

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



LAB REPORT
on
Machine Learning (23CS6PCMAL)

Submitted by

Shivaraj Kallappa Pujari (1BM22CS259)

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 , who is bonafide student **Shivaraj Kallappa Pujari(1BM22CS259)** 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. Shravya AR 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 | 2 |
| 2 | 3-3-2025 | Demonstrate various data pre-processing techniques for a given dataset | 8 |
| 3 | 10-3-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 12 |
| 4 | 17-3-2025 | Build Logistic Regression Model for a given dataset | 18 |
| 5 | 24-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample | 24 |
| 6 | 7-4-2025 | Build KNN Classification model for a given dataset | 32 |
| 7 | 21-4-2025 | Build Support vector machine model for a given dataset | 37 |
| 8 | 5-5-2025 | Implement Random forest ensemble method on a given dataset | 44 |
| 9 | 5-5-2025 | Implement Boosting ensemble method on a given dataset | 49 |
| 10 | 12-5-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file | 55 |
| 11 | 12-5-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method | 59 |

Github Link:

https://github.com/Shivarajpujari2004/1BM22CS259_ML-LAB

Program 1

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

Screenshot:

5/3/23
Bafna Gold
Tutor Paper

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

* To Do: →

1) Method 1: Initializing values directly into DataFrame

```
# insert your known values, five rows of data with
# column readings as "USN", "Name", "Marks"
→ import pandas as pd
data = {'Name': 'USN': ['1BM22CS259', '1BM22CS230', '1BM
    22CS231', '1BM22CS232', '1BM22CS233'],
    'Name': ['Shiv', 'Arsh', 'Salman', 'Virat', 'Jack'],
    'Marks': [99, 90, 89, 69, 42]}
```

df = pd.DataFrame(data)

```
print("Sample data:")
print(df.head(1))
```

2) Method 2: Importing datasets from sklearn.datasets

```
# Loading diabetes datasets
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
```

→ 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(1))

3) Method 3: Importing datasets from a specific.csv file

```
# Sample-sales-data.csv
```

```
→ import pandas as pd  
file_path = 'sales.csv'  
df = pd.read_csv(file_path)  
print("sample data:")  
print(df.head(2))
```

4) Method 4: Downloading datasets from existing
dataset repositories like kaggle, UCI, KEEL etc
Download diabetes dataset from Mendeley

```
→ df = pd.read_csv('diabetes.csv')  
print(df.head(1))
```

* To Do :-

1) HDFC Bank Ltd., ICICI Bank Ltd, Kotak
Mahindra Bank Ltd.

tickers = ["HDFCBANK.NS", "ICICI.NS", "KOTAK.NS"]

→ import pandas as pd

import yfinance as yf

import matplotlib.pyplot as plt

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

2) Start date : 2024-01-01 , End date : 2024-12-30

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

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

→ print(data.head())

print(data.shape)

print(data.columns)

hdfc_bank_data = data['HDFCBANK.NS']

print("HDFCBANK Statistics")

print(hdfc_bank_data.describe())

hdfc_bank_data['Daily Returns'] = hdfc_bank_data['Close'].
pct_change()

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

plt.subplot(2,1,1)

hdfc_data['Close'].plot(title='Reference HDFC Bank -
closing price')

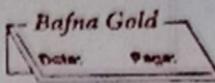
```
plt.subplot(2,1,1)
hdfc-data['Daily Return'].plot(title = "HDFC Bank -  
daily return", color = 'orange')
plt.tight_layout()
plt.show()

hdfc-data.to_csv('HDFC.csv')
print("HDFC bank data saved to 'hdfc.csv'.")
```

```
icibank-data = data['ICIBANK.NS']
print("ICICI stats")
print(icibank-data.describe())
icibank-data['Daily Return'] = icibank-data['close']
                           .pct_change()
icidata['Daily Return'] = icidata
plt.figure(12.6)
plt.subplot(2,1,1)
icidata['close'].plot(title = "ICICI")
plt.subplot(2,1,2)
icidata['Daily Return'].plot(title = "ICICI - DR",
                             color = 'orange')
plt.tight_layout()
plt.show()

icidiata.to_csv('icici-data.csv')
```

```
Kotak = data['KOTAKBANK.NS']
print("Kotak stats")
print(kotak.describe())
kotak-data['Daily Return'] = kotak['close'].pct_change()
```



```
plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
kotak['lose'].plot(title = "Kotak Bank")
plt.subplot(2,1,2)
kotak['Daily Return'].plot(title = "Kotak-DR",
                           color = 'orange')

plt.tight_layout()
plt.show()

kotak_data.to_csv('Kotak.csv')

print("Basic Kotak data saved")
```

dr2853125

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="2024-01-01", end="2024-12-30",
group_by='ticker')

hdfc_data = data['HDFCBANK.NS']
print("\nSummary statistics for HDFC bank :")
print(hdfc_data.describe())

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="Reliance Industries - Daily
Returns", color='orange')
plt.tight_layout()
plt.show()

icici_data = data['ICICIBANK.NS']
print("\nSummary statistics for ICICI bank :")
print(icici_data.describe())

icici_data['Daily Return'] = icici_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
icici_data['Close'].plot(title="ICICI bank - Closing Price")
plt.subplot(2, 1, 2)
icici_data['Daily Return'].plot(title="ICICI Bank - Daily Returns",
color='orange')
plt.tight_layout()
plt.show()
```

