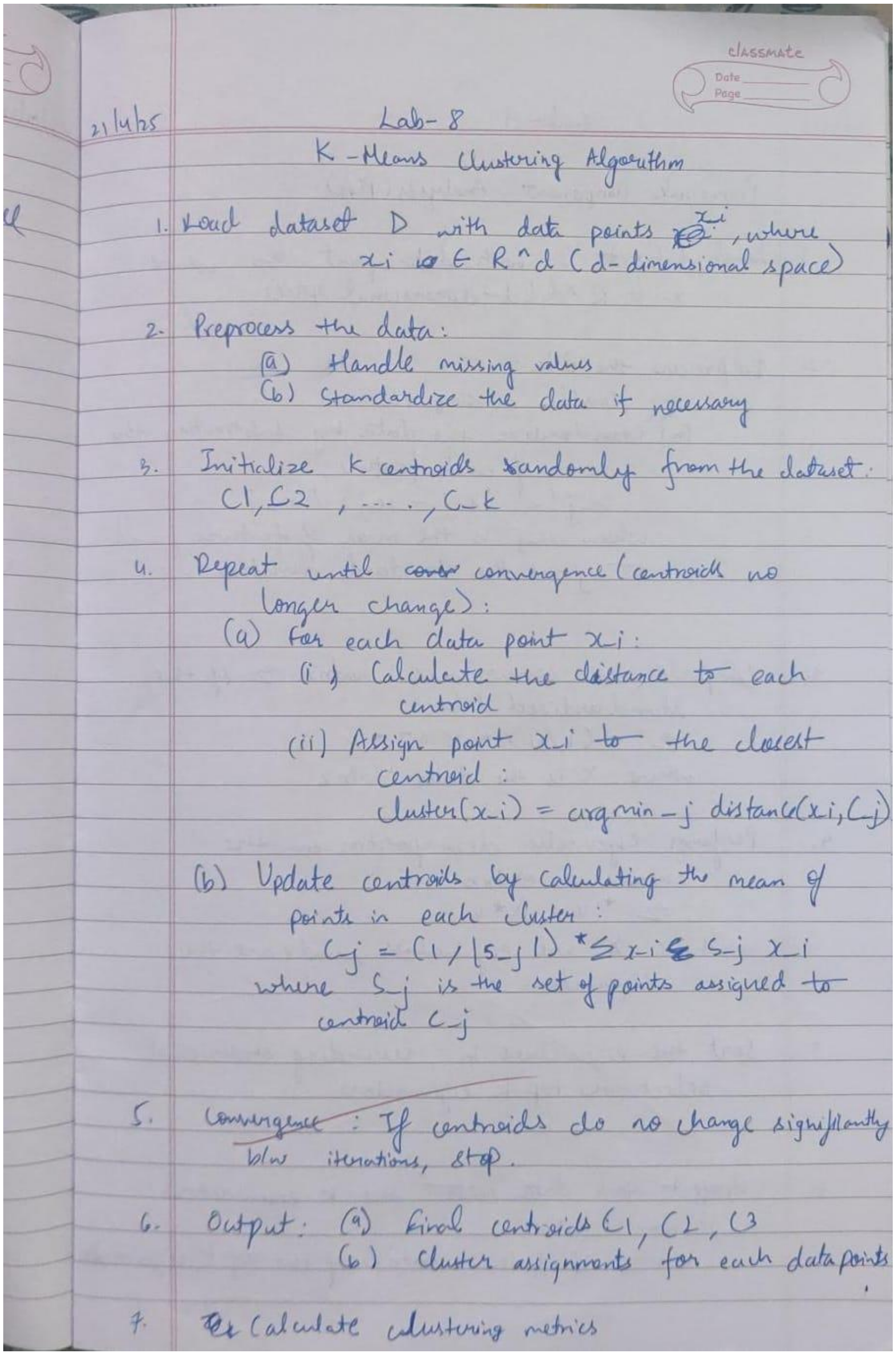
```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("income.csv")

# Preprocess (basic handling - encode categorical features)
df = df.dropna()  # drop missing values if any
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Features and target
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 1. AdaBoost with 10 estimators
model_default = AdaBoostClassifier(n_estimators=10, random_state=42)
model_default.fit(X_train, y_train)
y_pred_default = model_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default AdaBoost Accuracy (n estimators=10): {default score:.4f}")
```

```python
# 2. Fine-tune number of estimators
best_score = 0
best_n = 0
scores = []

for n in range(1, 101):
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    scores.append(score)
    if score > best_score:
        best_score = score
        best_n = n

print(f"Best AdaBoost Accuracy: {best_score:.4f} with n_estimators={best_n}")

# Plot accuracy vs number of estimators
plt.plot(range(1, 101), scores)
plt.xlabel("Number of Estimators")
plt.ylabel("Accuracy")
plt.title("AdaBoost Accuracy vs Number of Estimators")
plt.grid(True)
plt.show()
```

