VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Aditya Kumar (1BM22CS018)

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

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by **Aditya Kumar (1BM22CS018),** 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/aditya-9854/ML_LAB

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
```

```python
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())
```

Program 2

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

Observation:

LAB-2
4/9/25

Write python code, consider CSV file "housing.csv"

i) To read .csv file into the data frame
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 a dataset have missing values count greater than zero.

```
→ import pandas as pd

df = pd.read_csv('housing.csv')

print("Information of all columns:")
df.info()

print("\n Statistical information of all
    numerical columns:")
print(df.describe())

print("\n Count of unique label for 'ocean
proximity' column:")
print(df['Ocean Proximity'].value_counts
```

Figure 2.1

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:

Lab-4

Linear Regression Problem using matrix approach

x_i (week)	y_i (Sales in Thous)
1	2
2	4
3	5
4	9

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

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

$$\therefore Y = \beta_0 + \beta_1 x + e$$

$$Y = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}, \quad X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

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

$$X^T X = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix}_{2\times4} \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}_{4\times2} = \begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix}_{2\times2}$$

Figure 3.1

$$(X^TX)^{-1} = \begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix}^{-1}$$

$$= \begin{pmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{pmatrix}$$

$$((X^TX)^{-1}X^T) = \begin{pmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{pmatrix}_{2\times 2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix}_{2\times 4}$$

$$\begin{pmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{pmatrix}_{2\times 4}$$

$$((X^TX)^{-1}X^T)Y = \begin{pmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{pmatrix}_{2\times 4} \begin{pmatrix} 2 \\ 4 \\ 5 \\ 9 \end{pmatrix}_{4\times 1}$$

$$\begin{pmatrix} -0.5 \\ 2.2 \end{pmatrix}_{2\times 1} \qquad \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \begin{matrix} \to \text{intercept} \\ \to \text{slope} \end{matrix}$$