```
kotak_data = data['KOTAKBANK.NS']
print("\nSummary statistics for KOTAK bank :")
print(kotak_data.describe())

kotak_data['Daily Return'] = kotak_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
kotak_data['Close'].plot(title="KOTAK bank - Closing Price")
plt.subplot(2, 1, 2)
kotak_data['Daily Return'].plot(title="KOTAK Bank - Daily Returns",
color='orange')
plt.tight_layout()
plt.show()
```

Program 2

Screenshot:

Lab -1

Qn1 Work a program code ,filename as "housing.csv"

- i) To load .csv file into dataframe
- ii) To display information of all columns
- iii) To display statistical information of all numerical
- iv) To display the count of unique labels for "Ocean proximity" column
- v) To display which attributes in dataset have missing values count greater than zero

→ import pandas as pd
file-name = "housing.csv"
df = pd.read_csv(file-name)
print(df.info())
print(df.describe())
print(df['Ocean proximity'].count())
missing_values = df.isnull.sum()
columns with missing = missing-values[missing-values>0]
print(columns with missing)

Qn2 Work python code implement following data processing techniques for diabetes dataset.

- 1) Data Cleaning : Handling missing values, Handling categorical data , Handling outliers
- 2) Data transformations :- Min - Max Scaler / Normalized, standard Scaler

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler,
StandardScaler, LabelEncoder
diabetes = pd.read_csv('diabetes.csv')
diabetes.drop(columns=['ID', 'No_Patent'], inplace=True)
diabetes.dropna(inplace=True)
diabetes[['Gender', 'CLASS']] = diabetes[['Gender', 'CLASS']].apply(LabelEncoder().fit_transform)
diabetes = diabetes[(np.abs((diabetes.select_dtypes(include=[np.number]) - diabetes.mean()) / diabetes.std()) < 3).all(axis=1)]
```

```
scaler = MinMaxScaler()
diabetes_scaled = pd.DataFrame(scaler.fit_transform(diabetes.drop(['CLASS'], axis=1)), columns=diabetes.columns[:-1])
```

```
diabetes_scaled['CLASS'] = diabetes['CLASS'].values
print(diabetes_scaled.head())
```

① 7/12/25

Code:

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder

# Load dataset

diabetes = pd.read_csv('diabetes.csv') # Replace with actual file path

diabetes.drop(columns=['ID', 'No_Pation'], inplace=True) # Drop unnecessary columns

diabetes.dropna(inplace=True) # Handle missing values

diabetes[['Gender', 'CLASS']] = diabetes[['Gender', 'CLASS']].apply(LabelEncoder().fit_transform) # Encode categorical data

diabetes = diabetes[(np.abs((diabetes.select_dtypes(include=[np.number]) - diabetes.mean()) / diabetes.std()) < 3).all(axis=1)] # Remove outliers

scaler = MinMaxScaler()

diabetes_scaled =
pd.DataFrame(scaler.fit_transform(diabetes.drop(columns=['CLASS'])), columns=diabetes.columns[:-1])

diabetes_scaled['CLASS'] = diabetes['CLASS'].values # Add target column back
```

```
print(diabetes_scaled.head())
```

Program 3

Screenshot:

19/3/25 Lab-3

On: LINEAR REGRESSION :-

→ Type-1 (Independent & Dependent variable) (Slope method)

→ Using $y = mx + c$ or $y = a + bx$ formula to find the slope & intercept for dataset given below:-

| X (weeks) | Y (Sales in thousands) |
|-----------|------------------------|
| 1 | 1 |
| 2 | 3 |
| 3 | 4 |
| 4 | 8 |

* Code:-

