

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**
“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT
on**

Machine Learning (23CS6PCMAL)

Submitted by

SHREE SANKET (1BM22CS261)

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 **Shree sanket (1BM22CS261)**, who is 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 an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab Faculty Incharge	
Name: Ms. Saritha A N Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Sl. No.	Date	Experiment Title	Page No.
1	21-2-2025	Write a python program to import and export data using Pandas library functions	1-4
2	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	5-8
3	10-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	9-12
4	17-3-2025	Build Logistic Regression Model for a given dataset	13-16
5	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample	17-18
6	7-4-2025	Build KNN Classification model for a given dataset	19-21
7	21-4-2025	Build Support vector machine model for a given dataset	22-24
8	5-5-2025	Implement Random forest ensemble method on a given dataset	25-27
9	5-5-2025	Implement Boosting ensemble method on a given dataset	28-29
10	12-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file	30-31
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method	32-34

Github Link:

<https://github.com/shreesanket/ML-LAB>

Program 1

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

Screenshot:

LAB-09

1) Method-1 : Initializing values directly into Dataframe
Insert your unknown values, five rows of data with column heading as USN, NAME, MARKS.

```
→ import pandas as pd  
data = {  
    'USN': ['IBM22CS261', 'IBM22CS270', 'IBM22CS245',  
            'IBM22CS078', 'IBM22CS050'],  
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Tom'],  
    'Marks': [85, 90, 95, 92, 99]
```

3
df = pd.DataFrame(data)
print("Sample Data :")
print(df.head())

Method 2: Importing database from sklearn.
datasets / Loading diabetes dataset.

```
→ from sklearn.datasets import load_diabetes  
import pandas as pd  
diabetes = load_diabetes()  
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)  
df['target'] = diabetes.target  
print(df.head(5))
```

→ Method 3: Importing files from a specific CSV file.

→ file_path='data.csv'
df = pd.read_csv('content/sample_data.csv')
print(df.head(5))

Method 4: Downloading datasets from existing Datasets repositories like kaggle, UCI, medley.

```
→ df = pd.read_csv('content/Diabetics.csv').  
print(df.head(5))
```

To - Do - 2

Using the code given in the above slides, do the exercise of the "Stock market Data Analysis" Considering

- 1) HDFC Bank Ltd, ICICI Bank Ltd, Kotak Mahindra Bank Ltd.
tickers = ['HDFCBANK.NS', 'ICICIBANK.NS', 'KOTAKBANK.NS']
- 2) Start date: 2020-01-01, End date: 2024-12-31
- 3) Plot the closing Price & daily returns for all 3 Banks

→ Import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAK.NS"]

```
data = yf.download(tickers, start="2020-01-01",  
                  end="2024-12-31", group_by='tickers')
```

Bafna Gold

```

HDFC_Data = data['HDFCBANK.NS']
HDFC_Data['Daily Return'] = HDFC_Data['Close'].pct_change()
plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
plt.plot(HDFC_Data['Close'], title="HDFC - closing price")
plt.subplot(2,1,2)
HDFC_Data['Return'].plot(title="HDFC-Daily-Return")

ICICIData = data['ICICIBANK.NS']
ICICIData['Daily Return'] = HDFC_Data['close'].pct_change()
plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
ICICIData['close'].plot(title="ICICI - closing price")
plt.subplot(2,1,2)
ICICI_Data['Daily Return'].plot(title="ICICI - Daily Return")

KOTAKData = data['KOTAKBANK.NS']
KOTAKData['Daily Return'] = KOTAKData['close'].pct_change()
plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
KOTAKData['close'].plot(title='KOTAK - closing price')
plt.subplot(2,1,2)
KOTAK_Data['Daily Return'].plot(title="KOTAK")

```

Code:

```
import yfinance as yf
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
```

```
data = yf.download(tickers, start="2023-10-01", end="2024-10-01", group_by='ticker')
```

```
HDFC_Data = data['HDFCBANK.NS']
```

```
HDFC_Data['Daily Return'] = HDFC_Data['Close'].pct_change()
```

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

plt.subplot(2, 1, 1)

HDFC_Data['Close'].plot(title="HDFC Bank - Closing Price")

plt.subplot(2, 1, 2)

HDFC_Data['Daily Return'].plot(title="HDFC Bank - Daily Returns", color='orange')

plt.tight_layout()

plt.show()
```

```
ICICIBANK_Data = data['ICICIBANK.NS']

ICICIBANK_Data['Daily Return'] = ICICIBANK_Data['Close'].pct_change()
```

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

plt.subplot(2, 1, 1)
```

```
ICICIBANK_Data['Close'].plot(title="ICICIBANK_Data Bank - Closing Price")
```

```
plt.subplot(2, 1, 2)
```

```
ICICIBANK_Data['Daily Return'].plot(title="ICICIBANK_Data Bank - Daily Returns",
color='orange')
```

```
plt.tight_layout()
```

```
plt.show()
```

```
KOTAKBANK_Data = data['KOTAKBANK.NS']
```

```
KOTAKBANK_Data['Daily Return'] = KOTAKBANK_Data['Close'].pct_change()
```

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

```
plt.subplot(2, 1, 1)
KOTAKBANK_Data['Close'].plot(title="KOTAKBANK_Data Bank - Closing Price")
plt.subplot(2, 1, 2)
KOTAKBANK_Data['Daily Return'].plot(title="KOTAKBANK_Data Bank - Daily Returns",
color='orange')
plt.tight_layout()
plt.show()
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot :

LAB-02

```
import pandas as pd
df = pd.read_csv("housing.csv")
df.info()
df.describe()
df['ocean', 'proxinity'].value_counts()
npx_val = df['isnull'].sum()
val = npx_val[npx_val > 0]
print(val)

Diabetes
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
StandardScaler
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv("dataset of diabetes.csv")
print(df.isnull())
# missing values
print(df.isnull().sum())
# import
nC = df.select_dtypes(include=['float64', 'int64'])
```

```
import simplejson (strategy="max")
dt [num_columns] = impoter_file_transform(dt [num_columns])
cat = dt.select_dtypes(include=['object']).columns
# Handeling categorical data
label_Encoder = LabelEncoder()
dt['Gender'] = label_Encoder.fit_transform(dt['Gender'])
df['class'] = label_Encoder.fit_transform(df['class'])