$$\Rightarrow \beta_0 = -0.5, \; \beta_1 = 2.2$$

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:

LAB 3

18/3/25

Date ____
Page ____

1. Consider a binary classification problem where we want to predict whether a student will pass or fail based on their study hrs. The logistic regr model has trained, and the learned parameters are $a_0 = -5$ (intercept), and $a_1 = 0.8$ (coeff)

a) write the logistic regn eqn for it

b) calculate the prob that a student who studies for 7 hrs will pass

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

Sol.

$y = mx + b$ → linear regn

$$y = \frac{1}{1 + e^{-(mx+b)}} \rightarrow \text{logistic regn}$$

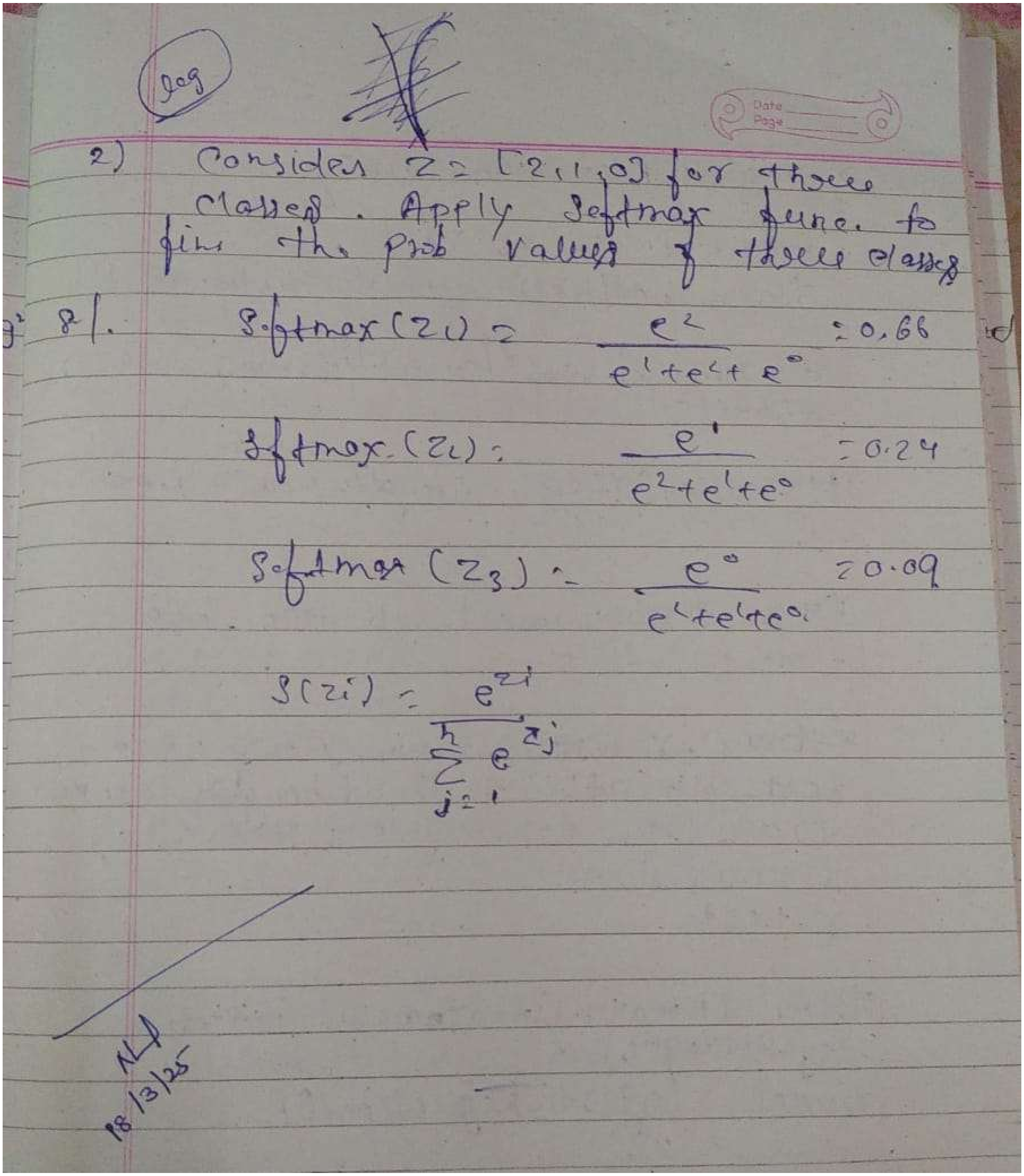
$$= \frac{1}{1 + e^{-(-5 + 0.8 \times 7)}} = 0.64$$

Since $0.64 > 0.5$

he will pass

$$p(x) = \frac{1}{1 + \exp(-(a_0 + a_1 x))}$$

Figure 4.1



leg

2) Consider $z = [2,1,0]$ for three classes. Apply softmax func. to find the prob values of three classes

Sol.

$$\text{softmax}(z_1) = \frac{e^2}{e^1+e^2+e^0} = 0.66$$

$$\text{softmax}(z_2) = \frac{e^1}{e^2+e^1+e^0} = 0.24$$

$$\text{softmax}(z_3) = \frac{e^0}{e^2+e^1+e^0} = 0.09$$

$$s(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

18/3/25

Accuracy of the multinomial logistic Regression model on the test set : 1.00

2. For dataset file "HR_comma_sep.csv"

i) Which var did you identify as having a direct and clear impact on employee retention? Why.

:) Satisfaction level, last evaluation, Time spent at company, No. of projects, Avg monthly Hrs, work accident, Promotion in last 5 yrs.

ii) Model Accuracy

: The accuracy of the logistic regr model depends on the dataset and model if the accuracy is high it is reasonable at recall F1-score.

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:

11/4/25

LAB 4

(Building Decision Tree)

Q)

Instance	a_2	a_3	Classification
1	Hot	High	No
2	Hot	High	No
3	Cool	High	No
7	Hot	High	No.
8	Hot	Normal	Yes

$$\text{Entropy}(y) = -\frac{4}{5}\log_2\left(\frac{4}{5}\right) - \frac{1}{5}\log_2\left(\frac{1}{5}\right)$$

$$= 0.7219$$

For a_2:

$$S_{Hot}[1+,3-] = -\frac{1}{4}\log\frac{1}{4} - \frac{3}{4}\log_2\left(\frac{3}{4}\right) = 0.8113$$

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

$$\cancel{S_{Gain}}\ (S, a_2)\ 0.7219 - \frac{4}{5} \times 0.813 = 0.0715$$

For a_3:

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

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

$$\text{Gain}(S, a_3) = 0.7219 - 0 - 0 = 0.7219$$

$\therefore$ a_3 is the root node since gain (S, a_3) is high.