```
→ import pandas as pd  
data = pd.read_csv('IContent/Sales.csv')  
x = data['X(weeks)']  
y = data['Y(Sales)']  
x_mean = x.mean()  
y_mean = y.mean()  
numerator = ((x - x_mean) * (y - y_mean)).sum()  
denominator = ((x - x_mean) ** 2).sum()  
b1 = numerator / denominator  
b0 = y_mean - (b1 * x_mean)  
print("Slope (b1):", b1)  
print("Intercept (b0):", b0)  
x_new = 5  
y_predicted = b0 + (b1 * x_new)  
print("Predicted value for x=5:", y_predicted)
```

→ Output:

Slope(b1): 2.2
Intercept(b0): -1.5
54 5G Predicted value for x=5: 9.5

Q) Type - 2: Matrix Form

$$\text{Slope} = \alpha_1 = ((X^T X)^{-1} X^T) Y$$

$$Y = \alpha_0 + \alpha_1 X$$

* Code:-

\rightarrow import numpy as np

import pandas as pd

data = pd.read_csv('contents/Sales.csv')

X = data[['X(Wkly)']]

Y = data[['Y(Sales)']]

X = X.values

Y = Y.values

X = np.ones((len(X), 2))

X[:, 1] = X

Y = Y.reshape(-1, 1)

theta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)

b0 = theta[0][0]

b1 = theta[1][0]

~~print("Slope (b1):", b1)~~

~~print("Intercept(b0):", b0)~~

x-new = 5

~~y-predicted = b0 + b1 * x-new~~

~~print("Predicted value for x=5:", y-predicted)~~

\rightarrow Output:

Slope(b1) = 2.200

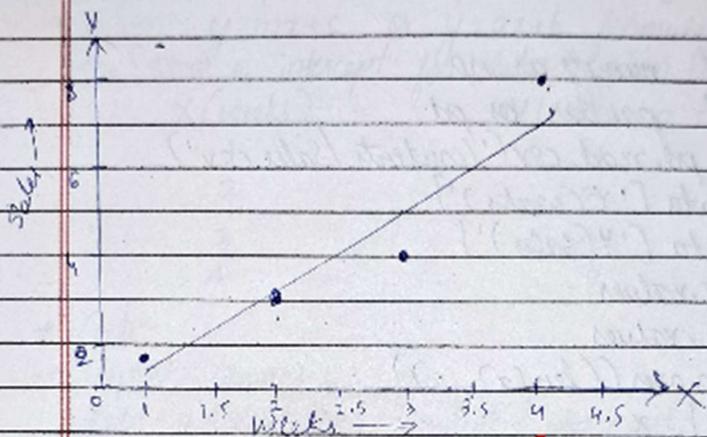
Intercept(b0) = -1.5

Predicted value for x=5 : 9.5000000004

* Plot the graph:

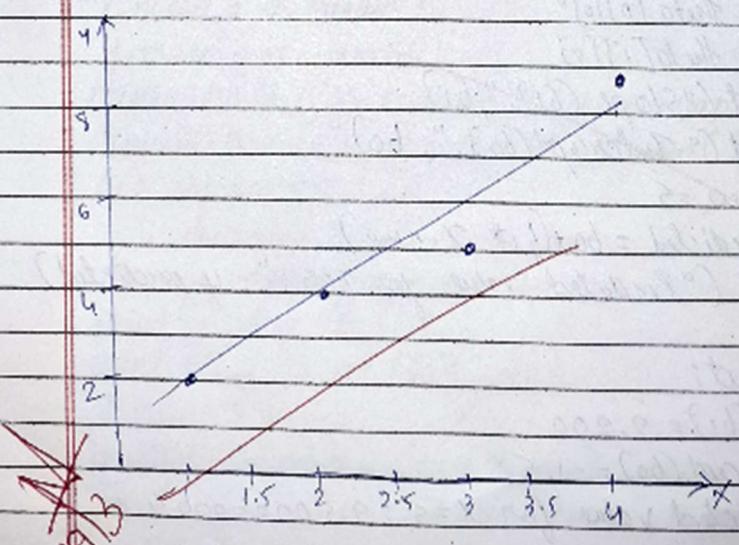
$$1) X = 1, 2, 3, 4$$

$$Y = 1, 3, 4, 8$$



$$2) X = 1, 2, 3, 4$$

$$Y = 2, 4, 5, 9$$



xy F54 5G

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


# Load the dataset from CSV file

dataset = pd.read_csv('/content/sales.csv')


# Display the first few rows of the dataset

print(dataset.head())


# Assuming the CSV has columns 'Week' and 'Sales' (adjust based on your
actual column names)

weeks = dataset['xi(week)'].values

sales = dataset['yi(Sales in thousands)'].values


# Reshaping weeks for matrix operations (make it a column vector)

X = weeks.reshape(-1, 1)

y = sales.reshape(-1, 1)


# Add a column of ones to X to account for the intercept (b)

X_b = np.c_[np.ones((len(X), 1)), X] # The "1" is for the bias term
(intercept)
```

```

# Compute theta using the normal equation:  $\theta = (X^T X)^{-1} X^T y$ 

theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)

# Extract slope (m) and intercept (b) from theta

b = theta[0]

m = theta[1]

# Print the regression line equation

print(f"The regression equation is: y = {m[0]:.2f}x + {b[0]:.2f}")

# Predict the sales for 7th and 9th weeks

week_7 = 7

week_9 = 9

predicted_sales_7 = m * week_7 + b

predicted_sales_9 = m * week_9 + b

print(f"Predicted sales for the 7th week: {predicted_sales_7[0]:.2f} thousand")

print(f"Predicted sales for the 9th week: {predicted_sales_9[0]:.2f} thousand")

```

```
# Plot the data points and the regression line

plt.scatter(weeks, sales, color='blue', label='Data Points')

plt.plot(weeks, m * weeks + b, color='red', label=f'Linear Regression: y = {m[0]:.2f}x + {b[0]:.2f}')

plt.xlabel('Weeks')

plt.ylabel('Sales (in thousands)')

plt.title('Sales Data and Linear Regression')

plt.legend()

plt.show()
```

Program 4

Screenshot:

Lab-4

Bafna Gold
Dinner Dinner.

1) Logistic Regression for Binary Classification

Qn (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 reduced & its learned parameters are $a_0 = -5$ (intercept) & $a_1 = 0.8$ (coefficient for study hours).

→ Write a logistic regression equation for this problem

→ $p(\text{pass}) = \frac{1}{1 + e^{-(a_0 + a_1 x)}}$

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

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

$$= \frac{1}{1 + e^{-0.6}}$$

$$= \frac{1}{1 + 0.548} \approx 0.645$$

∴ It is approximately 64.5%.

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

→ Since $p(\text{pass}) = 0.645$ which is greater than threshold, therefore predicted class is "Pass".

2) Consider $z = [9, 1, 0]$ for 3 classes. Apply softmax function to find the probability values of 3 classes.

→ $P_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$

always calculating probabilities for each class:

$$P_1 = \frac{C^2}{11.107} = \frac{7.389}{11.107} \approx 0.665$$

$$P_2 = \frac{C'}{11.107} = \frac{2.718}{11.107} \approx 0.245$$

$$P_3 = \frac{C''}{11.107} = \frac{1}{11.107} \approx 0.090$$

• Q3 Multiclass Classification:

i) If for dataset file "HR-comma-separated.csv"

ii) Which variables did you identify as having a direct and clear impact on employee retention? Why?

→ satisfaction-level → lower satisfaction increases attrition.

* number-project and avg.monthly-hours → Overworking leads to burnout.

* promotion.last.year and salary → lack of growth opportunities impacts retention.

iii) What was the accuracy of your model? Do you think this is a good accuracy? Why or why not?

→ Accuracy → 76.48%

→ It is good since it captures key patterns in relationships.

28 For zoo dataset

Did you perform any data preprocessing steps?
If yes, what were they and why were they necessary?

iv) Were there any missing or inconsistent values in the dataset? How did you handle them?

iii) What does the confusion matrix tell you about the performance of your model?

iv) Which class types were most frequently misclassified? Why do you think this happened?

→ v) Data preprocessing:

- * Removed animal name
- * Standardized numerical features
- * split dataset (80% train, 20% test)

vi) Handling missing data

- no missing or inconsistent values found.

vii) Confusion matrix insights

- Achieved 100% accuracy, no misclassifications.

viii) Misclassified class types

- None due to well-separated data features.

