VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Abhishek Yadav (1BM22CS009)

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
Feb-2025 to June-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 **Abhishek Yadav (1BM22CS009),** 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.

Dr. Seema Patil Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link:

https://github.com/AbhishekCS22/6A-ML-LAB-Batch1

Program 1

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

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
df=pd.read_csv('/content/Dataset of Diabetes .csv')
df.head()
df.shape
print(df.info())
# Summary statistics
print(df.describe())
missing_values=df.isnull().sum()
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical columns identified:", categorical_cols)
if len(categorical_cols) > 0:
    df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
    print("\nDataFrame after one-hot encoding:")
    print(df.head())
else:
    print("\nNo categorical columns found in the dataset.")
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import pandas as pd

numerical_cols = df.select_dtypes(include=['number']).columns

scaler = MinMaxScaler()
df_minmax = df.copy()  # Create a copy to avoid modifying the original
df_minmax[numerical_cols] = scaler.fit_transform(df[numerical_cols])

scaler = StandardScaler()
df_standard = df.copy()
df_standard[numerical_cols] = scaler.fit_transform(df[numerical_cols])
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())
print("\nDataFrame after Standardization:")
print(df_standard.head())
df1=pd.read_csv('/content/adult.csv')
df1.head()

from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

```
import pandas as pd

numerical_cols = df1.select_dtypes(include=['number']).columns

scaler = MinMaxScaler()
df_minmax = df1.copy()  # Create a copy to avoid modifying the original
df_minmax[numerical_cols] = scaler.fit_transform(df1[numerical_cols])

scaler = StandardScaler()
df_standard = df1.copy()
df_standard[numerical_cols] = scaler.fit_transform(df1[numerical_cols])
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())
print("\nDataFrame after Standardization:")
print(df_standard.head())
```

Observation:

Date: Page: 1

Date - 4/3/2025

ML LAB-1

1. Write python code, consider filename as "housing.csv"

import pandas as pd

i # load csv file into the Dataframe
filename = 'housing.csv'
housing_data = pd.read_csv(filename)

ii # Display information of all columns
print ("Information of all columns:")
print (housing_data.info())

iii # Display statistical information of all numerical columns
print ("\n Statistical information of all numerical columns:")
print (housing_data.describe())

iv # Display the count of unique labels for 'ocean_proximity' column:")
print ("\n Count of unique labels for 'ocean proximity' column:")
print (housing_data['ocean_proximity'].value_counts()

v # Display which attributes (columns) have missing values count greater than zero
print ("\n column with missing values count greater than zero:")

Bafna Gold

Date: Page:

```
missing_values = housing_data.isnull().sum()
missing_columns = missing_values[missing_values>0]
print(missing_column)
```

Output:

1. Information of all columns:
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 colums):

#	column	Non-Null count	Dtype
0	longitude	20640 non-null	float64

dtype: float64(9), object(1)
memory usage: 1.6+ MB

2. Statistical information of all numerical column:

count	longitude	latitud	housing_median
	20640	20640	20640

	total_bedroom	population	household
	20433	20640	20640

	total_rooms	media_income	median_value
	20640	20640	20640

3. count of unique labels for 'ocean proximity' column: ocean proximity

<1th ocean	9136	ISLAND	5
INLAND	6551		
Near Ocean	2658		
Near Bay	2290		

Bafna Gold

Program 2

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

Observation:

→ Data preprocessing
code Implemented
Answer the following questions for the two
dataset

Bafna Gold

Date: Page: 3

1) Diabetes.csv
a) which column in the dataset had missing values?
How to handle them?
→ missing values are present in numerical columns
if present, which are replaced by the mean of the
respective column
b) which categorical column did you identify? How
did you encode them?
→ No categorical data, do encoding
c) what is difference between MinMax scaling and
Standardization. when would u use one over the
other?
→ MinMax scaling transform data to a fixed range
[0,1] using
$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$ Standardization: transform data to
have non zero mean & unit variance
$x' = \frac{x - \mu}{\sigma}$

2) Adult income.csv
a) which column in the dataset had missing values?
→ Missing values are represented by "?", replace 'NAN'
For Numerical column - Replace with mean
For categorical column - Replace with mode
b) which categorical column did you identify?
How did you encode them?
→ workclass, education, marital status, relationship,
gender etc
encoding method: label encoding

4-3-25

Bafna Gold

Code:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder
from google.colab import files

# Upload Files Manually in Google Colab
uploaded = files.upload()

# Load the datasets (replace filenames accordingly after uploading)
diabetes_df = pd.read_csv("diabetes.csv")
adult_df = pd.read_csv("adult.csv")

# --- Data Cleaning ---
# Handling Missing Values: Fill numerical columns with median, categorical with mode
for df in [diabetes_df, adult_df]:
    for col in df.columns:
        if df[col].isnull().sum() > 0:
            if df[col].dtype == "object":
                df[col].fillna(df[col].mode()[0], inplace=True)
            else:
                df[col].fillna(df[col].median(), inplace=True)


# Handling Outliers: Capping values beyond 1.5*IQR
for df in [diabetes_df, adult_df]:
    for col in df.select_dtypes(include=np.number).columns:
        Q1, Q3 = df[col].quantile(0.25), df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower, upper = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR
        df[col] = np.clip(df[col], lower, upper)

# --- Handling Categorical Data ---
for df in [diabetes_df, adult_df]:
    categorical_cols = df.select_dtypes(include="object").columns
    for col in categorical_cols:
        df[col] = LabelEncoder().fit_transform(df[col])