Figure 5.1

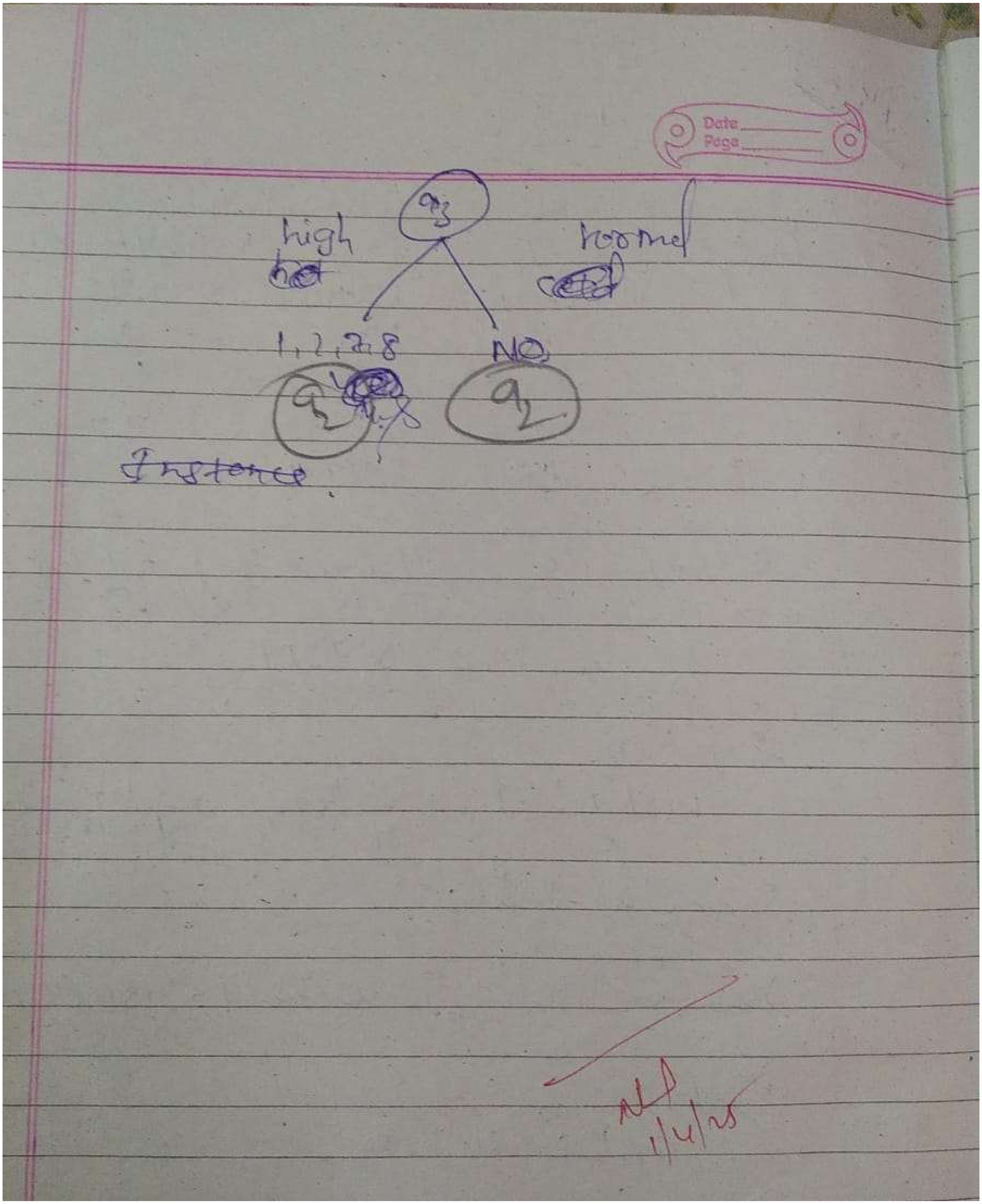


Figure 5.2

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:

1/9/25

LAB 5

Consider the following dataset for K=3 and data (x, 35, 100) as (Person, Age, Salary(K)) solve using KNN classify and predict the target.

Person	Age	SalaryK	Target	dist	R
A	18	50	N	52.8	
B	23	55	N	46.57	
C	29	70	N	31.95	2
D	41	60	Y	40.44	3
E	43	70	Y	31.04	1
F	38	40	Y	60.07	
X	35	100	? Yes		

K: 3.

For rank 1 → Yes

2 → No

3 → Yes

So overall Yes.