Q
a) what's

Code:

```
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 dataset

file_path = "/content/HR_comma_sep"

df = pd.read_csv(file_path)

# Exploratory Data Analysis

# Plot bar chart showing impact of salaries on retention

plt.figure(figsize=(8, 5))

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

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

plt.xlabel("Salary Level")

plt.ylabel("Number of Employees")

plt.legend(["Stayed", "Left"])

plt.show()
```

```

# Plot bar chart showing correlation between department and employee
retention

plt.figure(figsize=(10, 5))

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

plt.title("Department-wise Employee Retention")

plt.xlabel("Department")

plt.ylabel("Number of Employees")

plt.xticks(rotation=45)

plt.legend(["Stayed", "Left"])

plt.show()

# Selecting key features based on analysis

X = df[['satisfaction_level', 'time_spend_company', 'Work_accident',
'salary']]

X = pd.get_dummies(X, columns=['salary'], drop_first=True) # Convert
categorical 'salary' to numerical

y = df['left']

# Splitting data into training and testing sets

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

# Train logistic regression model

model = LogisticRegression(max_iter=1000)

```

```

model.fit(X_train, y_train)

# Predictions

y_predicted = model.predict(X_test)

# Model accuracy

accuracy = accuracy_score(y_test, y_predicted)

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

# Plotting actual vs predicted values

plt.figure(figsize=(6, 4))

sns.heatmap(pd.crosstab(y_test, y_predicted), annot=True, fmt='d',
cmap='Blues')

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

# Predict probability of an employee leaving

def predict_leave_probability(features):

    return model.predict_proba([features])[0][1] # Probability of leaving

# Example prediction

```

```

example_employee = [[0.5, 3, 0, 1, 0]] # Sample feature values with one-hot
encoded salary

probability = predict_leave_probability(example_employee[0])

print(f"Probability of leaving: {probability:.4f}")

```

Program 5

Screenshot:

Lab - 2

Qn: Use an appropriate dataset for building the decision tree (ID3) & apply this knowledge to classify a new sample.

```

→ import pandas as pd
import numpy as np
from collections import Counter
import math

def entropy(y):
    count = Counter(y)
    prob = [count[x] for x in count]
    return -sum(p * math.log2(p) for p in prob)

def info_gain(data, feature, target):
    total_entropy = entropy(data[target])
    values = data[feature].unique()
    weighted_entropy = sum([len(data[data[feature] == v]) / len(data) * entropy(data[data[feature] == v]) for v in values])
    return total_entropy - weighted_entropy

def id3(data, features, target):
    if len(set(data[target])) == 1:
        return data[target].iloc[0]
    if len(features) == 0:
        return data[target].mode().mode()
    gains = {feature: info_gain(data, feature, target) for feature in features}
    best_feature = max(gains, key=gains.get)

```

54 5G

```
tree = {best_feature: 2}  
return tree
```

```
def print_tree(tree, indent=" "):  
    if not isinstance(tree, dict):  
        print(indent + "->" + str(tree))  
        return  
    for key, value in tree.items():  
        print(indent + str(key))  
        for sub_key, sub_tree in value.items():  
            print(indent + "  " + str(sub_key))
```

```
path = 'C:/content/dennis.csv'  
data = pd.read_csv(path)  
features = list(data.columns[1:-1])  
target = 'play'  
decision_tree = id3(data, features, target)  
print_tree(decision_tree)
```

→ Output :

```
outlook
  + Sunny
    + humidity
      + high: no
      + normal: yes
    - overcast: yes
  - rainy
    - windy
      + False: yes
      - True: no
```

Ans 13

galaxy F54 5G

Code:

```
import pandas as pd

import numpy as np


# Sample weather dataset

data = {

    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain',
    'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast',
    'Rain'],

    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool',
    'Mild', 'Cool', 'Mild', 'Mild', 'Hot', 'Mild'],

    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal',
    'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],

    'Windy': ['No', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No',
    'Yes', 'Yes', 'No', 'Yes'],

    'Play Tennis?': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No',
    'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

}

# Convert to DataFrame

df = pd.DataFrame(data)

# Function to calculate entropy

def entropy(target):

    # Get the counts of each class
```

```

class_counts = target.value_counts()

# Calculate the entropy using the formula

probabilities = class_counts / len(target)

return -np.sum(probabilities * np.log2(probabilities))

# Function to calculate information gain

def information_gain(data, feature, target):

    # Calculate the entropy of the whole dataset

    entropy_before = entropy(target)

    # Get the unique values of the feature

    feature_values = data[feature].unique()

    # Calculate the weighted entropy after the split

    weighted_entropy = 0

    for value in feature_values:

        subset = target[data[feature] == value]

        weighted_entropy += (len(subset) / len(target)) * entropy(subset)

    # Information gain is the reduction in entropy

    return entropy_before - weighted_entropy

```

```

# Function to print entropy and information gain for each feature

def print_entropy_and_gain(data, features, target):

    print("\nEntropy and Information Gain for each feature:")

    for feature in features:

        gain = information_gain(data, feature, target)

        ent = entropy(target)

        print(f"Feature: {feature} | Entropy: {ent:.4f} | Information Gain: {gain:.4f}")

# Function to build the decision tree recursively

def build_tree(data, target, features):

    # Base case: If all target values are the same, return a leaf node

    if len(target.unique()) == 1:

        return target.iloc[0]

    # Base case: If no features left to split, return the majority class

    if len(features) == 0:

        return target.mode()[0]

    # Calculate information gain for each feature

    gains = {feature: information_gain(data, feature, target) for feature in features}

```

```

# Find the feature with the highest information gain

best_feature = max(gains, key=gains.get)

# Create the tree node with the best feature

tree = {best_feature: {}}

# Get the unique values of the best feature

feature_values = data[best_feature].unique()

# Recursively build the tree for each subset of the data

for value in feature_values:

    subset_data = data[data[best_feature] == value]

    subset_target = target[data[best_feature] == value]

    # Remove the best feature from the list of features for the next
    level

    remaining_features = [f for f in features if f != best_feature]

    # Build the subtree for the subset

    subtree = build_tree(subset_data, subset_target, remaining_features)

    # Add the subtree to the tree

    tree[best_feature][value] = subtree