# --- Data Transformations ---
scaler_minmax = MinMaxScaler()
scaler_standard = StandardScaler()

for df in [diabetes_df, adult_df]:
    numerical_cols = df.select_dtypes(include=np.number).columns
    df[numerical_cols] = scaler_minmax.fit_transform(df[numerical_cols])
    df[numerical_cols] = scaler_standard.fit_transform(df[numerical_cols])

# Save processed datasets
```

```
diabetes_df.to_csv("processed_diabetes.csv", index=False)
adult_df.to_csv("processed_adult.csv", index=False)

# Download processed files
files.download("processed_diabetes.csv")
files.download("processed_adult.csv")
import pandas as pd
from google.colab import files

# Upload Files Manually in Google Colab
uploaded = files.upload()

# Load the datasets
diabetes_df = pd.read_csv("diabetes.csv")
adult_df = pd.read_csv("adult.csv")

# Check for missing values
missing_diabetes = diabetes_df.isnull().sum()
missing_adult = adult_df.isnull().sum()

# Display columns with missing values
print("Missing values in Diabetes Dataset:")
print(missing_diabetes[missing_diabetes > 0])

print("\nMissing values in Adult Income Dataset:")
print(missing_adult[missing_adult > 0])

print("Missing Values Count in Diabetes Dataset:")
print(missing_diabetes)

print("\nMissing Values Count in Adult Income Dataset:")
print(missing_adult)

categorical_diabetes = diabetes_df.select_dtypes(include="object").columns.tolist()
categorical_adult = adult_df.select_dtypes(include="object").columns.tolist()

# Display categorical columns
print("Categorical Columns in Diabetes Dataset:", categorical_diabetes)
print("\nCategorical Columns in Adult Income Dataset:", categorical_adult)
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset.

Observation:

Date: Page:

Date - 11/03/2025

lab-4 Linear - Regression

X_i (week)	Y_i (sales in thousands)
1	2
2	4
3	5
4	9

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

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \qquad Y = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}$$

$$X^T X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$$

$$\text{Now } (X^T X)^{-1} = \begin{bmatrix} 30 & -10 \\ -10 & 4 \end{bmatrix} \frac{1}{20}$$

$$(X^T X)^{-1} X^T = \begin{bmatrix} 30 & -10 \\ -10 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \frac{1}{20}$$

$$= \begin{bmatrix} 20 & 10 & 0 & -10 \\ -6 & -2 & 2 & 6 \end{bmatrix} \frac{1}{20}$$

$$((X^T X)^{-1} X^T) Y = \begin{bmatrix} 20 & 10 & 0 & -10 \\ -6 & -2 & 2 & 6 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix} = \begin{bmatrix} -10 \\ 44 \end{bmatrix} \frac{1}{20}$$