Figure 6.1

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.

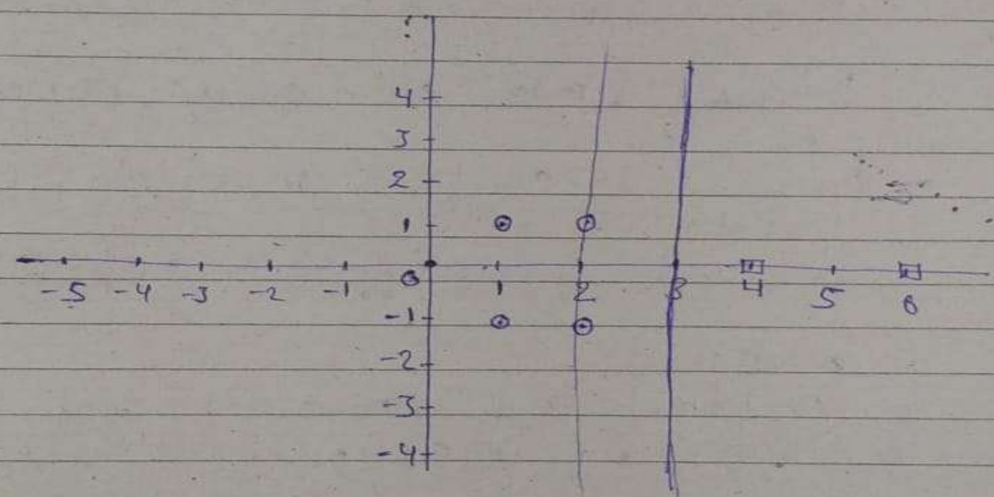
Observation:

8/4/25 LAB-7 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



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

$\bar{S}_1 = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} \quad \bar{S}_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} \quad \bar{S}_3 = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$

$$\alpha_1 \bar{S}_1 \cdot \bar{S}_1 + \alpha_2 \bar{S}_2 \cdot \bar{S}_1 + \alpha_3 \bar{S}_3 \cdot \bar{S}_1 = +1$$
$$\alpha_1 \bar{S}_1 \cdot \bar{S}_2 + \alpha_2 \bar{S}_2 \cdot \bar{S}_2 + \alpha_3 \bar{S}_3 \bar{S}_2 = +1$$
$$\alpha_1 \bar{S}_1 \bar{S}_3 + \alpha_2 \bar{S}_2 \bar{S}_3 + \alpha_3 \bar{S}_3 \bar{S}_3 = -1$$

Figure 7.1

$\alpha_1(6) + \alpha_2(4) + \alpha_3(9) = +1$

$\alpha_1(4) + \alpha_2(6) + \alpha_3(9) = +1$

$\alpha_1(9) + \alpha_2(9) + \alpha_3(17) = -1$

$\alpha_1 = 13/4, \quad \alpha_2 = 13/4, \quad \alpha_3 = -7/2$

$w = \alpha_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3$

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

$= -\begin{bmatrix}1\\0\\-3\end{bmatrix}$

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

$b + 3 = 0$

interception x-axis $= 3$

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

8-4-25

Figure 7.2

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))
```