```

```

return tree

# Function to print the tree in a visually structured way

def print_tree(tree, indent=""):

    if isinstance(tree, dict):

        for feature, branches in tree.items():

            print(f"{indent}{feature}:")

            for value, subtree in branches.items():

                print(f"  {value} ->, end=" ")

                print_tree(subtree, indent + "      ")

    else:

        print(f"{indent}{tree}")

# Target variable

target = df['Play Tennis?']

# Features

features = ['Outlook', 'Temperature', 'Humidity', 'Windy']

# Step 1: Print entropy and information gain for each feature

print_entropy_and_gain(df, features, target)

```

```
# Step 2: Build the decision tree

tree = build_tree(df, target, features)

# Step 3: Print the decision tree (formatted)

print("\nDecision Tree:")

print_tree(tree, indent="    ")
```

Program 6

Screenshot:

7/11/25

Lab-5

KNN

Classification

Q4 Consider the following dataset for k=3 to test data
(X, 35, 100) as (Person, Age, Salary) solve using
Knn classifier & predict the target.

| Person | Age | Salary | k | Target | Distance | Step |
|--------|-----|--------|---|--------|----------|------|
| A | 18 | 50 | N | | 52.8 | |
| B | 23 | 55 | N | | 46.6 | |
| C | 24 | 70 | N | | 31.9 | 2 |
| D | 41 | 60 | Y | | 40.4 | 3 |
| E | 43 | 70 | Y | | 31.1 | 1 |
| F | 38 | 40 | Y | | 60.1 | |
| X | 35 | 100 | ? | | | |

$$\rightarrow \text{Euclidean distance: } d = \sqrt{(Age_2 - Age_1)^2 + (Salary_2 - Salary_1)^2}$$

Step-1:

$$A \rightarrow \sqrt{(35-18)^2 + (100-50)^2} = 52.8$$

$$B \rightarrow 46.6$$

$$C \rightarrow 31.9$$

$$D \rightarrow 40.4$$

$$E \rightarrow 31.1$$

$$F \rightarrow 60.1$$

Step-2: Identify 3 nearest neighbours

1) E: (31.1, Y)

2) C: (31.9, N)

3) D: (40.4, Y)

Step-3: Majority voting :

\rightarrow Y

\rightarrow X(35, 100) is 'Y'

Qn: 1) For iris dataset

↳ How to choose the K value? Demonstrate using accuracy rate & error rate.

Qn: 2) For diabetes dataset.

What is the purpose of feature scaling? How to perform it?

→ 1) * Train KNN with different K values (e.g., 1 to 20)

* Measure accuracy (higher is better) & error rate (lower is better)

* The best K is where accuracy is highest & error rate is lowest (typically 5-10)

→ 2) * KNN relies on distance, so large scaling features can dominate the smaller ones.

* Standardization ensures all features contribute equally.

* Scaling improves accuracy & prevents biased predictions.

Q

9/4/25

Code:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

# Load the Iris dataset

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

X_iris = iris.drop(columns=['species'])

y_iris = iris['species']

# Split the dataset into training and testing sets

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 the best k value (testing k from 1 to 20)

k_values = range(1, 21)

accuracy_list = []
```

```

for k in k_values:

    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(X_train_iris, y_train_iris)

    y_pred = knn.predict(X_test_iris)

    accuracy_list.append(accuracy_score(y_test_iris, y_pred))

best_k_iris = k_values[np.argmax(accuracy_list)]

print(f"Optimal k value for Iris dataset: {best_k_iris}")

# Train KNN with the best k value

knn_iris = KNeighborsClassifier(n_neighbors=best_k_iris)

knn_iris.fit(X_train_iris, y_train_iris)

y_pred_iris = knn_iris.predict(X_test_iris)

# Display Accuracy and Confusion Matrix for Iris dataset

print("Iris Dataset Accuracy:", accuracy_score(y_test_iris, y_pred_iris))

print("Confusion Matrix for Iris:")

print(confusion_matrix(y_test_iris, y_pred_iris))

print("Classification Report for Iris:")

print(classification_report(y_test_iris, y_pred_iris))

# Load the Diabetes dataset

```

```

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

X_diabetes = diabetes.drop(columns=['Outcome'])

y_diabetes = diabetes['Outcome']

# Split into training and testing

X_train_diabetes, X_test_diabetes, y_train_diabetes, y_test_diabetes =
train_test_split(X_diabetes, y_diabetes, test_size=0.2, random_state=42)

# Feature Scaling

scaler = StandardScaler()

X_train_diabetes = scaler.fit_transform(X_train_diabetes)

X_test_diabetes = scaler.transform(X_test_diabetes)

# Train KNN Classifier for Diabetes dataset

best_k_diabetes = 7 # Assume 7 as a reasonable k value (can be tuned
further)

knn_diabetes = KNeighborsClassifier(n_neighbors=best_k_diabetes)

knn_diabetes.fit(X_train_diabetes, y_train_diabetes)

y_pred_diabetes = knn_diabetes.predict(X_test_diabetes)

# Display Accuracy and Confusion Matrix for Diabetes dataset

print("Diabetes Dataset Accuracy:", accuracy_score(y_test_diabetes,
y_pred_diabetes))

print("Confusion Matrix for Diabetes:")

```

```

print(confusion_matrix(y_test_diabetes, y_pred_diabetes))