Date: Page: 4

$$\therefore \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$$

$\therefore \beta_0 = -0.5$ and $\beta_1 = 2.2$

$$Y = \beta_0 + \beta_1 X$$
$$= -0.5 + (2.2)5 \qquad X = 5$$
$$= 10.5$$

$\therefore Y$ for $x = 5$ $= 10.5$

Nа
11-3-25

Code:

```
import numpy as np

# Given data
# x: Week numbers
# y: Sales in thousands
x = np.array([1, 2, 3, 4])
y = np.array([2, 4, 5, 9])

# Construct the design matrix X by adding a column of ones (for the intercept)
X = np.column_stack((np.ones(x.shape[0]), x))

# Compute the coefficients using the formula: beta = (X^T X)^(-1) X^T y
XtX = X.T.dot(X)           # Compute X^T X
XtX_inv = np.linalg.inv(XtX)  # Invert X^T X
XtY = X.T.dot(y)           # Compute X^T y

beta = XtX_inv.dot(XtY)     # Compute beta

# Display the computed coefficients
print("Computed coefficients (beta):", beta)

import matplotlib.pyplot as plt

# ... (previous code)

# Generate points for the regression line
x_line = np.linspace(x.min(), x.max(), 100)  # Create 100 points for a smooth line
y_line = beta[0] + beta[1] * x_line         # Calculate y-values for the line

# Plot the data points
plt.scatter(x, y, label='Data Points', color='blue')

# Plot the regression line
plt.plot(x_line, y_line, label='Linear Regression', color='red')

# Customize the plot
plt.xlabel('Week Number (x)')
plt.ylabel('Sales (thousands) (y)')
plt.title('Linear Regression Plot')
plt.legend()  # Show the legend
plt.grid(True)  # Show the grid

# Display the plot
plt.show()
```

```
import numpy as np

# Given data
x = np.array([8, 10, 12])
y = np.array([10, 13, 16])

# Construct the design matrix X (adding a column of ones for the intercept)
X = np.column_stack((np.ones(x.shape[0]), x))

# Compute beta using the normal equation: beta = (X^T X)^(-1) X^T y
XtX = X.T.dot(X)
XtX_inv = np.linalg.inv(XtX)
XtY = X.T.dot(y)
beta = XtX_inv.dot(XtY)

# Extract coefficients
beta0, beta1 = beta
print("Intercept (beta0):", beta0)
print("Slope (beta1):", beta1)

# Predict the price for a 20-inch pizza
x_new = 20
y_pred = beta0 + beta1 * x_new
print("Predicted price for a 20-inch pizza: $", y_pred)

import pandas as pd
from sklearn.linear_model import LinearRegression
# Load the data
income_data = pd.read_csv("canada_per_capita_income.csv")
# Assumed data columns: 'Year' and 'PerCapitaIncome'
print("Canada Income Data Head:")
print(income_data.head())
# Prepare feature and target
X_income = income_data[["year"]]     # Predictor variable: Year
y_income = income_data["per capita income (US$)"]  # Target variable: Per capita income

# Build and train the linear regression model
model_income = LinearRegression()
model_income.fit(X_income, y_income)
# Predict per capita income for the year 2020
predicted_income = model_income.predict([[2020]])
print("\nPredicted per capita income for Canada in 2020:", predicted_income[0])

import matplotlib.pyplot as plt

# ... (previous code)

# Predict per capita income for the year 2020
```

```
predicted_income = model_income.predict([[2020]])
print("\nPredicted per capita income for Canada in 2020:", predicted_income[0])

# Plot the data points and the regression line
plt.scatter(X_income, y_income, color='blue', label='Actual Data')
plt.plot(X_income, model_income.predict(X_income), color='red', label='Regression Line')

# Plot the prediction for 2020
plt.scatter(2020, predicted_income[0], color='green', label='Prediction for 2020')

# Customize the plot
plt.xlabel('Year')
plt.ylabel('Per Capita Income (US$)')
plt.title('Canada Per Capita Income Prediction')
plt.legend()
plt.grid(True)

# Display the plot
plt.show()

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

# Load the salary data
salary_data = pd.read_csv("salary.csv")
print(income_data.head())
# Check for null values and handle them (e.g., imputation or removal)
if salary_data.isnull().values.any():
    print("Null values found in the salary dataset. Handling null values...")
    # Example: Fill null values with the mean of the 'YearsExperience' column
    salary_data['YearsExperience'].fillna(salary_data['YearsExperience'].mean(), inplace=True)
    # Other options: Remove rows with nulls or use more sophisticated imputation methods

# Prepare feature and target
X_salary = salary_data[["YearsExperience"]]  # Predictor variable: Years of Experience
y_salary = salary_data["Salary"]           # Target variable: Salary
# Build and train the linear regression model
model_salary = LinearRegression()
model_salary.fit(X_salary, y_salary)
# Predict salary for an employee with 12 years of experience
predicted_salary = model_salary.predict([[12]])
print("\nPredicted salary for an employee with 12 years of experience:", predicted_salary[0])

import matplotlib.pyplot as plt
# Plot the data points and the regression line
plt.scatter(X_salary, y_salary, color='blue', label='Actual Data')
```

```
plt.plot(X_salary, model_salary.predict(X_salary), color='red', label='Regression Line')

# Plot the prediction for 12 years of experience
plt.scatter(12, predicted_salary[0], color='green', label='Prediction for 12 years')

# Customize the plot
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Salary Prediction based on Experience')
plt.legend()
plt.grid(True)

# Display the plot
plt.show()

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression

# Read the CSV file (ensure the file is uploaded in your Colab environment)
df = pd.read_csv("hiring.csv")
# Rename columns for convenience
df.columns = ['experience', 'test_score', 'interview_score', 'salary']

print("Original Data:")
print(df)
# Define a mapping for text to numeric conversion for the 'experience' column
num_map = {
    "zero": 0,
    "one": 1,
    "two": 2,
    "three": 3,
    "four": 4,
    "five": 5,
    "six": 6,
    "seven": 7,
    "eight": 8,
    "nine": 9,
    "ten": 10,
    "eleven": 11,
    "twelve": 12
}

# Function to convert experience values to numeric
def convert_experience(x):
    try:
        return float(x)
    except:
```

```
    x_lower = str(x).strip().lower()
    return num_map.get(x_lower, np.nan)

# Convert the 'experience' column using the mapping
df['experience'] = df['experience'].apply(convert_experience)

# Convert 'test_score', 'interview_score', and 'salary' to numeric (coerce errors to NaN)
df['test_score'] = pd.to_numeric(df['test_score'], errors='coerce')
df['interview_score'] = pd.to_numeric(df['interview_score'], errors='coerce')
df['salary'] = pd.to_numeric(df['salary'], errors='coerce')

print("\nData After Conversion:")
print(df)
# Fill missing values in numeric columns using the column mean
df['experience'].fillna(df['experience'].mean(), inplace=True)
df['test_score'].fillna(df['test_score'].mean(), inplace=True)
df['interview_score'].fillna(df['interview_score'].mean(), inplace=True)

print("\nData After Filling Missing Values:")
print(df)
# Prepare the feature matrix X and target vector y
X = df[['experience', 'test_score', 'interview_score']]
y = df['salary']

# Build and train the Multiple Linear Regression model
model = LinearRegression()
model.fit(X, y)
# Predict salaries for the given candidate profiles
# Candidate 1: 2 years of experience, 9 test score, 6 interview score
candidate1 = np.array([[2, 9, 6]])
predicted_salary1 = model.predict(candidate1)

# Candidate 2: 12 years of experience, 10 test score, 10 interview score
candidate2 = np.array([[12, 10, 10]])
predicted_salary2 = model.predict(candidate2)
import matplotlib.pyplot as plt

# Create the plot
plt.figure(figsize=(10, 6))  # Adjust figure size for better visualization
plt.scatter(df['experience'], y, color='blue', label='Actual Salary') #Plot actual salary against years of
experience

# Plot the regression line (this is an approximation since it's a multi-variable regression)
# You can visualize a single feature against the predicted salary
plt.plot(df['experience'], model.predict(X), color='red', label='Regression Line')

# Highlight predictions
plt.scatter(candidate1[0, 0], predicted_salary1, color='green', label='Candidate 1 Prediction')
```

```
plt.scatter(candidate2[0, 0], predicted_salary2, color='purple', label='Candidate 2 Prediction')

# Add labels and title
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.title("Salary Prediction based on Experience, Test Score, Interview Score")

# Add a legend
plt.legend()
plt.grid(True)
plt.show()

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression

# Read the CSV file (ensure the file is uploaded in your Colab environment)
df = pd.read_csv("1000_Companies.csv")
# Display the first few rows
print("Original Data:")
print(df.head())
# --- Data Preprocessing ---

# For numeric columns, fill missing values with the column mean
numeric_cols = ["R&D Spend", "Administration", "Marketing Spend", "Profit"]
for col in numeric_cols:
    df[col].fillna(df[col].mean(), inplace=True)

# For the categorical column 'State', fill missing values with a placeholder
df["State"].fillna("Unknown", inplace=True)

# Confirm that missing values are handled
print("\nMissing Values After Processing:")
print(df.isnull().sum())
# Separate the features and target variable
features = ["R&D Spend", "Administration", "Marketing Spend"] + \
           [col for col in df_encoded.columns if col.startswith("State_")]
X = df_encoded[features]
y = df_encoded["Profit"]
# --- Prediction for a New Company ---

# Given sample data:
# R&D Spend = 91694.48, Administration = 515841.3, Marketing Spend = 11931.24, State = 'Florida'
new_company = pd.DataFrame({
    "R&D Spend": [91694.48],
    "Administration": [515841.3],
    "Marketing Spend": [11931.24],
    "State": ["Florida"]
```

```
})
# One-hot encode the 'State' column using the same strategy as training data
new_company_encoded = pd.get_dummies(new_company, columns=["State"], drop_first=True)

# Align the new data's columns with the training features (fill missing columns with 0)
new_company_encoded = new_company_encoded.reindex(columns=X.columns, fill_value=0)

# Predict the profit using the trained model
predicted_profit = model.predict(new_company_encoded)
print("\nPredicted Profit for the New Company: $", round(predicted_profit[0], 2))


import matplotlib.pyplot as plt

# Assuming 'df_encoded', 'features', 'X', 'y', 'model', 'new_company_encoded', and 'predicted_profit' are
defined from the previous code

# Create the plot
plt.figure(figsize=(10, 6))

# Scatter plot of actual profits vs. R&D Spend
plt.scatter(df_encoded["R&D Spend"], y, color='blue', label='Actual Profit')

# Plot the regression line (approximation for visualization)
plt.plot(df_encoded["R&D Spend"], model.predict(X), color='red', label='Regression Line')

# Highlight the new company's prediction
plt.scatter(new_company_encoded["R&D Spend"], predicted_profit, color='green', label='New
Company Prediction')

# Add labels and title
plt.xlabel("R&D Spend")
plt.ylabel("Profit")
plt.title("Profit Prediction based on R&D Spend")

# Add a legend
plt.legend()
plt.grid(True)
plt.show()
```

