VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Madhu Sarika (1BM22CS140)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2024 to Jan-2025

B.M.S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by **Madhu Sarika (1BM22CS140),** 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. Pallavi G B Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Sl. No.	Date	Experiment Title	Page No.
1	21-2-2025	Write a python program to import and export data using Pandas library functions	4
2	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	5
3	10-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	9
4	17-3-2025	Build Logistic Regression Model for a given dataset	15
5	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	19
6	7-4-2025	Build KNN Classification model for a given dataset.	22
7	21-4-2025	Build Support vector machine model for a given dataset	25
8	5-5-2025	Implement Random Forest ensemble method on a given dataset.	27
9	5-5-2025	Implement Boosting ensemble method on a given dataset.	29
10	12-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	31
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	33

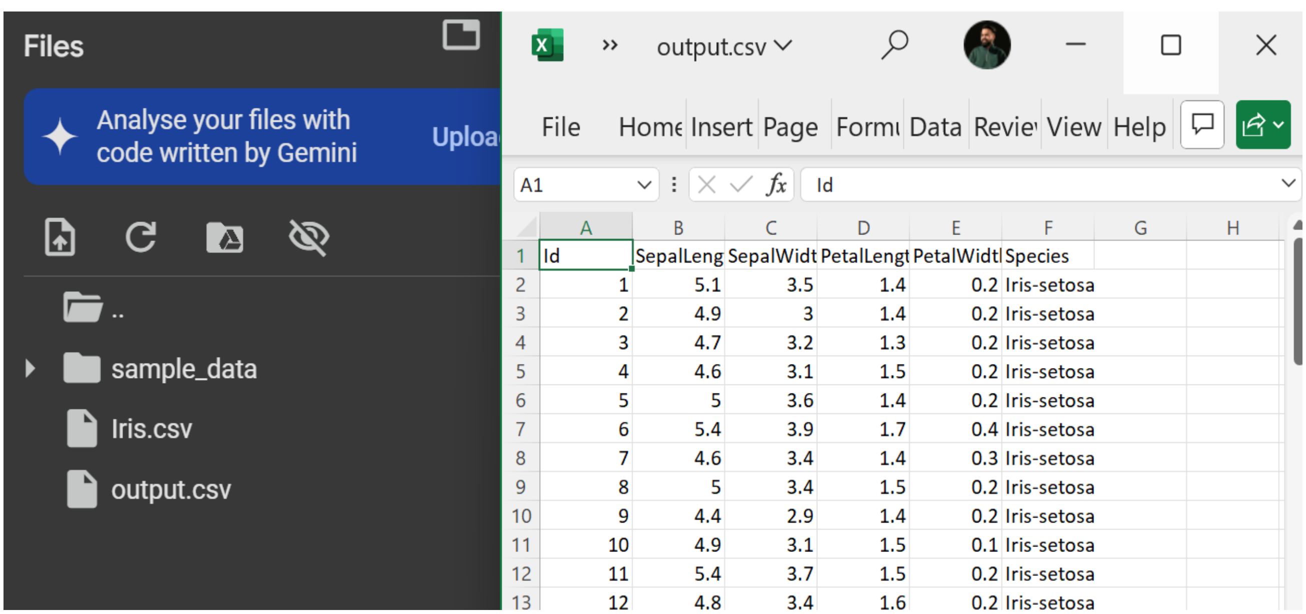
Github Link:

https://github.com/madhupandeyy/6th-Sem-ML-LAB

Program 1

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

```
import pandas as pd

# Import data from a CSV file

data = pd.read_csv('/content/Iris.csv')

print("Imported Data:")

print(data.head())  # Show first 5 rows

# Display info about the dataset

print("\nDataset Info:")

print(data.info())

# Export the data to a new CSV file

data.to_csv('output.csv', index=False)

print("\nData exported successfully to 'output.csv'")
```