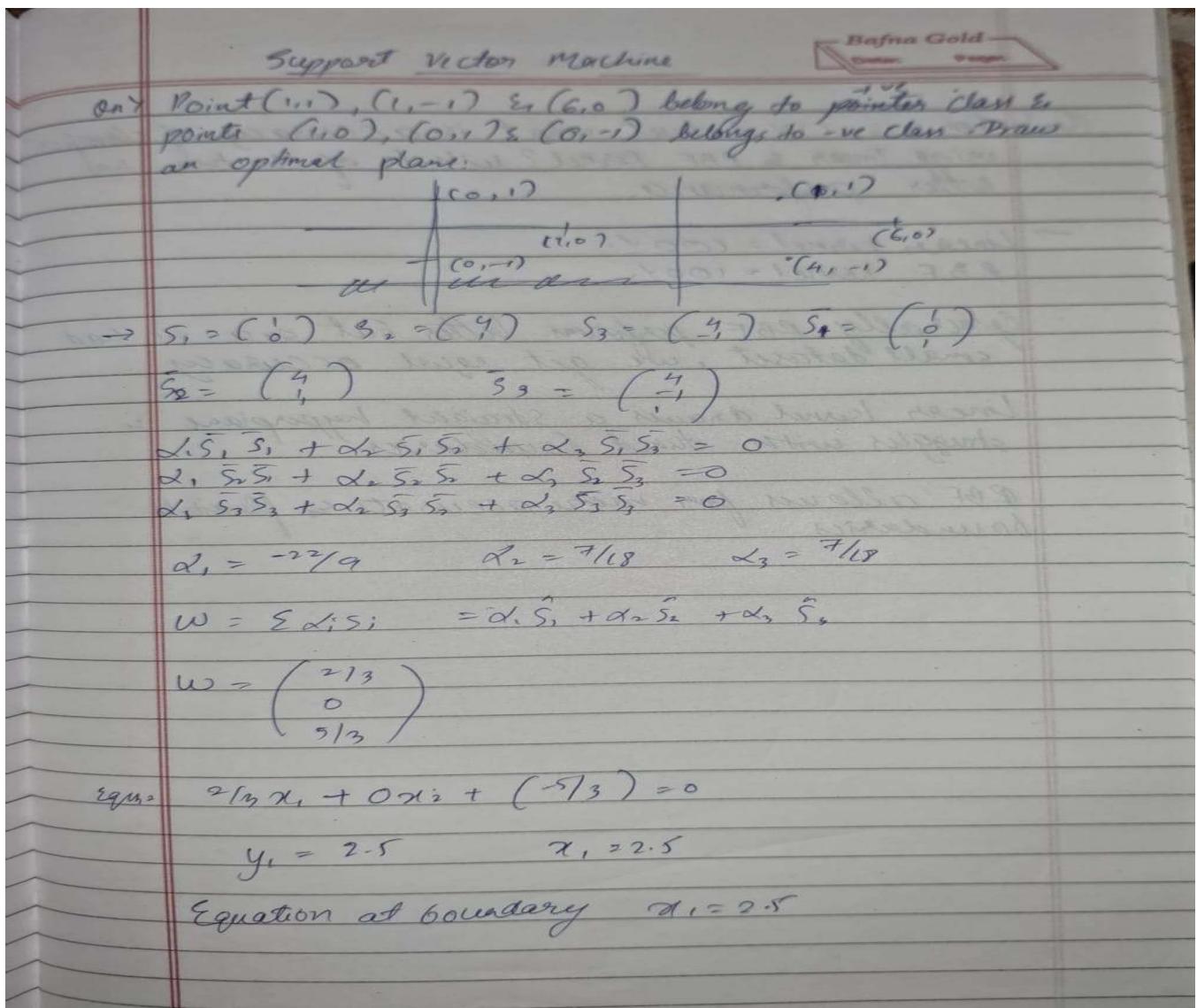
print("Classification Report for Diabetes:")

print(classification_report(y_test_diabetes, y_pred_diabetes))

```

Program 7

Screenshot:



On Iris .CSV dataset what is accuracy of classifier using linear & RBF kernel? Which of the two has better performance.

→ Linear kernel = 100%

RBF Kernel = 100%

Generally RBF performs better but since we had small dataset, we get equal accuracy.

Linear kernel draws a straight hyperplane & struggles with curved boundaries.

RBF allows for more accurate & flexible boundaries

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) # Convert labels to -1 and 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)

    # Plot existing data points

    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 "")

    # Plot decision boundary

    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--')

# Plot the new point

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__":

```

```

# Training data

X = np.array([
    [1, 0],
    [0, 1],
    [0, -1],
    [4, -1],
    [4, 1],
    [6, 0]
])

y = np.array([0, 0, 0, 1, 1, 1]) # 0 -> -1, 1 -> +1


# New point to classify

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


# Train and predict

svm = SVM()

svm.fit(X, y)

prediction = svm.predict(new_point)[0]


# Visualize

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