Program 4

Build Logistic Regression Model for a given dataset.

Observation:

Date- 18/3/2025

LAB-3

- Binary classification problem

Given $a_0 = -5$

$a_1 = 0.8$

$x = 7$

$z = a_0 + a_1 x$

$= (-5) + 0.8 \times 7 = 0.6$

$$y = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-0.6}} = 0.64$$

a) given threshold = 0.5

$0.64 > 0.5$

∴ Student who studies for 7 hours will pass

b) Softmax function

$z = [2, 1, 0]$ for 3 classes

$$z_k = \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}$$

$$z_1 = \frac{e^2}{e^2+e^1+e^0} = \frac{7.39}{7.39+2.72+1} \approx 0.665$$

$$z_2 = \frac{e^1}{e^2+e^1+e^0} = \frac{7.39 = 2.72}{7.39+2.72+1} \approx 0.244$$

$$z_3 = \frac{e^0}{e^2+e^1+e^0} = \frac{1}{7.39+2.72} \approx 0.091$$

Probabilities = 66.5%, 24.4%, 9.1%

Bafna Gold

Code:

Hr.csv

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load dataset
df = pd.read_csv('HR_comma_sep.csv')

# Basic Info
print("Dataset Info:")
print(df.info())
print("\nFirst few rows:")
print(df.head())
plt.figure(figsize=(8, 6))
# sns.countplot(x='salary', hue='left', data=df)
sns.barplot(x='Department', y='satisfaction_level', data=df)

# plt.title('Salary vs Employee Retention')
plt.xlabel('Departments')
plt.ylabel('Satisfaction level')

plt.show()

import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Encode categorical variables (drop_first avoids dummy variable trap)
df_encoded = pd.get_dummies(df, columns=['salary', 'Department'], drop_first=True)

plt.figure(figsize=(15, 8))
sns.heatmap(df_encoded.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

plt.figure(figsize=(8, 5))
sns.countplot(x='salary', hue='left', data=df, order=['low', 'medium', 'high'])

plt.title('Impact of Salary on Employee Retention')
plt.xlabel('Salary Level')
plt.ylabel('Number of Employees')
plt.legend(title='Left', labels=['Stayed', 'Left'])
```

```
plt.show()
df_encoded = pd.get_dummies(df, columns=['Department', 'salary'], drop_first=True)