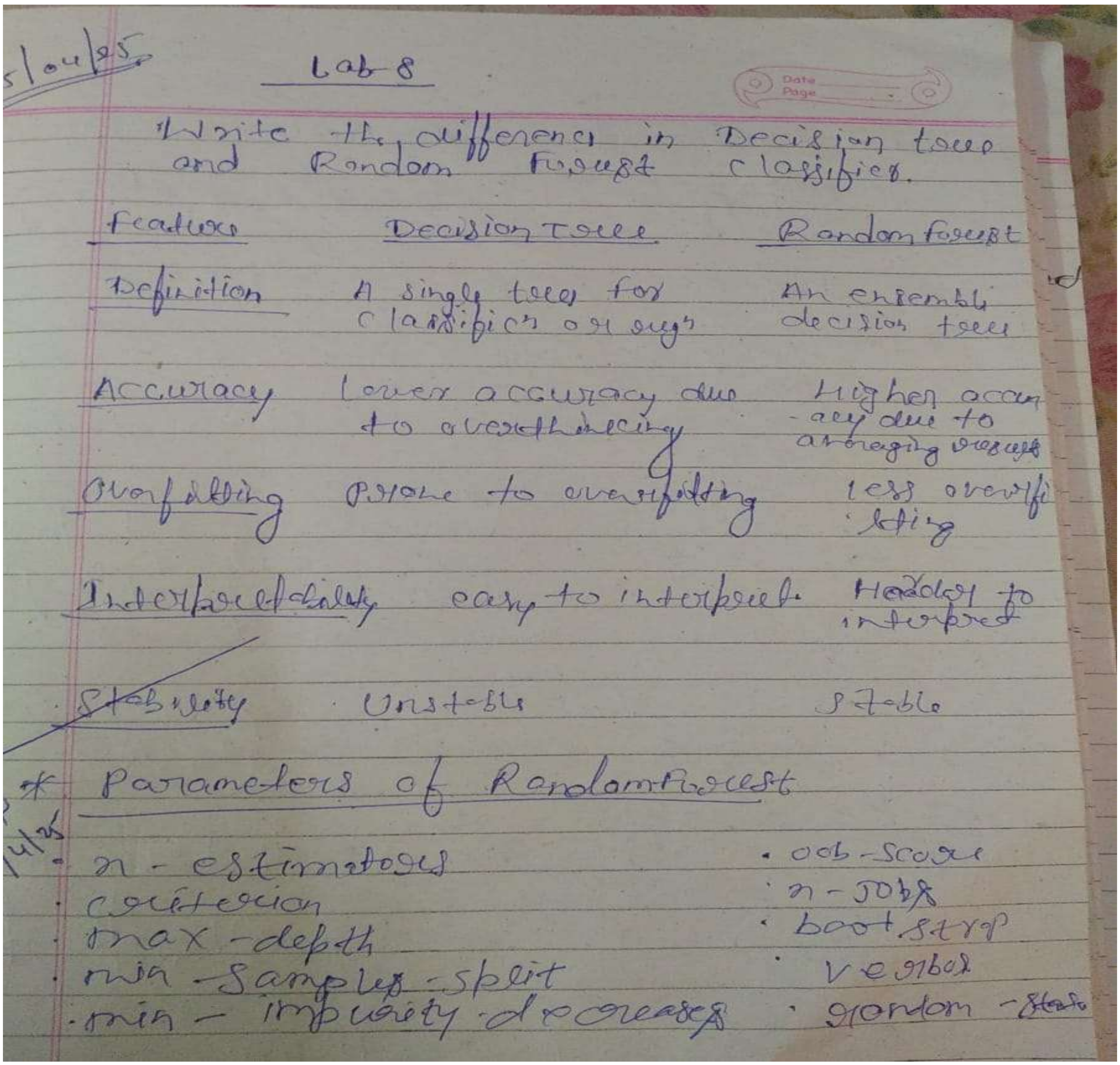
Digits.csv

```
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:



5/04/25

Lab 8

Write the difference in Decision tree and Random Forest classifier.

Features	Decision Tree	Random Forest
Definition	A single tree for classification or regr	An ensemble decision tree
Accuracy	Lower accuracy due to overfitting	Higher accuracy due to averaging results
Overfitting	Prone to overfitting	less overfitting
Interpretability	easy to interpret	Harder to interpret
Stability	Unstable	Stable

* Parameters of RandomForest

- n-estimators
- criterion
- max-depth
- min-samples-split
- min-impurity-decrease
- oob-score
- n-jobs
- bootstrap
- verbose
- random-state

Figure 8.1

* Algorithm of Random Forest classifier

1) Input: Dataset D with features X & labels Y, number of trees N.

2) For each tree (i=1 to N)

- Randomly select a bootstrap sample from the dataset
- Train a Decision Tree on this sample
- When splitting nodes:
  - → Instead of considering all features
  - → Select a random subset of feat
- Choose the best feature to split from this dataset
- Grow the tree fully or until a stopping criterion is met

3) Prediction

- For classification: each tree gives a prediction
- For regr: Average the o/p of all trees

4) Output: Final Prediction

Figure 8.2

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:

15/4/25

Date ____
Page ____

Lab 9

Bootstraping.

* What is bootstraping ?