# Handeling outliers
Q1 = dt[num_columns].quantile(0.25)
Q3 = df[num_columns].quantile(0.75)
IQR = Q3 - Q1
dt_clean = dt[(~(dt['class'] < col - 1.5 * IQR) & (~dt['class'] > col + 1.5 * IQR))]
any (axis=1)

# Data transformation
# Apply minmax or standard scalar
scalar_choice = "minmax"
if scalar_choice == "minmax"
    scalar = MinMaxScaler()
else:
    scalar = StandardScaler()
df_scaled = pd.DataFrame(scalar.fit_transform(dt_clean[num_columns]),
                           columns=dt_clean.columns)
```

df_final = pd.concat([df_clean[cols_columns], dt_scaled7, axis=3]).
i) No column had missing values Gender & clean min max scaling if fixed range for feature bounded range no outliers Standard scalar Outlier exist Normal distributed values.
(1) - Average Feature scaling (2) - Standard Feature scaling (3) - Min Max Feature scaling (4) - Z score
scale columns is suitable when considering multiple models standardization makes it easier to compare different models standardization makes it easier to compare different models
df_final = df_final.drop(['Pregnancies'], axis=1)

Code:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

```
diabetes_df = pd.read_csv("/content/Dataset of Diabetes .csv")
```

```
adult_income_df = pd.read_csv("/content/adult.csv")

def preprocess_data(df):
    missing_values = df.isnull().sum()
    print("Columns with missing values:\n", missing_values[missing_values > 0])

    numerical_cols = df.select_dtypes(include=["float64", "int64"]).columns
    categorical_cols = df.select_dtypes(include=["object"]).columns

    imputer = SimpleImputer(strategy='most_frequent')
    df[categorical_cols] = imputer.fit_transform(df[categorical_cols])

    imputer = SimpleImputer(strategy='median')
    df[numerical_cols] = imputer.fit_transform(df[numerical_cols])

    print("\nCategorical Columns:\n", categorical_cols)

    encoder = OneHotEncoder(drop='first', sparse_output=False)
    encoded_categorical_data = encoder.fit_transform(df[categorical_cols])

    encoded_categorical_df = pd.DataFrame(encoded_categorical_data,
                                           columns=encoder.get_feature_names_out(categorical_cols))

    df = df.drop(categorical_cols, axis=1)
    df = pd.concat([df, encoded_categorical_df], axis=1)
```

```
from scipy import stats

z_scores = np.abs(stats.zscore(df[numerical_cols]))

df = df[(z_scores < 3).all(axis=1)]


min_max_scaler = MinMaxScaler()

df[numerical_cols] = min_max_scaler.fit_transform(df[numerical_cols])


standard_scaler = StandardScaler()

df[numerical_cols] = standard_scaler.fit_transform(df[numerical_cols])


return df


diabetes_df = preprocess_data(diabetes_df)

print("\nPreprocessed Diabetes Dataset:")

print(diabetes_df.head())


adult_income_df = preprocess_data(adult_income_df)

print("\nPreprocessed Adult Income Dataset:")

print(adult_income_df.head())
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:

LAB - D3
Date: 12/03 Page: 09

1) Consider a binary classification problem where we want to predict whether a student will pass or fail based on their study hours. The logistic regression model has been trained & the learned parameters are $a_0 = -5$ & $a_1 = 0.8$.

a) Write the logistic regression equation for this pattern.

$$\rightarrow p(\text{pass}) = \frac{e^{-(-5 + 0.8x)}}{1 + e^{-(-5 + 0.8x)}}$$

b) calculate the probability that a student who studies for 7 hours will pass

$$\rightarrow p(\text{pass}) = \frac{e^{-(-5 + 0.8 \cdot 7)}}{1 + e^{-(-5 + 0.8 \cdot 7)}}$$

$$= \frac{e^{-(-5 + 5.6)}}{1 + e^{-(-5 + 5.6)}} = \frac{e^{0.4}}{1 + e^{0.4}} = \frac{0.5488}{1.5488} = 0.3545$$

~~p(pass) = 0.645~~

c) Determine the predicted class (pass or fail) for this student based on a threshold of 0.5

~~as threshold is 0.5 but student passing probability is 0.645~~

~~Predicted class is "pass"~~

Data: Page: 10

2) Consider $z = [2, 1, 0]$ for three classes, apply softmax function to find the probability values of three classes

$$\rightarrow P(y_j) = \frac{e^{z_j}}{\sum_{j=1}^3 e^{z_j}}$$

$$e^2 = 7.389 \quad e^1 = 2.718 \quad e^0 = 1$$

$$\Rightarrow \sum_{j=1}^3 e^{z_j} = 7.389 + 2.718 + 1 = 11.107$$

$$\text{Now, } P(y_1) = \frac{e^2}{11.107} = \frac{7.389}{11.107} = 0.665$$

$$P(y_2) = \frac{e^1}{11.107} = \frac{2.718}{11.107} = 0.245$$

$$P(y_3) = \frac{e^0}{11.107} = \frac{1}{11.107} = 0.090$$

Linear regression

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

$$df = pd.read_csv("./content/data.csv")$$

$$X = df[['area']].values$$

$$Y = df['sales'].values$$

```

Date: Page: 1
model = LinearRegression()
model.fit(x, y)

slope = model.coef_[0]
intercept = model.intercept_

y_pred = model.predict(x)

print(f"Linear Regression Equation: y = {slope}x + {intercept}")

weeks_input = float(input("Enter number of weeks"))
sales_output = model.predict([weeks_input])
print(f"predicted sales for {weeks_input} weeks: {sales_output:.2f}")

output Linear Regression Equation: y = 2.00x + -0.50
Enter number of weeks: 5
Predicted sales for 5.0 weeks: 10.50

Linear regression (Matrix approach)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

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

X = df[['weeks']].values
y = df['sales'].values

model = LinearRegression()
model.fit(X, y)

slope = model.coef_[0]