	A	B	C	D	E	F	G	H
1	Id	SepalLeng	SepalWidt	PetalLengt	PetalWidtl	Species		
2	1	5.1	3.5	1.4	0.2	Iris-setosa		
3	2	4.9	3	1.4	0.2	Iris-setosa		
4	3	4.7	3.2	1.3	0.2	Iris-setosa		
5	4	4.6	3.1	1.5	0.2	Iris-setosa		
6	5	5	3.6	1.4	0.2	Iris-setosa		
7	6	5.4	3.9	1.7	0.4	Iris-setosa		
8	7	4.6	3.4	1.4	0.3	Iris-setosa		
9	8	5	3.4	1.5	0.2	Iris-setosa		
10	9	4.4	2.9	1.4	0.2	Iris-setosa		
11	10	4.9	3.1	1.5	0.1	Iris-setosa		
12	11	5.4	3.7	1.5	0.2	Iris-setosa		
13	12	4.8	3.4	1.6	0.2	Iris-setosa		

Program 2

Demonstrate various data pre-processing techniques for a given dataset

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

# Function to check missing values
def check_missing(df, name):
    if df.isnull().any().any():
        print(f"Missing values detected in {name}:")
        print(df.isnull().sum())
    else:
        print(f"No missing values in {name}\n")

# ===================== Diabetes Dataset =====================

# 1. Load diabetes dataset
print("Loading Diabetes dataset...")
diabetes = pd.read_csv('/content/Dataset of Diabetes .csv')
print(diabetes.head())  # Show first few rows
print(f"Shape after load: {diabetes.shape}\n")

# 2. Check for missing values before cleaning
check_missing(diabetes, 'Diabetes')

# 3. Drop any rows with missing values
diabetes.dropna(inplace=True)
print("After dropping missing values:")
print(f"Shape: {diabetes.shape}\n")
check_missing(diabetes, 'Diabetes')

# 4. Encode categorical columns ('Gender' and 'CLASS') with label encoding
le_diab = LabelEncoder()
diabetes['Gender'] = le_diab.fit_transform(diabetes['Gender'])
diabetes['CLASS']  = le_diab.fit_transform(diabetes['CLASS'])
print("After label- encoding 'Gender' and 'CLASS':")
print(diabetes[['Gender','CLASS']].head(), "\n")

# 5. Handle outliers using the IQR method for numerical columns
numerical_cols = ['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']
for col in numerical_cols:
    Q1  = diabetes[col].quantile(0.25)
    Q3  = diabetes[col].quantile(0.75)
    IQR = Q3 - Q1
```

```
    # Clip values to [Q1 - 1.5*IQR, Q3 + 1.5*IQR]
    diabetes[col] = np.clip(diabetes[col],
                            Q1 - 1.5 * IQR,
                            Q3 + 1.5 * IQR)
print("After outlier clipping (IQR) on numerical columns:")
print(diabetes[numerical_cols].describe(), "\n")

# 6. Separate features and target
X_diab = diabetes.drop(['ID', 'No_Pation', 'CLASS'], axis=1)
y_diab = diabetes['CLASS']
print("Features and target separated for Diabetes:")
print(f"  X_diab shape: {X_diab.shape}")
print(f"  y_diab shape: {y_diab.shape}\n")

# 7. Scale feature data using MinMaxScaler and StandardScaler
minmax       = MinMaxScaler()
std_scaler   = StandardScaler()

X_diab_minmax   = minmax.fit_transform(X_diab)
print("After MinMax scaling Diabetes features:")
print(f"  X_diab_minmax shape: {X_diab_minmax.shape}\n")

X_diab_standard = std_scaler.fit_transform(X_diab)
print("After Standard scaling Diabetes features:")
print(f"  X_diab_standard shape: {X_diab_standard.shape}\n")

# ===================== Adult Income Dataset =====================

# 1. Load adult income dataset
print("Loading Adult Income dataset...")
adult = pd.read_csv('/content/adult.csv')
print(adult.head())
print(f"Shape after load: {adult.shape}\n")

# 2. Replace '?' entries with NaN for proper missing- value handling
adult.replace('?', np.nan, inplace=True)
print("After marking '?' as NaN:")
check_missing(adult, 'Adult Income')

# 3. Identify categorical vs numerical columns
raw_cat_cols = adult.select_dtypes(include=['object']).columns.tolist()
raw_num_cols = adult.select_dtypes(include=['int64','float64']).columns.tolist()
print("Column types detected:")
print(f"  Categorical cols: {raw_cat_cols}")
print(f"  Numerical cols:   {raw_num_cols}\n")