> It is an ensemble learning technique that combines the prediction of several base cases to create a string classifier

* Parameters in AdaBoostClassifier()

- base - estimator
- n - estimators
- learning - rate
- algorithm
- random - state
- loss
- warm - start
- subsample
- max - depth

* AdaBoost Algo

1) Initialise weights
2) For each iteration:
   - Train a base learner
   - compute the error
   - compute the classifier's weight
   - update sample weights
3) Final model

Figure 9.3

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:

CIE-1

Starting → Kmeans, PCA

K-means

$A_1(2,10)$, $A_2(2,5)$, $A_3(8,4)$, $A_4(5,8)$,
$A_5(7,5)$, $A_6(6,4)$, $A_7(1,2)$, $A_8(4,9)$

Initial cluster centers are:
$A_1(2,10)$, $A_4(5,8)$, $A_7(1,2)$

Dist: $|x_1 - x_2| + |y_1 - y_2|$

Data points			Dist to 2, 10	5, 8	1, 2	Point belongs to
A_1	2	10	0	5	9	C1
A_2	2	5	5	6	4	C3
A_3	8	4	12	7	9	C2
A_4	5	8	5	0	10	C2
A_5	7	5	10	5	9	C2
A_6	6	4	10	5	7	C2
A_7	1	2	9	10	0	C3
A_8	4	9	3	2	10	C2

Center of cluster 2 $= ((8+5+7+6+4)/5,$
$(4+8+5+4+9)/5)$
$= (6,6)$

center of cluster3 = (2+1)/2 , (5+2)/2
= (1.5 , 3.5)

Data points			Dist to 2 10	Dist to 6 6	Dist to 1.5 3.5	Pts belong to cluster
A1	2	10	0	8	7	C1
A2	2	5	5	5	2	C3
A3	8	4	12	4	7	C2
A4	5	8	5	3	8	C2
A5	7	5	10	2	7	C2
A6	6	4	10	2	5	C2
A7	1	2	9	9	2	C3
A8	4	9	3	5	8	C1

Center of cluster1 = (2+4)/2 , (10+9)/2
= (3 , 9.5)

center of cluster2 = (8+5+7+6)/4 ,
(4+8+5+4)/4 = (6.5 , 5.25)

center of cluster3 = (2+1)/2 , (5+2)/2
= (1.5 , 3.5)

Data points			3, 9.5	6.5, 5.25	1.5, 3.5	cluster
A1	2	10	1.5	9.25	7	C1
A2	2	5	5.5	4.25	[illegible]	C3
A3	8	4	10.5	2.75	7	C2
A4	5	8	3.5	4.25	[illegible]	C1
A5	7	5	8.5	0.75	7	C2
A6	6	4	8.5	1.75	5	C2
A7	1	2	9.5	8.25	2	C3
A8	4	9	1.5	6.25	8	C1

$C_1 = (2+5+4)/3 = 3.66, \; (10+8+9)/3 = 9$

$C_2 = (8+7+6)/3 = 7, \; (4+5+4)/3 = 4.33$

$C_3 = (2+1)/2 = 1.5, \; (5+2)/2 = 3.5$

Data points			3.66, 9	7, 4.33	1.5, 3.5	cluster
A1	2	10	2.66	10.67	8	C1
A2	2	5	5.66	5.67	2	C3
A3	8	4	9.34	1.33	7	C2
A4	5	8	2.34	5.67	8	C1
A5	7	5	7.34	0.67	7	C2
A6	6	4	7.34	1.33	6	C2
A7	1	2	9.66	8.33	2	C3
A8	4	9	0.34	7.66	8	C1

==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 mean
2. calculation of covariance matrix
3. Eigenvalues of covariance matrix
4. Computation of the eigenvectors
   ... eigenvectors
5. Geometrical meaning of first principal component

Features	ex1	ex2	ex3	ex4
x_1	4	8	13	7
x_2	11	4	5	14

$\bar{x}_1 = 8 \qquad \bar{x}_2 = 8.5$

$$Cov(x_1, x_1) = \frac{1}{N-1} \sum_{k=1}^{N} (x_{1k} - \bar{x}_1)^2$$