```

```

x_b = np.c_[np.ones((x.shape[0], 1)), x]
theta_best = np.linalg.inv(x_b.T @ x_b) @
x_b.T @ y

y_pred = x_b @ theta_best

plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x, y_pred, color='red', linestyle='--',
label=f'Regression line: y = {theta_best[0]:.2f}x + {theta_best[1]:.2f}')
plt.xlabel('Weeks')
plt.ylabel('Sales')
plt.title('Linear Regression (Matrix approach)')
plt.legend()
plt.show()

print(f"Linear Regression Equation: y = {theta_best[0]:.2f}x + {theta_best[1]:.2f}")

def predict_sales(weeks):
    return np.array([1, weeks]) @ theta_best

week_input = float(input("Enter number of weeks"))
sales_output = predict_sales(weeks_input)
print(f"predicted sales for {weeks_input} weeks: {sales_output:.2f}")

Output: Enter number of weeks: 5
predicted sales for 5.0 weeks: 10.50
Bafna Gold

```

Code:

Linear regression

```

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

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

X = df[['weeks']].values
y = df['sales'].values

model = LinearRegression()
model.fit(X, y)

slope = model.coef_[0]

```

```

intercept = model.intercept_
y_pred = model.predict(X)

plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', linestyle='--', label=f'Regression Line: y = {slope:.2f}x + {intercept:.2f}')
plt.xlabel('Weeks')
plt.ylabel('Sales')
plt.title('Linear Regression')
plt.legend()
plt.show()

print(f'Linear Regression Equation: y = {slope:.2f}x + {intercept:.2f}')

week_input = float(input("Enter number of weeks: "))
sales_output = model.predict([[week_input]])[0]
print(f'Predicted sales for {week_input} weeks: {sales_output:.2f}')

```

Multi-linear regression

```

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

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

X = df[['weeks']].values
y = df['sales'].values

X_b = np.c_[np.ones((X.shape[0], 1)), X]
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y

y_pred = X_b @ theta_best

plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', linestyle='--', label=f'Regression Line: y = {theta_best[1]:.2f}x + {theta_best[0]:.2f}')
plt.xlabel('Weeks')
plt.ylabel('Sales')
plt.title('Linear Regression (Matrix Approach)')
plt.legend()
plt.show()

print(f'Linear Regression Equation: y = {theta_best[1]:.2f}x + {theta_best[0]:.2f}')

```

```
def predict_sales(week):
    return np.array([1, week]) @ theta_best

week_input = float(input("Enter number of weeks: "))
sales_output = predict_sales(week_input)
print(f'Predicted sales for {week_input} weeks: {sales_output:.2f}')
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:

LAB-04 Data Output: 13

Logistic Regression

- 1) Consider a binary classification problem where we want to predict logistic regression model has been trained, & the learned parameters are $a_0 = -5$ & $a_1 = 0.8$
- a) write the logistic equation for the problem

$$p(\text{pass}) = \frac{1}{1 + e^{-(5 + 0.8x_1)}}$$
- b) calculate the probability that a student who studies for 7 hours will pass

$$p(\text{pass}) = \frac{1}{1 + e^{-(5 + (0.8 \cdot 7))}} = \frac{1}{0.5488} \approx 0.645$$
- c) determine the predicted class (pass or fail)
 → for the student, based on threshold of 0.5
 → predicted class "pass"
- 2) Consider $z = [2, 1, 0]$, for three classes apply softmax function to find the probability values of three classes

$$\rightarrow p(y_i) = \frac{e^{z_i}}{\sum_{j=1}^3 e^{z_j}}$$

$$e^2 = 7.389$$

$$e^1 = 2.718$$

$$e^0 = 1$$

$$\sum_{j=1}^3 e^{z_j} = 2.718 + 7.389 + 1 = 11.107$$
- Now $p(y_1) = \frac{e^2}{11.107} = \frac{7.389}{11.107} \approx 0.665$
- $p(y_2) = \frac{e^1}{11.107} = \frac{2.718}{11.107} \approx 0.2465$
- $p(y_3) = \frac{e^0}{11.107} = \frac{1}{11.107} \approx 0.090$

Ques. For dataset file "HR Comma Sep.CSV"
 i) Which variable you identify as having a direct & clean impact on employee retention? why?
 * Satisfaction level: correlation with low salary are more likely to leave
 ii) average monthly hours: Correlation - Overworked employee tend to leave
 iii) time spent at company: positive correlation: longer tenure increases attrition risk
 iv) salary level (low salary correlates with high attrition)
 v) What was the accuracy of your logistic regression model? Do you think this is a good accuracy? Why or why not?
 Accuracy = 76% - 80%
 Yes accuracy is good.
 But accuracy alone isn't enough. Precision/recall for class 1 matters more in imbalanced dataset
 vi) 200 datasets
 i) Did you perform any data preparation steps? If yes what were they & why were they necessary
 → a) merged dataset
 b) dropping non-featurer columns
 c) checked for missing values

- Date: 30-9-23 Page: 15
- merging encoded meaningful class labels.
 - cleaned dataset from unrelated features
 - i) are there any missing or inconsistent values in the dataset? If yes, do you handle them?
 - No missing values.
 - If present.
 - Drop rows
 - Or impute
 - ii) what does the confusion matrix tell you about the performance of your model?
 - High diagonal values → most classes predicted correctly
 - misclassification occurred between Smoker classes.
 - iv) which class types were most frequently miss classified? Why do you think this happens?
 - Bug vs Severe headache

Code:

Binary logistic regression

```

import pandas as pd
from matplotlib import pyplot as plt

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_

model.intercept_

import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

def prediction_function(age):
    z = 0.127 * age - 4.973
    y = sigmoid(z)
    return y

age = 35
prediction_function(age)

```

Multiclass logistic regression

```

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

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

```

```
iris.head()

X = iris.drop('species', axis='columns')
y = iris.species

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

model = LogisticRegression(multi_class='multinomial')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

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()
```

Program 5

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

Screenshot:

LAB-05

```

use an appropriate dataset for building the decision tree (ID3) and apply this knowledge to classify a new sample

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier,
export_text, plot_tree

data = pd.read_csv("tennis.csv")

from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
for col in data.columns:
    data[col] = encoder.fit_transform(data[col])