```

```
# Print prediction

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

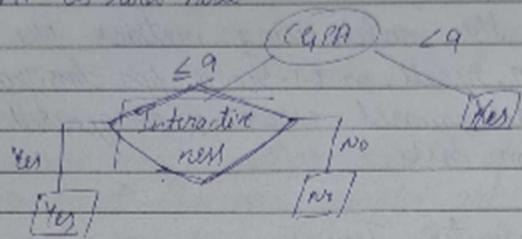
Program 8

Screenshot:

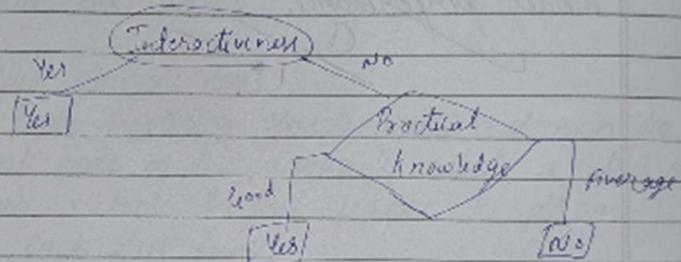
Lab - 2

Random Forest

* CGPA as root node:



* Interactionness as root node:



? For "iris.csv" dataset

Best - accuracy :

Accuracy Score = 0.973 or 97.3%.

Confusion matrix : $\begin{bmatrix} 16 & 0 & 0 \\ 0 & 14 & 1 \\ 0 & 0 & 14 \end{bmatrix}$

Classified 44 out of 45 test samples correctly
with 100 trees.

54 5G

Code:

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns


# Load dataset

df = pd.read_csv("/content/iris (4).csv")



# 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. Default Random Forest with 10 estimators

# -----



clf_default = RandomForestClassifier(n_estimators=10, random_state=42)

clf_default.fit(X_train, y_train)

y_pred_default = clf_default.predict(X_test)
```

```

default_accuracy = accuracy_score(y_test, y_pred_default)

default_cm = confusion_matrix(y_test, y_pred_default)

print("==== Default Model (10 trees) ====")

print(f"Accuracy: {default_accuracy:.4f}")

print("Confusion Matrix:")

print(default_cm)

# -----
# 2. Fine-tune number of trees for best accuracy

# -----
best_score = 0

best_n = 0

best_model = None

best_cm = None

scores = []

n_estimators_range = range(1, 101)

for n in n_estimators_range:

    clf = RandomForestClassifier(n_estimators=n, random_state=42)

    clf.fit(X_train, y_train)

```

```

y_pred = clf.predict(X_test)

score = accuracy_score(y_test, y_pred)

scores.append(score)

if score > best_score:

    best_score = score

    best_n = n

    best_model = clf

    best_cm = confusion_matrix(y_test, y_pred)

# Print best result

print("\n==== Tuned Model ====")

print(f"Best Accuracy: {best_score:.4f}")

print(f"Best Number of Trees: {best_n}")

print("Confusion Matrix:")

print(best_cm)

# Plot accuracy vs number of trees

plt.figure(figsize=(8, 5))

plt.plot(n_estimators_range, scores, marker='o')

plt.xlabel("Number of Trees")

plt.ylabel("Accuracy")

```

```
plt.title("Random Forest Accuracy vs Number of Trees")

plt.grid(True)

plt.show()

# Plot best confusion matrix

plt.figure(figsize=(6, 4))

sns.heatmap(best_cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=y.unique(), yticklabels=y.unique())

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Best Confusion Matrix")

plt.show()
```

Program 9

Screenshot:

A Implement Boosting ensemble method on a given dataset.

| C4PP | Actual Class | Job profile | Initial wt | Updated wt |
|----------|--------------|-------------|------------|------------|
| ≥ 9 | Yes | Yes | $1/6$ | 0.1249 |
| < 9 | No | Yes | $1/6$ | 0.2501 |
| ≥ 9 | Yes | No | $1/6$ | 0.2501 |
| < 9 | No | No | $1/6$ | 0.1249 |
| ≥ 9 | Yes | Yes | $1/6$ | 0.1249 |
| ≥ 9 | Yes | Yes | $1/6$ | 0.1249 |

Step-1: Assign initial weight = $1/6$

$$\text{Step-2: } \partial_t \cdot \Sigma_i = \sum_{j=1}^6 H_j(d_j) w_j(t(d_j))$$

weighted error E_{C4PP}

$$E_{\text{C4PP}} = 2 \times \frac{1}{6} = 0.333$$

b) compute weight of each weak classifier:

$$d_{\text{weak}} = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$

$$= \frac{1}{2} \ln \left(\frac{1 - 0.33}{0.33} \right)$$

$$d_{\text{C4PP}} = 0.347$$

c) calculate the normalizing factor

$$Z_{\text{C4PP}} = \frac{1}{6} \times 4 \times e^{-0.347} + \frac{1}{6} + \frac{1}{2} \times e^{+0.347}$$

$$Z_{\text{C4PP}} = 6.9428$$

d) Update the weight of all data instances

$$w_t(c_j)_i = \underline{w_t(c_j)_i} + \text{count of correct instance} \cdot e^{-\epsilon c_j p_i}$$

$$= \frac{1}{16} + e^{-0.344} \cdot 7 \cdot 0.9428$$

$$= 0.1249$$

$$= \frac{1}{16} + e^{0.344} \cdot 0.9428$$

$$= 0.2501$$

→ For "in闯. CSV" dataset

Test Accuracy score : 0.86 or 86%

Confusion matrix :

| TN | FP |
|------|------|
| 6873 | 1129 |
| FN | TP |
| 845 | 1328 |

F54 5G

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns

# Load dataset

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

# Feature and target split

X = df.iloc[:, :-1]

y = df.iloc[:, -1]

# Handle categorical variables (if any)

X = pd.get_dummies(X)

y = pd.factorize(y)[0]

# Train/test split

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

```

# -----
# 1. Default AdaBoost model (10 estimators)

# -----

clf_default = AdaBoostClassifier(n_estimators=10, random_state=42)

clf_default.fit(X_train, y_train)

y_pred_default = clf_default.predict(X_test)

default_accuracy = accuracy_score(y_test, y_pred_default)

default_cm = confusion_matrix(y_test, y_pred_default)

print("==== Default AdaBoost Model (10 estimators) ====")

print(f"Accuracy: {default_accuracy:.4f}")

print("Confusion Matrix:")

print(default_cm)

# -----
# 2. Fine-tune n_estimators

# -----



best_score = 0

best_n = 0

best_cm = None

scores = []

```

```

for n in range(1, 101):

    clf = AdaBoostClassifier(n_estimators=n, random_state=42)

    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    score = accuracy_score(y_test, y_pred)

    scores.append(score)

    if score > best_score:

        best_score = score

        best_n = n

        best_cm = confusion_matrix(y_test, y_pred)

print("\n==== Best AdaBoost Model ====")

print(f"Best Accuracy: {best_score:.4f}")

print(f"Best Number of Estimators: {best_n}")

print("Confusion Matrix:")

print(best_cm)

# Plot accuracy vs number of estimators

plt.figure(figsize=(8, 5))

plt.plot(range(1, 101), scores, marker='o')

plt.xlabel("Number of Estimators")

```

```
plt.ylabel("Accuracy")

plt.title("AdaBoost Accuracy vs Number of Estimators")

plt.grid(True)

plt.show()

# Plot best confusion matrix

plt.figure(figsize=(6, 4))

sns.heatmap(best_cm, annot=True, fmt="d", cmap="Blues")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Best Confusion Matrix")

plt.show()
```

Program 10

Screenshot:

Lab - 9

K-Means Algorithm

Bafna Gold
Dinner Paper.