Program 10

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

Screenshot:



classmate

Date

Page

21/4/25

Lab-8

K-Means Clustering Algorithm

1. Load dataset D with data points x_i, where $x_i \in R^d$ (d-dimensional space)

2. Preprocess the data:
   (a) Handle missing values
   (b) Standardize the data if necessary

3. Initialize k centroids randomly from the dataset: C1, C2, ..., C_k

4. Repeat until convergence (centroids no longer change):
   (a) For each data point x_i:
      (i) Calculate the distance to each centroid
      (ii) Assign point x_i to the closest centroid:
      cluster(x_i) = argmin_j distance(x_i, C_j)
   (b) Update centroids by calculating the mean of points in each cluster:
      C_j = (1/|S_j|) * Σ x_i ∈ S_j x_i
      where S_j is the set of points assigned to centroid C_j

5. Convergence: If centroids do no change significantly b/w iterations, stop.

6. Output: (a) Final centroids C1, C2, C3
   (b) Cluster assignments for each data points

7. Calculate clustering metrics

Code:

```
import pandas as pd
```

```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load dataset
df = pd.read_csv("iris_.csv")   # Use correct path if needed

# Use only petal length and petal width
X = df[['petal_length', 'petal_width']]

# Optional: Scaling for better clustering performance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Elbow Method to find optimal k
inertia = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
# Plot elbow graph
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, 'bo-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.title('Elbow Method For Optimal k')
plt.grid(True)
plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

Screenshot:

21/4/25 Lab-9

Principal Component Analysis (PCA)

1. Load dataset D with data point x_i, where x_i ∈ R^d (d-dimensional space)

2. Preprocess the data:
   (a) Handle missing values
   (b) Standardize the data by subtracting the mean of each feature:
   $x\_j' = (x\_j - \mu\_j) / \sigma\_j$
   where μ_j is the mean of feature j and σ_j is the standard deviation

3. Compute the covariance matrix Σ of the standardized data:
   $Σ = (1/N) * X^T * X$
   where X is the data Matrix

4. Perform eigenvalue decomposition on the covariance Matrix Σ:
   $Σ * v = \lambda * v$
   where λ are eigenvalues, and v are the eigenvectors

5. Sort the eigenvalues in descending order and select the top k eigenvectors

6. Project the data onto the k eigenvectors
   $X\_new = X * V$
   where V is the matrix of the top k eigenvectors

7. Output :

(a) Transformed dataset X new

(b) visualize the reduced data (if $k=2$ ~~axes~~ or $k=3$)

Code:

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


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


categorical_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina',
'ST_Slope']
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)


X = df.drop("HeartDisease", axis=1)
y = df["HeartDisease"]


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

models = {
    "SVM": SVC(),
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier()
}

accuracy_before_pca = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy_before_pca[name] = accuracy_score(y_test, y_pred)

pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)
```

```python
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y,
test_size=0.2, random_state=42)

accuracy_after_pca = {}
for name, model in models.items():
    model.fit(X_train_pca, y_train_pca)
    y_pred_pca = model.predict(X_test_pca)
    accuracy_after_pca[name] = accuracy_score(y_test_pca, y_pred_pca)

print("□ Accuracy BEFORE PCA:")
for name, acc in accuracy_before_pca.items():
    print(f"{name}: {acc:.4f}")

print("\n□ Accuracy AFTER PCA:")
for name, acc in accuracy_after_pca.items():
    print(f"{name}: {acc:.4f}")

print(f"\nOriginal features: {X.shape[1]}")
print(f"Features after PCA: {X_pca.shape[1]}")
```