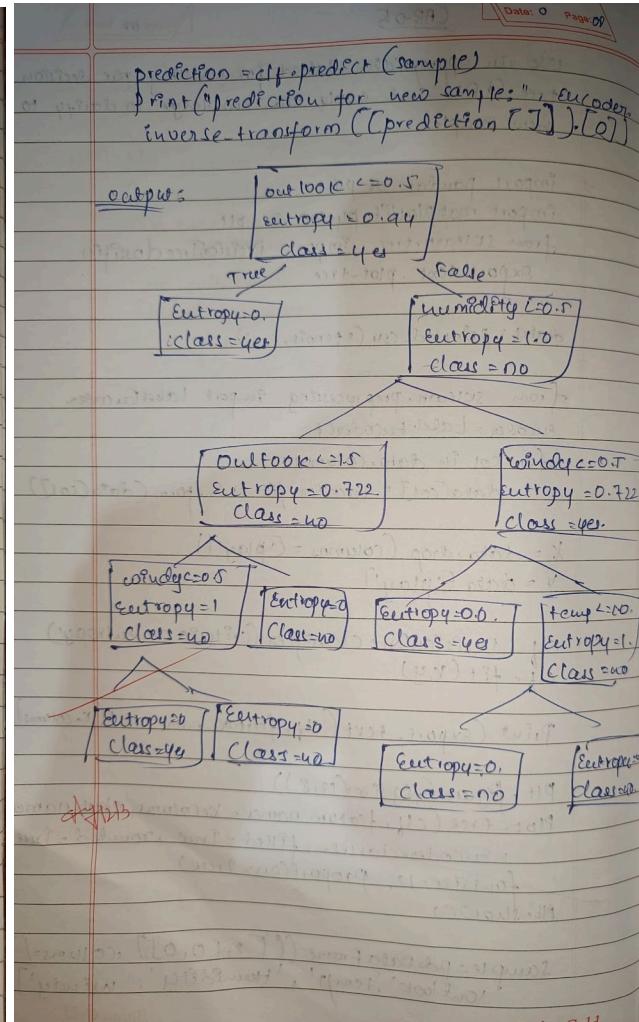
X = data.drop(columns=['play'])
y = data['play']

clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X,y)

print(export_text(clf, feature_names=list(X.columns)))
plt.figure(figsize=(12,8))
plot_tree(clf, feature_names=X.columns, class_names=encoder.classes_, filled=True, rounded=True,
fontsize=12, proportion=True)
plt.show()

sample = pd.DataFrame([[2,1,0,0]], columns=['outlook','temp','humidity','windy'])

```



Code:

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

```

```
import matplotlib.pyplot as plt

data = pd.read_csv('/content/seattle-weather.csv')
data_cleaned = data.drop('date', axis=1)

label_encoder = LabelEncoder()
data_cleaned['weather'] = label_encoder.fit_transform(data_cleaned['weather'])

X = data_cleaned.drop('weather', axis=1)
y = data_cleaned['weather']

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

id3_classifier = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=42)
id3_classifier.fit(X_train, y_train)

plt.figure(figsize=(20, 12))
tree.plot_tree(id3_classifier, feature_names=X.columns, class_names=list(label_encoder.classes_), filled=True, fontsize=10)
plt.title("ID3 Decision Tree Classifier (Enhanced Clarity)")
plt.show()
```

Program 6

Build KNN Classification model for a given dataset

Screenshot:

LAB-06

Date: 02/04/2018

Build KNN classification model for a given dataset

1) Consider the following dataset for k=3
 & test data (x, 35, 100) as (Person, Age, Salary) Solve using KNN classifier model
 & Predict the target.

Person	Age	Salary	Target
A	18	50	N
B	23	55	N
C	24	70	N
D	41	60	y
E	48	70	y
F	38	40	y
x	35	100	?

Step 1: Since Age & Salary are on different scales. Normalize using Z-Score. Standardize.
 * Mean (Age) = 31.16
 * Std Dev (Age) = 9.52
 * Mean (Salary) = 60.83
 * Std Dev (Salary) = 15.46

Scaled values (age, salary).

A: (-1.34, 0.70)
 B: (-0.83, -0.38)
 C: (-0.73, 0.59)
 D: (1.00, -0.05)
 E: (1.20, 0.59)
 F: (0.90, -1.35)
 X: (0.39, 2.57)

Bafna Gold

Date: _____ Page: 17

compute Euclidean distance

Person	Age	Salary	Distance
A	-1.34	0.70	3.32
B	-0.83	-0.38	2.02
C	-0.73	0.59	1.06
D	1.00	-0.05	2.58
E	1.20	0.59	2.06
F	0.90	-1.35	3.89

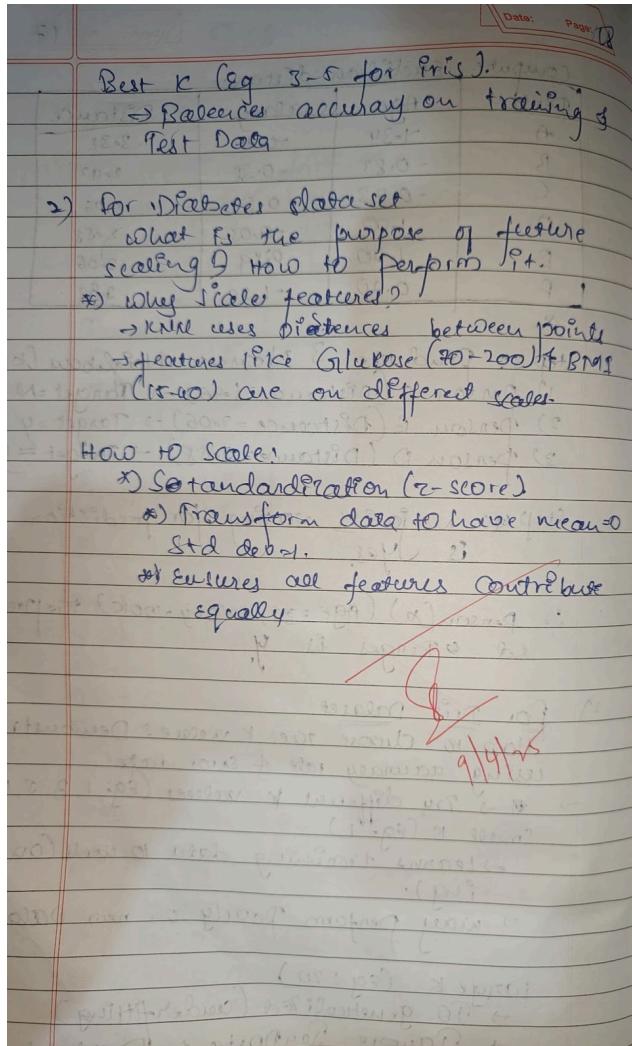
Step 3: Find the 3 Nearest Neighbors (k=3)
 1) person C (Distance = 1.06) \rightarrow Target = N
 2) person E (Distance = 2.06) \rightarrow Target = y
 3) person D (Distance = 2.58) \rightarrow Target = y

Step 4: Majority voting for prediction is Yes.