$$= \frac{1}{3} \left[(4-8)^2 + (8-8)^2 + (13-8)^2 + (7-8)^2 \right]$$

$$= 14$$

$$Cov(x_1, x_2) = \frac{1}{N-1} \sum_{k=1}^{N} (x_{1k} - \bar{x}_1)(x_{2k} - \bar{x}_2)$$

$$= \frac{1}{3} \left[(4-8)(11-8.5) + (8-8)(4-8.5) + (13-8)(5-8.5) + (7-8)(14-8.5) \right]$$

$$= -11$$

$$Cov(x_2, x_1) = -11$$

$$\mathrm{cov}(x_2, x_2) = \frac{1}{N-1} \sum_{k=1}^{N} (x_{2k} - \bar{x}_2)^2$$

$$= 23$$

Cov matrix $= \begin{bmatrix} \mathrm{cov}(x_1, x_1) & \mathrm{cov}(x_1, x_2) \\ \mathrm{cov}(x_2, x_1) & \mathrm{cov}(x_2, x_2) \end{bmatrix}$

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

Finding eigen values

$$0 = \det[S - \lambda I]$$

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

$$= (14-\lambda)(23-\lambda) - 121 = 0$$

$$322 - 37\lambda + \lambda^2 - 121 = 0$$

$$= \lambda^2 - 37\lambda + 201 = 0$$

$$\lambda_1 = 30.38 \quad , \lambda_2 = 6.615$$

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

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

$$\begin{bmatrix} 14-\lambda_1 & -11 \\ -11 & 23-\lambda_1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

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

$$-11u_1 + (23 - \lambda_1)u_2 = 0$$

~~= -11u_1 + (23~~

$(14 - 30.38)u_1 - 11u_2 = 0$

$-11u_1 + (23 - 30.38)u_2 = 0$

$$\frac{u_1}{11} = \frac{u_2}{14-\lambda_1} = t_1$$

$u_1 = 11t_1 \qquad u_2 = (14-\lambda_1)t_1$

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

$$\|U_1\| = \sqrt{11^2 + (14-\lambda_1)^2}$$

$$= \sqrt{11^2 + (14-30.38)^2}$$

$$= 19.7348$$

$$e_1 = \begin{bmatrix} 11/\|U_1\| \\ (14-\lambda_1)/\|U_1\| \end{bmatrix} = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$$

for e_2

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

$(14 - \lambda_2)u_1 - 11u_2 = 0$

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

$$e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$$

let $\begin{bmatrix} x_{1k} \\ x_{2k} \end{bmatrix}$

$$e_1^T = \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix}$$

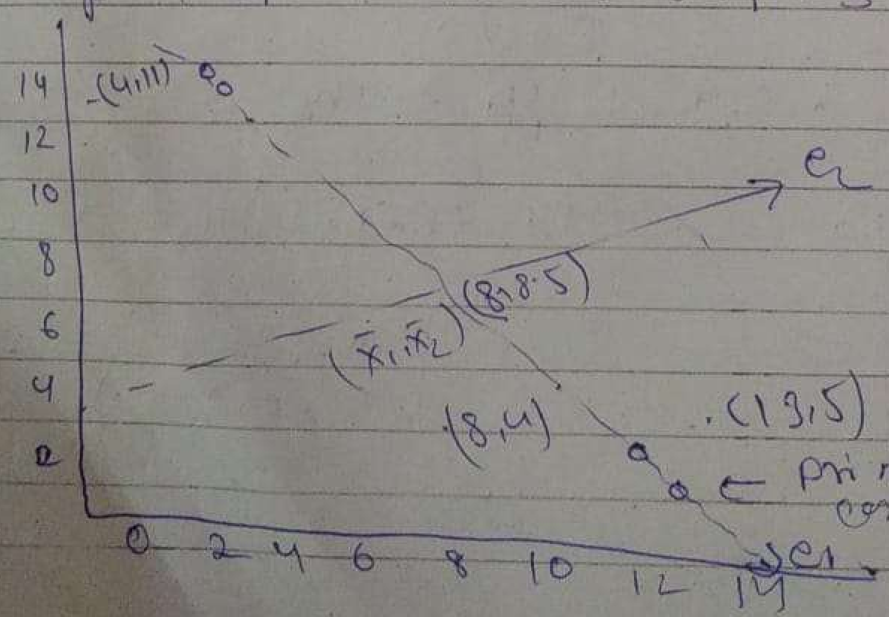
$$= [0.5574 \quad -0.8303] \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix}$$

$$= 0.5574(x_{1k} - \bar{x}_1) - 0.8303(x_{2k} - \bar{x}_2)$$
$$= 0.55(4-8) - 0.83(11-8.5)$$
$$-4.3052$$

$$\begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} = \begin{bmatrix} 4 \\ 11 \end{bmatrix}$$

find

x_1	4	8	13	7
~~x_2~~	~~-4.3~~	~~3.7~~	~~5.69~~	~~-5.12~~
x_2 :	11	4	5	14
P comp : ~~e_1~~	-4.3	3.7	5.69	-5.12



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}")
```