# Calculate the correlation matrix
correlation_matrix = df_encoded.corr()

# Extract the correlation with 'left' (employee retention)
correlation_with_left = correlation_matrix['left'].sort_values(ascending=False)

# Display the correlation
print(correlation_with_left)
plt.figure(figsize=(12, 6))
sns.countplot(x='Department', hue='left', data=df)

# Title and labels
plt.title('Impact of Department on Employee Retention')
plt.xlabel('Department')
plt.ylabel('Number of Employees')
plt.legend(title='Left', labels=['Stayed', 'Left'])
plt.xticks(rotation=45)  # Rotate department names for readability
plt.show()
# Step 1: Preprocess the data
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
df = pd.read_csv('HR_comma_sep.csv')

# Select important features and encode categorical variable
df_encoded = pd.get_dummies(df, columns=['salary'], drop_first=True)  # This encodes salary (low ->
low salary column)

# Step 2: Define features (X) and target (y)
X = df_encoded[['satisfaction_level', 'time_spend_company', 'salary_low']]  # Using low salary as a
feature
y = df_encoded['left']  # Target variable (whether the employee left or stayed)

# Step 3: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Build and train the logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Step 5: Make predictions
y_pred = model.predict(X_test)
```

```
# Step 6: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Program 5

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

Observation:

Lab-15 Date:-15|4|2025

Building Decision Tree :-

Q?

Instance	a_2	a_3	classification
1	Hot	High	No
2	Hot	High	No
3	Cool	High	No
4	Hot	High	No
8	Hot	Normal	Yes

$$\text{Entropy} = -\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5}$$

$$= 0.7219$$

- for a_2 :

$$S_{Hot}\,[1+, 3-] = -\frac{1}{4}\log\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4}$$

$$= 0.8113$$

$$S_{cool}\,(0+, 1-) = 0$$

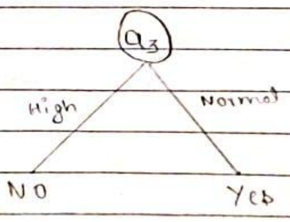
$$S_{Gain} = 0.7219 - \frac{4}{5} \times 0.813 - 0 = 0.0715$$

- for a_3 :

$$S_{High}\,[0+, 4-] = 0$$

$$S_{Normal}\,[1+, 0-] = 0$$

$$S_{Gain}\,[S, a_3] = 0.7219$$



Bafna Gold

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
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
data = {
    'a1': [True, True, False, False, False, True, True, True, False, False],
    'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],
    'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal', 'High'],
    'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes']
}
df = pd.DataFrame(data)
df.head()
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
df.head()
X = df.drop('Classification', axis=1)
y = df['Classification']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Program 6

Build KNN Classification model for a given dataset.

Observation:

Page: 9

lab-6 Date - 1/4/2025

Q.7 KNN

Person	Age	Salary	Target	Distance	Rank
A	18	50	N	52.8	
B	23	55	N	46.57	
C	24	70	N	31.95	2
D	41	60	Y	40.04	3
E	43	70	Y	31.04	1
F	38	40	Y	60.07	