\therefore Person(x) (Age = 35, Salary = 100) is predicted \rightarrow Target is y.

2) For this dataset
 How to choose the k value? Demonstrate using accuracy rate & error rate
 \rightarrow 1 \rightarrow Try different k values (Eg: 1, 3, 5, 10)
 Small k (Eg. 1)
 \rightarrow Learn training data to well (Overfitting).
 \rightarrow may perform poorly on new data
 Large k (Eg: 20)
 \rightarrow To generalized (Underfitting)
 \rightarrow Ignore important features

Bafna Gold



Code:

```

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

heart_df = pd.read_csv("/content/heart.csv")
X_heart = heart_df.iloc[:, :-1].values
y_heart = heart_df.iloc[:, -1].values

X_train_heart, X_test_heart, y_train_heart, y_test_heart = train_test_split(
    X_heart, y_heart, test_size=0.20, random_state=42)
  
```

```

)
scaler = StandardScaler()
X_train_heart = scaler.fit_transform(X_train_heart)
X_test_heart = scaler.transform(X_test_heart)

param_grid = {'n_neighbors': range(1, 21)}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_heart, y_train_heart)
best_k = grid_search.best_params_['n_neighbors']

knn_heart = KNeighborsClassifier(n_neighbors=best_k)
knn_heart.fit(X_train_heart, y_train_heart)
y_pred_heart = knn_heart.predict(X_test_heart)

print(f"[HEART RESULTS - Best k = {best_k}]")
print("Accuracy:", accuracy_score(y_test_heart, y_pred_heart))
print("Confusion Matrix:\n", confusion_matrix(y_test_heart, y_pred_heart))
print("Classification Report:\n", classification_report(y_test_heart, y_pred_heart))

plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test_heart, y_pred_heart),
            annot=True, fmt="d", cmap="Blues",
            xticklabels=["No Disease", "Disease"],
            yticklabels=["No Disease", "Disease"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

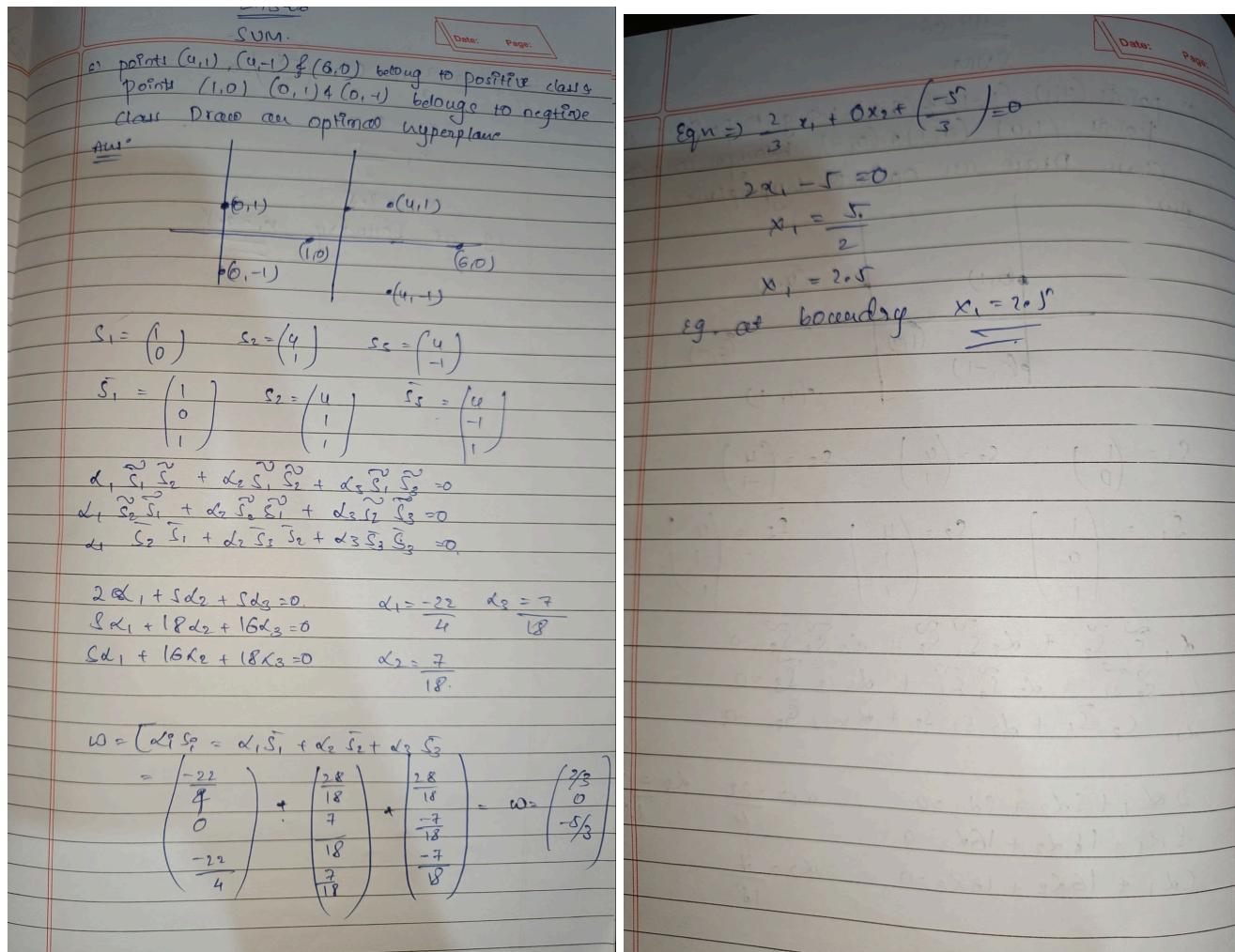
k_values = range(1, 21)
accuracies = [grid_search.cv_results_['mean_test_score'][i-1] for i in k_values]
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracies, marker='o')
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.title("Accuracy vs. k Value (Cross-Validated)")
plt.grid()
plt.show()