* For the given data, compute two clusters using K-means algorithm for clustering, where initial cluster centers are (1.0, 1.0) & (5.0, 7.0)

| Record No. | A | B | C ₁ | C ₂ | Assigned Cluster |
|----------------|-----|------|----------------|----------------|------------------|
| R ₁ | 1.0 | 0.0 | 7.21 | | C ₁ |
| R ₂ | 2.0 | 1.12 | 6.10 | | C ₁ |
| R ₃ | 3.0 | 4.0 | 4.24 | | C ₁ |
| R ₄ | 5.0 | 7.0 | 4.21 | 6.0 | C ₂ |
| R ₅ | 5.5 | 5.0 | 4.72 | 2.5 | C ₂ |
| R ₆ | 4.5 | 5.0 | 5.32 | 2.0 | C ₂ |
| R ₇ | 3.5 | 4.5 | 4.30 | 2.5 | C ₂ |

Cluster 1 : R₁, R₂, R₃

Cluster 2 : R₄, R₅, R₆, R₇

Step 2: Recompute cluster centers

$C_1 = (x, y) \quad x = (1 + 1.5 + 3) / 3 = 1.83$

$y = (1 + 2 + 4) / 3 = 2.33$

New $C_1 = (1.83, 2.33)$

New $C_2 = (4.13, 5.38)$

$x = (5 + 3.5 + 4.5 + 3.5) / 4 = 4.13$

$y = (7 + 5 + 5 + 4.5) / 4 = 5.38$

xy F54 5G

+ Iteration - 2 : $(1.33, 0.33)$ $(4.13, 5.33)$ Assigned
 Record A B C₁ C₂ Clust_y
 N_o

| | A | B | C ₁ | C ₂ | Clust _y |
|----------------|-----|-----|----------------|----------------|--------------------|
| R ₁ | 1.6 | 1.0 | 1.37 | 5.62 | C ₁ |
| R ₂ | 1.5 | 2.0 | 0.47 | 4.53 | C ₁ |
| R ₃ | 3.0 | 4.0 | 2.03 | 1.92 | C ₂ |
| R ₄ | 5.0 | 7.0 | 5.57 | 1.89 | C ₂ |
| R ₅ | 3.5 | 5.0 | 2.63 | 0.71 | C ₂ |
| R ₆ | 6.5 | 5.0 | 3.25 | 0.47 | C ₂ |
| R ₇ | 3.5 | 4.5 | 2.73 | 0.94 | C ₂ |

$$C_1 = R_1, R_2$$

$$C_2 = R_3, R_4, R_5, R_6, R_7$$

* Final Centroid

$$C_1 = (1.25, 1.5)$$

$$C_2 = (3.9, 5.1)$$

Code:

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import numpy as np


# Load dataset

df = pd.read_csv("iris (4).csv") # Replace with your actual filename if needed


# Use only petal length and petal width

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


# Feature scaling

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_)

# Automatically determine the elbow point (optional)

diff = np.diff(inertia)

diff_r = np.diff(diff)

optimal_k = np.argwhere(diff_r > -0.1)[0][0] + 2 # add 2 due to second
derivative shift

# Plot elbow graph with optimal K marked

plt.figure(figsize=(8, 5))

plt.plot(k_range, inertia, 'bo-')

plt.axvline(x=optimal_k, color='red', linestyle='--', label=f'Optimal k = {optimal_k}')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Inertia')

plt.title('Elbow Method For Optimal k')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()

# Output the optimal k

```

```
print(f"Optimal number of clusters (k): {optimal_k}")
```

Program 11

Screenshot:

Cat -10

Principle Component Analysis (PCA) *Bafna Gold*

* Reduce the dimension from 2 to 1 using the principal component (PCA). Compute the first PC.

| Feature | Ex 1 | Ex 2 | Ex 3 | Ex 4 |
|---------|------|------|------|------|
| X_1 | 4 | 8 | 13 | 7 |
| X_2 | 11 | 4 | 5 | 14 |

* Eigen values : $\lambda_1 = 30.3849$
 $\lambda_2 = 6.6151$

* Eigen vectors : $e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$
 $e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

Step 2: Mean for $X_1 = \frac{4+8+13+7}{4} = 8$

Mean for $X_2 = \frac{11+4+5+14}{4} = 8.5$

$X_{\text{centered}} = \begin{bmatrix} 4-8 & 8-8 & 13-8 & 7-8 \\ 11-8.5 & 4-8.5 & 5-8.5 & 14-8.5 \end{bmatrix}$
 $= \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}$

Step 3: Largest Eigen value = λ_1
Corresponding eigen vector = $e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$

Step 4: $Z = e_1^T \cdot X_{\text{centered}}$
4 5G $Z = [0.5574 \quad -0.8303] \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}$

$$Z_1 = (0.8574)(-4) + (-0.5303)(2.5)$$

$$= 1.5385$$

$$Z_1 = 1.5385$$

$$Z_2 = 3.73635$$

$$Z_3 = 0.11905$$

$$Z_4 = -4.00925$$

✓
15385

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 (1).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)

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("\nAccuracy 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]}")
```