Rank	Target
E	Y
C	N
D	Y

	Target
∴ X	Y

1/4/2

Bafna Gold

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
data = pd.read_csv("diabetes.csv")
data.head()
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
ss = StandardScaler()
X[["Pregnancies"]] = ss.fit_transform(X[["Pregnancies"]])
X[["Glucose"]] = ss.fit_transform(X[["Glucose"]])
X[["BloodPressure"]] = ss.fit_transform(X[["BloodPressure"]])
X[["SkinThickness"]] = ss.fit_transform(X[["SkinThickness"]])
X[["Insulin"]] = ss.fit_transform(X[["Insulin"]])
X[["BMI"]] = ss.fit_transform(X[["BMI"]])
X[["DiabetesPedigreeFunction"]] = ss.fit_transform(X[["DiabetesPedigreeFunction"]])
X[["Age"]] = ss.fit_transform(X[["Age"]])
X.head()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
knn = KNeighborsClassifier()
param_grid = {"n_neighbors": [1, 3, 5, 7, 9]}
grid = GridSearchCV(estimator = knn, param_grid = param_grid, cv = 5, scoring = "accuracy")
grid.fit(X_train, y_train)
grid.best_params_
best = grid.best_estimator_
best
y_pred = best.predict(X_test)
accuracy_score(y_test, y_pred)
```

Program 7

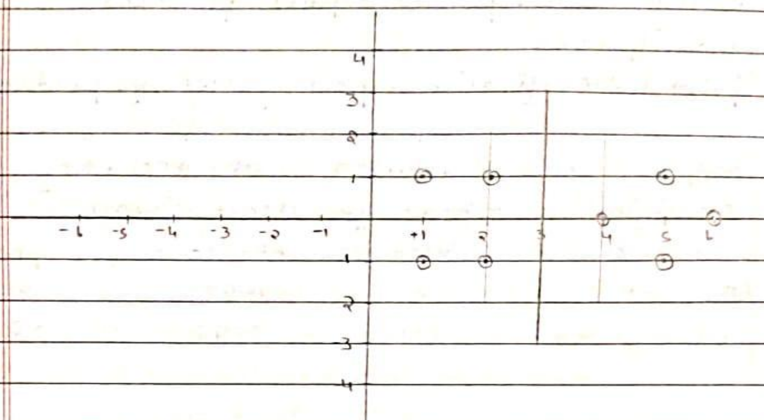
Build Support vector machine model for a given dataset.

Observation:

lab-7 Date - 8/4/2025

SVM

- Draw an optimal hyperplane using linear SVM to classify the following points
  $\{(1,1), (2,1), (1,-1), (2,-1)\}$ - +ve labelled
  $\{(4,0), (5,1), (5,-1), (6,0)\}$ - -ve labelled



$$S_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad S_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix} \quad S_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$\tilde{S_1} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} \quad \tilde{S_2} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} \quad \tilde{S_3} = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$$

$$\alpha_1 \tilde{S_1}\tilde{S_1} + \alpha_2 \tilde{S_2}\tilde{S_1} + \alpha_3 \tilde{S_3}\tilde{S_1} = +1$$
$$\alpha_1 \tilde{S_1}\tilde{S_2} + \alpha_2 \tilde{S_2}\tilde{S_2} + \alpha_3 \tilde{S_3}\tilde{S_2} = +1$$
$$\alpha_1 \tilde{S_1}\tilde{S_3} + \alpha_2 \tilde{S_2}\tilde{S_3} + \alpha_3 \tilde{S_3}\tilde{S_3} = -1$$

$$\alpha_1(6) + \alpha_2(4) + \alpha_3(9) = +1 \qquad \alpha_1 = 13/4$$
$$\alpha_1(4) + \alpha_2(6) + \alpha_3(9) = +1 \qquad \alpha_2 = 13/4$$
$$\alpha_1(9) + \alpha_2(9) + \alpha_3(17) = -1 \qquad \alpha_3 = -7/2$$

$$w = \alpha_1 \tilde{S_1} + \alpha_2 \tilde{S_2} + \alpha_3 \tilde{S_3}$$

Bafna Gold

Page 12

$$= \frac{13}{4}\begin{bmatrix}2\\1\\1\end{bmatrix} + \frac{13}{4}\begin{bmatrix}2\\-1\\1\end{bmatrix} + \frac{-7}{2}\begin{bmatrix}4\\0\\1\end{bmatrix} = \begin{bmatrix}-1\\0\\3\end{bmatrix} = -\begin{bmatrix}1\\0\\-3\end{bmatrix}$$

$z = \begin{bmatrix}+1\\0\end{bmatrix}$ $\quad b = -3$

$b + 3 = 0$

intercept on x axis = 3

$z \begin{bmatrix}1\\0\end{bmatrix} \rightarrow$ line parallel to y axis

8-4-2?

Code:

Iris.csv

```
import pandas as pd
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import OrdinalEncoder
data = pd.read_csv("iris (1).csv")
data.head()
oe = OrdinalEncoder()
data[["species"]] = oe.fit_transform(data[["species"]])
data.head()
y = data.iloc[:, -1]
X = data.iloc[:, :-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
rbf_model = SVC(kernel='rbf')
rbf_model.fit(X_train, y_train)
rbf_model.score(X_test,y_test)
y_pred = rbf_model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
linear_model = SVC(kernel='linear')
linear_model.fit(X_train,y_train)
linear_model.score(X_test,y_test)
y_pred = rbf_model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
```

Digits.csv

```
import pandas as pd
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
digits = load_digits()
digits.target
dir(digits)
X_train, X_test, y_train, y_test = train_test_split(df.drop('target',axis='columns'), df.target,
test_size=0.3)
rbf_model = SVC(kernel='rbf')
rbf_model.fit(X_train, y_train)
linear_model = SVC(kernel='linear')
linear_model.fit(X_train,y_train)
```

Program 8

Implement Random forest ensemble method on a given dataset.

Observation:

Date: Page: 14

lab-8 Date - 15/04/2025

Questions

1. Difference between decision tree and Random forest classifier

⇒ Decision tree | Random forest

Decision tree	Random forest
• A decision tree is a single tree structure where decisions are made by splitting the dataset at each node	• It is an ensemble learning method that builds multiple decision trees and combines their results
• These reduce overfitting by averaging multiple especially with complex datasets	• These reduce overfitting by averaging multiple decision trees, which generally improves performance
• A model has high variance and low bias	• A models tend to have low variance and moderate bias as they combine the predictions from many trees

2. Discuss all the parameters of random forest classifier

1. n-estimators :- The no of trees in the forest
2. iteration :- The numbers of times to occurs
3. max - depth :- The max depth of the tree
4. min- sample split :- The main no of sample required to split an internal node
5. min - samples leaf :- The min no of sample required to be featured in leaf node
6. max - features :- The no of features to consider when looking for the best split

Bafna Gold

7 bootstrap: whether to use bootstrapping when creating trees
8 oob_score: whether to use out of bag samples to estimates the generalization accuracy
9. n_jobs :- The no of jobs to run in parallel for both fit() and predict()
10 random_state :- controls the randomness of the estimator

3. Algorithm
-> Step 1: The data set is divided into input features (x) and the target labels (y)
-> Step 2: For each tree, create a bootstrap sample from the dataset
-> Step 3: for each tree, randomly select a subset of features. It is determined by max - features parameters
-> Step 4: Build a decision tree on the bootstrapped dataset using the selected subset of features
-> Step 5:- Once all trees are built, make predictions by aggregating the predictions of all the trees
-> Step 6:- During training, the data points that were not models generalization performance
-> Step 7:- The final model is evaluated using metric line accuracy, confusion matrix and AUC score depending on the task of hand

Bafna Gold

Code:

```
import numpy as np
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
from sklearn.preprocessing import OrdinalEncoder
data = pd.read_csv("iris (2).csv")
data.head()
oe = OrdinalEncoder()
data[["species"]] = oe.fit_transform(data[["species"]])
data.head()
y = data.iloc[:, -1]
X = data.iloc[:, :-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
rf = RandomForestClassifier(n_estimators=10, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracy
n_estimators_list = [10, 50, 100, 200, 500, 1000]
accuracies = []

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)
    accuracies.append(accuracy)
    print(f"Accuracy with n_estimators={n}: {accuracy:.4f}")
plt.plot(n_estimators_list, accuracies, marker='o')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.title('Random Forest Accuracy vs Number of Trees')
plt.show()
optimal_n_estimators = n_estimators_list[np.argmax(accuracies)]
print(f"Best accuracy is obtained with n_estimators={optimal_n_estimators}")
```

Program 9

Implement Boosting ensemble method on a given dataset.

Observation:

Date: Page:

lab-9 "Adaboost"

=> Boosting is an ensemble learning technique that combines multiple weak decision trees to create a strong learner. It works by sequentially training models, where each new model focuses on mistakes made by previous ones. The model prediction are weighted based on their performance and the final prediction is made by combining the weighted predictions of these models.

=> Parameters of AdaBoost classifiers

* best estimator (object, default = None) - based model to be used for boosting, uses decision trees by default
* n-estimators (int) - the number of boosting rounds, or weak learners to train
* learning rate (float) - scaling factor for the contribution of each estimator lower values make model robust
* Algorithm IS 'SAMME', 'SAMME-R'}, default ='SAMME-R') boosting algorith to use SAMME.R uses probabilistic approaches
* random - state (int) - control randomness of the estimator
* loss {{'linear', 'square', 'exponential'}} - loss function to be used for boosting

=> Algorithm

1. Initialize: set the weights of all training samples equally

Bafna Gold

Page: 16

2. For each initialization (n-estimators)
- Train a base learner on weighted training set data
- calculate error of the model on training data
- compute model's weight based on error
- update sample weight - increase weight of misclassified samples

3. combine:
The final model is the weighted sum of the base learners, where learners with lower errors have higher weights

Code:

```
import numpy as np
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
from sklearn.preprocessing import OrdinalEncoder
data = pd.read_csv("income.csv")
data.head()
y = data.iloc[:, -1]
X = data.iloc[:, :-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
rf = AdaBoostClassifier(n_estimators=1000, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracy
n_estimators_list = [10, 50, 100, 200, 500, 1000]
accuracies = []

for n in n_estimators_list:
    rf = AdaBoostClassifier(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)
    accuracies.append(accuracy)
    print(f"Accuracy with n_estimators={n}: {accuracy:.4f}")


plt.plot(n_estimators_list, accuracies, marker='o')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.title('Random Forest Accuracy vs Number of Trees')
plt.show()
optimal_n_estimators = n_estimators_list[np.argmax(accuracies)]
print(f"Best accuracy is obtained with n_estimators={optimal_n_estimators}")
```

Program 10

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

Observation:

Page 17

lab-10 K-means

⇒ Algorithm

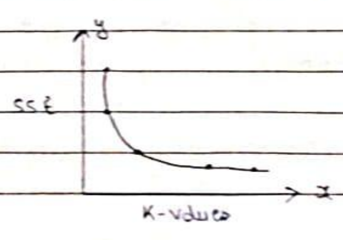
1. Select the number K to decide the number of cluster
2. Select random K points or centroids
3. Assign each data point to their closest centroid which will form the predefined K cluster
4. Calculate the variance & place a new centroid of each cluster
5. Repeat the 3rd step which means reassign each datapoint to the new closest centroid of each cluster
6. If any reassignment occurs then go to step 4 - finish

⇒ K determines the number of clusters
Elbow method
Silhouette score
cross validation
gap statistics

⇒ SSE is defined as the sum of squared Euclidean distance of each point to its closest centroid

$SSE_j = \Sigma_{i=0} dist(x_i - c_j)^2$

$SSE = SSE_1 + SSE_2 + \ldots SSE_k$



Bafna Gold

Date Page

→ Elbow method is a technique used to determine the optimal numbers of clusters (K) in K-mean clustering. It involves plotting within cluster scene of squares (WCSS) again the number of clusters and identifying the elbow point where the decrease in WCSS starts to level off. The elbow point suggests the optimal number of clusters.

→ • n-clusters : int, default = 8 : number of cluster to form the well as the number of centroid to generate

- init : {K-mean, random} : chooses a method for initialization with different centroid speed
- max-iter : max number of iterations of K-mean algo for a single-run
- tol, float : Relative tolerance with regards to Frobenive norms of the different in the cluster centre of two consecutive iterations to declare convergence
- verbose ; verbosity mode
- random state
- copy - X
- Algorithm : {"lloyd", "elkan"}

==Code:

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

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

# Use only petal_length and petal_width
X = df[["petal_length", "petal_width"]]

# Scale the features (helps with KMeans)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Elbow method to determine 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 the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.title("Elbow Method for Optimal k")
plt.xlabel("Number of clusters (k)")
plt.ylabel("Inertia (Within-Cluster Sum of Squares)")
plt.grid(True)
plt.show()

# Find optimal k using "elbow" (visually)
optimal_k = 3  # for IRIS, elbow is usually at 3
print(f"Optimal number of clusters (k): {optimal_k}")
```

Program 11

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

Observation:

Lab-11

PCA

1. Calculate the mean
2. Calculation of covariance matrix
3. Eigen values of two covariance matrix
4. Conversion of the eigen vector - Unit eigen vector
5. Computation of first principal component
6. Geometrical meaning of first principal component

⇒ Reduce dimensions 2 to 1 using PCA

Feature	Example 1	2	3	4
X_1	4	8	13	7
X_2	11	4	5	14

Step 1: Calculate Mean

$\bar{X}_1 = 8$

$\bar{X}_2 = 8.5$

Step 2: Calculation of covariance matrix

$$Cov(X_1, X_1) = \frac{1}{N-1} \sum_{K=1}^{N} (X_{1K} - \bar{X}_1)^2$$

$$= \frac{1}{3} (4-8)^2 + (8-8)^2 + (13-8)^2 + (7-8)^2 = 14$$

$$cov(X_1, X_2) = \frac{1}{N-1} \sum_{K=1}^{N} (X_{1K} - \bar{X}_1)(X_{2K} - \bar{X}_2)$$

$$= -11$$

$$Cov(X_2, X_1) = Cov(X_1, X_2) = -11$$

$$Cov(X_2, X_2) = \frac{1}{N-1} \sum_{K=1}^{N} (X_{2K} - \bar{X}_2)^2$$

$$= 23$$

$$S = \begin{bmatrix} cov(X_1, X_1) & cov(X_1, X_2) \\ Cov(X_2, X_1) & Cov(X_2, X_2) \end{bmatrix}$$

Bafna Gold

Date: Page: 19

Step 3: Eigen values of the covariance matrix

The characteristics equation of the covariance matrix is $0 = \det(S - \lambda I)$

$$= \begin{vmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{vmatrix}$$

$$= (14-\lambda)(23-\lambda) - (-11) * (-11)$$

$$= \lambda^2 - 37\lambda + 201$$

Solving characteristic equation we get

$$\lambda = \frac{1}{2}(37 \pm \sqrt{565})$$

$$= 30.3849,\ 6.6151$$

$$= \lambda_1, \lambda_2 \text{ (say)}$$

Step 4: computation of the eigen vector

larget eigen value $= \lambda_1$

$\lambda = \lambda_1$

$$V = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = (S - \lambda_1 I) V$$

$$= \begin{bmatrix} 14-\lambda_1 & -11 \\ -11 & 23-\lambda_1 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\Rightarrow (14-\lambda_1)u_1 - 11u_2 = 0$$

$$-11u_1 + (23-\lambda_2)u_2 = 0$$

$$\frac{u_2}{11} = \frac{u_2}{14-\lambda_1} = t$$

$u_1 = 11t,\ u_2 = (14-\lambda_1)t$ where t is any real number

Taking

$t = 1$, we get an eigen vector, corresponding to λ_1 as $u_1 = \begin{bmatrix} 11 \\ 14-\lambda_1 \end{bmatrix}$

Bafna Gold

Date Page

$$\|V_1\| = \sqrt{11^2 + (14-\lambda_1)^2}$$
$$= \sqrt{11^2 + (14-30.3844)^2}$$
$$= 19.7348$$

i.e. a unit eigen vector corresponding to λ_1 is

$$e_1 = \begin{bmatrix} 11 / \|V_1\| \\ (14-\lambda_1) / \|V_1\| \end{bmatrix}$$

$$= \begin{bmatrix} 11 / 19.7348 \\ (14-38.44) / 19.7348 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$$

eigen vector e_2 corresponding to eigen value λ_2

$$e_2 = \begin{bmatrix} 0.8305 \\ 0.5579 \end{bmatrix}$$

Step 5: computation of 1st principal component

$$= \begin{bmatrix} X_{1k} \\ X_{2k} \end{bmatrix}$$

$$e_1^T \begin{bmatrix} X_{1k} - \bar{X}_1 \\ X_{2k} - \bar{X}_2 \end{bmatrix} \cdot [0.5579 \; -0.8305] \begin{bmatrix} X_{1k} - \bar{X}_1 \\ X_{2k} - \bar{X}_2 \end{bmatrix}$$

$$= 0.557 + (X_{1k} - \bar{X}_1) - 0.8303 (X_{2k} - \bar{X}_2)$$

Code on next page.

Code:

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

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

# Separate features and target
X = df.drop("HeartDisease", axis=1)
y = df["HeartDisease"]

# Identify categorical columns
cat_cols = X.select_dtypes(include=['object']).columns.tolist()

# Label Encode binary categorical columns
label_enc = LabelEncoder()
for col in cat_cols:
    if X[col].nunique() == 2:
        X[col] = label_enc.fit_transform(X[col])
        cat_cols.remove(col)

# One-hot encode remaining categorical columns
X = pd.get_dummies(X, columns=cat_cols)

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

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

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

```
# Store accuracy scores
accuracy_before_pca = {}
accuracy_after_pca = {}

# Training and evaluating models before PCA
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)
    accuracy_before_pca[name] = acc

# Apply PCA
pca = PCA(n_components=0.95)  # retain 95% variance
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Training and evaluating models after 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)
    accuracy_after_pca[name] = acc

# Print accuracy comparison
print("Model Accuracy Comparison (Before vs After PCA):")
print(f"{'Model':<20} {'Before PCA':<15} {'After PCA':<15}")
for name in models.keys():
    print(f"{name:<20} {accuracy_before_pca[name]:<15.4f} {accuracy_after_pca[name]:<15.4f}")
```