```

Program 7

Build Support vector machine model for a given dataset

Screenshot:



Code:

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

class SVM:

```
def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
    self.lr = learning_rate
    self.lambda_param = lambda_param
    self.n_iters = n_iters
    self.w = None
```

```

self.b = None

def fit(self, X, y):
    y = np.where(y <= 0, -1, 1)
    n_samples, n_features = X.shape
    self.w = np.zeros(n_features)
    self.b = 0

    for _ in range(self.n_iters):
        for idx, x_i in enumerate(X):
            condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1
            if condition:
                self.w -= self.lr * (2 * self.lambda_param * self.w)
            else:
                self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y[idx]))
                self.b += self.lr * y[idx]

def predict(self, X):
    approx = np.dot(X, self.w) + self.b
    return np.sign(approx)

def visualize(self, X, y, new_point=None, prediction=None):
    def get_hyperplane(x, w, b, offset):
        return (-w[0] * x + b + offset) / w[1]

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)

    for i, sample in enumerate(X):
        if y[i] == 1:
            plt.scatter(sample[0], sample[1], marker='o', color='blue', label='Class +1' if i == 0 else '')
        else:
            plt.scatter(sample[0], sample[1], marker='x', color='red', label='Class -1' if i == 0 else '')

    x0 = np.linspace(np.min(X[:, 0])-1, np.max(X[:, 0])+1, 100)
    x1 = get_hyperplane(x0, self.w, self.b, 0)
    x1_m = get_hyperplane(x0, self.w, self.b, -1)
    x1_p = get_hyperplane(x0, self.w, self.b, 1)

    ax.plot(x0, x1, 'k-', label='Decision Boundary')
    ax.plot(x0, x1_m, 'k--', label='Margins')
    ax.plot(x0, x1_p, 'k--')

    if new_point is not None:
        color = 'green' if prediction == 1 else 'orange'
        label = f'New Point: Class {"1" if prediction == 1 else "0"}'
        plt.scatter(new_point[0], new_point[1], c=color, s=100, edgecolors='black', label=label,
                    marker='*')

```

```

ax.legend()
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("SVM with New Point Prediction")
plt.grid(True)
plt.show()

if __name__ == "__main__":
    X = np.array([
        [1, 0],
        [0, 1],
        [0, -1],
        [4, -1],
        [4, 1],
        [6, 0]
    ])
    y = np.array([0, 0, 0, 1, 1, 1])

    new_point = np.array([[5, 5]])

    svm = SVM()
    svm.fit(X, y)
    prediction = svm.predict(new_point)[0]

    svm.visualize(X, y, new_point=new_point[0], prediction=prediction)

    print(f'New point {new_point[0]} classified as: {'Class 1' if prediction == 1 else 'Class 0'}')

```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:

LAB-007 Date: 19 Page: 1

Random forest

A random forest is a collection of decision trees that work together to make prediction.

Key features of Random forest

- Handles missing data
- Scales well with large & complex data
- Features based on their importance in making predictions
- Randomness

Algorithm: Randomforest

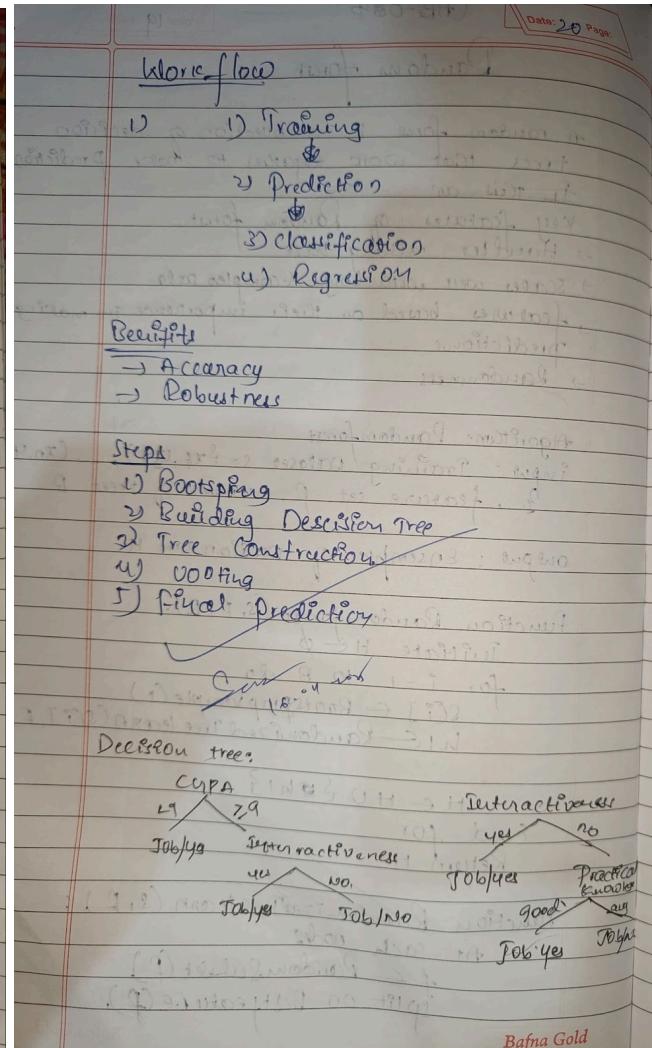
Input: Training Dataset $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$, F , number of trees B

Output: Ensemble of decision tree H

```

function Randomforest( $S, F, B$ ):
    Initialize  $H \leftarrow \emptyset$ 
    for  $i = 1$  to  $B$  do
         $S[i] \leftarrow \text{Bootstrap sample}(S)$ 
         $H_i \leftarrow \text{RandomizedTreeLearn}(S[i], F)$ 
    end for
    Return  $H$ .
  
```

function RandomizedTreeLearn(S, F):
 At each node
 $f \leftarrow \text{RandomSubset}(F)$
 split on BestFeature(f)



Date: 21 Page:

Q) For this dataset what is the best accuracy score & confusion matrix of the classifier you observed & testing how many trees.

Aw: Best Accuracy score observed is 0.9778.
Number of Trees used: 15

Confusion matrix

	versicolor	virginica
setosa	15	0
versicolor	0	14
virginica	0	1

Code:

```

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

iris_data = pd.read_csv('/content/iris.csv')

X = iris_data.drop('species', axis=1)
y = iris_data['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

rf_default = RandomForestClassifier(random_state=42)
rf_default.fit(X_train, y_train)

```

```

y_pred_default = rf_default.predict(X_test)
default_accuracy = accuracy_score(y_test, y_pred_default)

print(f'Accuracy with default n_estimators (10): {default_accuracy:.4f}')

n_estimators_list = range(1, 101)
accuracy_scores = []

for n in n_estimators_list:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)
    if n % 20 == 0:
        print(f'n_estimators: {n}, Accuracy: {accuracy:.4f}')

best_accuracy = max(accuracy_scores)
best_n = n_estimators_list[accuracy_scores.index(best_accuracy)]

print(f'\nBest accuracy: {best_accuracy:.4f} achieved with {best_n} trees')

plt.figure(figsize=(10, 6))
plt.plot(n_estimators_list, accuracy_scores)
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.title('Random Forest Accuracy vs Number of Trees')
plt.axvline(x=best_n, color='r', linestyle='--', label=f'Best n_estimators: {best_n}')
plt.legend()
plt.grid(True)
plt.show()

```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:

DNB - 08

Boosting Ensemble method

Date 22/03/2023

1) Consider ADA Boosting algorithm for sample Data show 10x2 Decision stamp calculator Steps for the attribute (Gpa).

Actual

Gpa	Interactions	Practical Knowledge	Communication	Total
>=9	Yes	Good	Good	44
<9	No	Good	Moderate	44
>=9	No	Avg	Moderate	44
<9	Yes	Avg	Good	40
>=9	Yes	Good	Moderate	44
>=9	Yes	Good	Moderate	44

1) Initialize Sample weight Di
 $D_i = \frac{1}{6}$ for all $i \in \{1, 2, \dots, 6\}$.

2) choose a decision stump based on Gpa.

Group 1) Gpa ≥ 9
 Group 2) Gpa < 9

Group results : Gpa ≥ 9 (sample 1, 3, 5, 6)
 samples : Yes, No, Yes, Yes \rightarrow Tip 0
 Gpa < 9 (sample 2, 4)
 samples : Yes, No \rightarrow Tip 0, choose yes.

Predicted labels

Sample	Gpa	Actual	Predicted
1	≥ 9	Yes	Yes
2	<9	Yes	Yes
3	≥ 9	No	Yes
4	<9	No	Yes
5	≥ 9	"	"

DNB - 08

Boosting Ensemble method

Date 23/03/2023

4) Error calculation
 $\text{Error} = D_1(3) + D_2(4) = \frac{1}{6} + \frac{1}{6} = \frac{2}{6} = 0.33$

5) Decision stream conclusion.
 Gpa $\geq 9 \rightarrow$ Yes
 Gpa < 9 \rightarrow Yes
 $\text{Error} = 0.33$

Sample	Gpa	Actual	Predicted
1	9.1	9.1	9.1
2	8.5	7.1	7.1
3	9.2	9.2	9.2
4	8.8	8.8	8.8
5	9.2	9.2	9.2
6	8.5	8.5	8.5

Code:

```

import pandas as pd
import numpy as np
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

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

```

```

X = df.drop('income_level', axis=1)
y = df['income_level']

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

base_estimator = DecisionTreeClassifier(max_depth=1)

ada = AdaBoostClassifier(estimator=base_estimator, n_estimators=10, random_state=42)
ada.fit(X_train, y_train)
y_pred = ada.predict(X_test)
score_10 = accuracy_score(y_test, y_pred)
print(f'Accuracy with 10 estimators: {score_10:.4f}')

n_estimators_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200]
train_scores = []
test_scores = []

for n in n_estimators_range:
    ada = AdaBoostClassifier(estimator=base_estimator, n_estimators=n, random_state=42)
    ada.fit(X_train, y_train)
    train_pred = ada.predict(X_train)
    train_acc = accuracy_score(y_train, train_pred)
    train_scores.append(train_acc)
    test_pred = ada.predict(X_test)
    test_acc = accuracy_score(y_test, test_pred)
    test_scores.append(test_acc)
    print(f'n_estimators: {n:3d} | Train Accuracy: {train_acc:.4f} | Test Accuracy: {test_acc:.4f}')

best_n = n_estimators_range[np.argmax(test_scores)]
best_score = max(test_scores)
print(f'\nBest performance: {best_score:.4f} with {best_n} estimators')

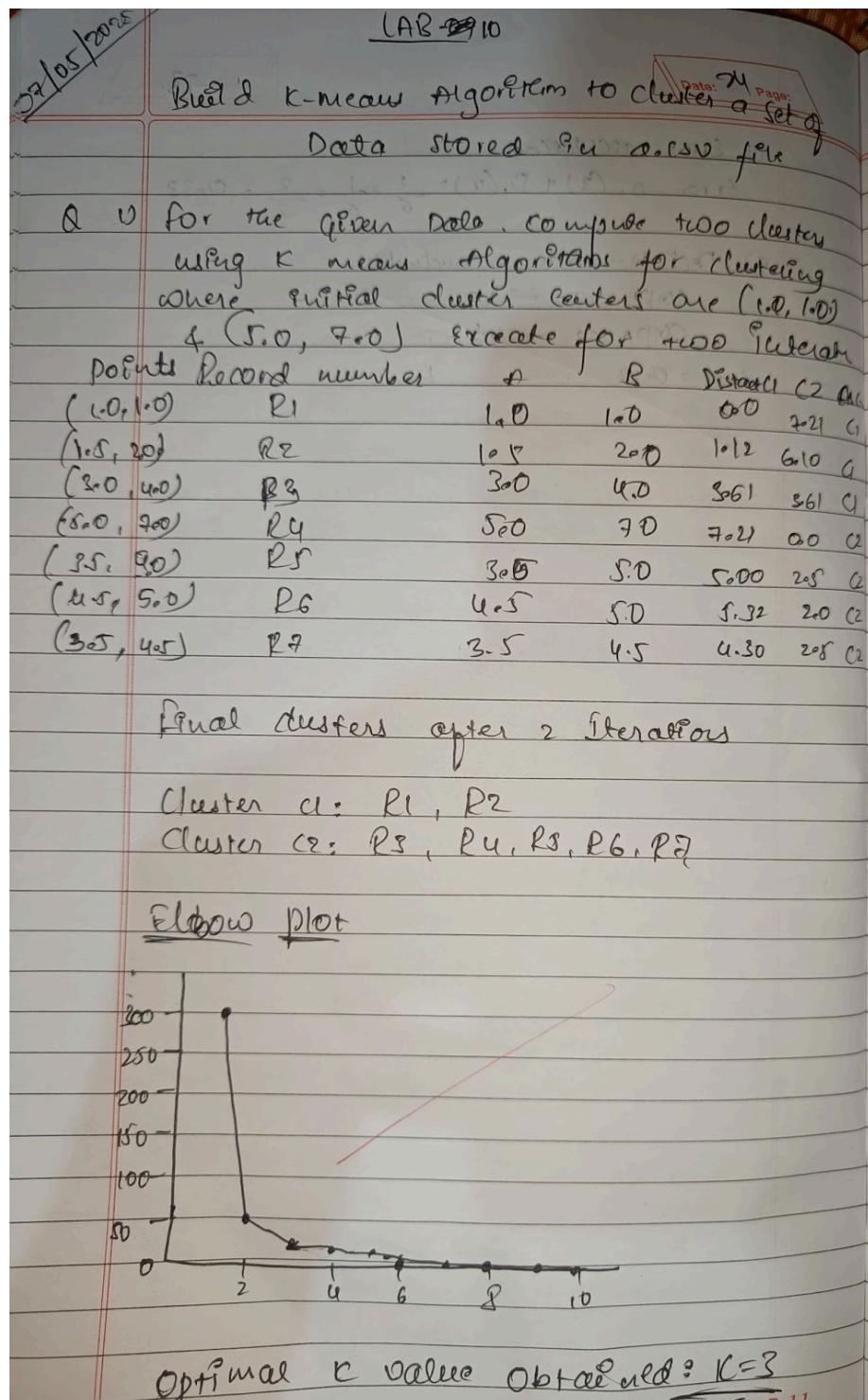
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, train_scores, label='Train Accuracy', marker='o', color='#00bcd4')
plt.plot(n_estimators_range, test_scores, label='Test Accuracy', marker='o', color='#ffcc80')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.title('AdaBoost Performance vs Number of Estimators')
plt.axvline(x=best_n, color='r', linestyle='--', label=f'Best n_estimators={best_n}')
plt.legend()
plt.grid(True)
plt.show()

```

Program 10

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

Screenshot:



Code:

```

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

iris_df = pd.read_csv('/content/iris.csv')

X = iris_df[['petal_length', 'petal_width']]

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

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_)

plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method For Optimal k')
plt.grid(True)
plt.show()

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)

iris_df['cluster'] = kmeans.labels_

print(iris_df.head())

```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:

LAB-101

Principle component analysis (PCA)

Q1 Given the data in table reduce the dimensions from 2 to 1 using PCA compute for first principal component

Feature	Example 1	Example 2	3	4
x_1	4	8	18	7
x_2	11	9	5	14

Eigen values + Eigen vector

 $\lambda_1 = 30.3849 \quad e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$
 $\lambda_2 = 6.6151 \quad e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

Step 1) Organise as matrix

 $X = \begin{bmatrix} 4 & 8 & 18 & 7 \\ 11 & 9 & 5 & 14 \end{bmatrix}$

Step 2) mean-center the data

 $\bar{x}_1 = \frac{4+8+18+7}{4} = 8.5$
 $\bar{x}_2 = \frac{11+9+5+14}{4} = 9.5$
 $X_{centered} = \begin{bmatrix} 4-8 & 8-8 & 18-8 & 7-8 \\ 11-9 & 9-9 & 5-9 & 14-9 \end{bmatrix}$
 $= \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2 & 0 & -4 & 5 \end{bmatrix}$

(Step 3) use PC1 (largest eigenvalue)

 $\lambda_1 = 30.3849 \quad e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$

Step 4) Project the Data onto the first PC

 $x^t = e_1^t \cdot X_{centered}$
 $z_1 = (-4)(0.5574) + (2)(-0.8303) = -2.296 - 2.0758 = -4.3724$
 $z_2 = (0)(0.5574) + (-4)(-0.8303) = 3.364$
 $z_3 = (5)(0.5574) + (5)(-0.8303) = 5.693$
 $z_4 = (-1)(0.5574) + (5)(-0.8303) = -5.181$

Final 1D Transformed Data

Example First PC

1	-4.3724
2	3.364
3	5.693
4	-5.181

model Accuracy Before PCA Accuracy after PCA

SVM	1.00	1.00
Logistic Regression	1.00	1.00
Random Forest	1.00	1.00

No. of PCA components used: 1

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.decomposition import PCA
```

```

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

binary_cols = ['Sex', 'ExerciseAngina']
multi_cols = ['ChestPainType', 'RestingECG', 'ST_Slope']

le = LabelEncoder()
for col in binary_cols:
    df[col] = le.fit_transform(df[col])

df = pd.get_dummies(df, columns=multi_cols)

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

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

models = {
    'SVM': SVC(random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42)
}

print("Model Performance WITHOUT PCA:")
best_acc = 0
best_model = None

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    acc = accuracy_score(y_test, y_pred)
    print(f'{name}: Accuracy = {acc:.4f}')
    if acc > best_acc:
        best_acc = acc
        best_model = name
if name == 'Random Forest':
    print("Confusion Matrix (Random Forest):")
    print(confusion_matrix(y_test, y_pred))

pca = PCA(n_components=0.95, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("\nModel Performance WITH PCA:")

```

```
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    acc = accuracy_score(y_test, y_pred)
    print(f'{name} (PCA): Accuracy = {acc:.4f}')

print(f'\nBest model without PCA: {best_model} (Accuracy: {best_acc:.4f})')
print(f'Number of PCA components: {pca.n_components_}')
```