# 4. Fill missing values for features (exclude 'income' target)
for col in raw_cat_cols:
```

```
    if col != 'income':
        adult[col] = adult[col].fillna(adult[col].mode()[0])
for col in raw_num_cols:
    adult[col] = adult[col].fillna(adult[col].mean())
print("After filling missing values in features:")
check_missing(adult, 'Adult Income')

# 5. Encode the target column 'income' using LabelEncoder
le_adult = LabelEncoder()
adult['income'] = le_adult.fit_transform(adult['income'])
print("After label- encoding 'income' target:")
print(adult['income'].value_counts(), "\n")

# 6. Separate features and target
X_adult = adult.drop('income', axis=1)
y_adult = adult['income']
print("Features and target separated for Adult Income:")
print(f"  X_adult shape: {X_adult.shape}")
print(f"  y_adult shape: {y_adult.shape}\n")

# 7. One- hot encode all categorical feature columns
cat_features = [c for c in raw_cat_cols if c != 'income']
X_adult = pd.get_dummies(X_adult, columns=cat_features)
print("After one- hot encoding categorical features:")
print(f"  X_adult shape: {X_adult.shape}\n")

# 8. Handle outliers on numeric features using IQR clipping
num_features = X_adult.select_dtypes(include=[np.number]).columns.tolist()
for col in num_features:
    Q1  = X_adult[col].quantile(0.25)
    Q3  = X_adult[col].quantile(0.75)
    IQR = Q3 - Q1
    X_adult[col] = np.clip(X_adult[col],
                           Q1 - 1.5 * IQR,
                           Q3 + 1.5 * IQR)
print("After outlier clipping on Adult numeric features:")
print(X_adult[num_features].describe(), "\n")

# 9. Scale adult features with the same scalers
X_adult_minmax   = minmax.fit_transform(X_adult)
print("After MinMax scaling Adult features:")
print(f"  X_adult_minmax shape: {X_adult_minmax.shape}\n")

X_adult_standard = std_scaler.fit_transform(X_adult)
print("After Standard scaling Adult features:")
print(f"  X_adult_standard shape: {X_adult_standard.shape}")
```

Date 3/3/25
Page 1

Week - 1

1) import pandas as pd

2) df = pd.read_csv("housing.csv")

Display information of all columns :
3) print("Column Information")
print(df.info())

4) df.describe()
print("\n Statistical Information of Numerical Columns:")

O/p :

	RM	LSTAT	~~MEDV~~
count	489.000000	489.000000	~~489.000000~~
mean	6.240288	12.9396	
std	0.644	7.08199	
min	3.56100	1.98000	
25%	5.88	7.37000	
50%	6.18	11.09	

5) print("\n Unique Value counts for '~~ocean proximity~~' LSTAT Column :")
print(df["~~ocean-proximity~~" LSTAT].value_counts())

Display columns with missing values count greater than zero

6) missing_values = df.isnull().sum()
missing_values = missing_values[missing_values > 0]

O/p :

	Column	Non-null	count
0	RM	489	non-null
1	LSTAT	489	non-null
2	MEDV	489	non-null

Count of unique labels

LSTAT	
8.05	3
7.79	3
6.36	3

Date __/__/__
Page 2

Handling Missing Value :

Adult ~~Dataset~~ dataset :
Columns with missing values : workclass, occupation, and native country

Used mode imputation for categorical and median imputation for numerical ones.

Diabetes dataset :
Numerical column like BMI had missing value.

Encoding categorical variables :

Diabetes dataset : The dataset mainly contained numerical values, if any present, they were handled using ordinal encoding.

Adult income dataset -
Categorical columns: workclass, education.
Applied hot encoding for multiple categories and ordinal encoding for binary.

Comparison of min-max scaling and standardisation

Min-max scaling -
• Transform datavalues into fixed range (0 to 1).

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

X = original feature value
X_{min} = min feature value
X_{max} = max feature value
X' - transformed feature value

Date __/__/__
Page 3

• Suitable for distance-based algorithms

Standardisation

→ Transform data to have a mean of 0 and a standard deviation of 1.
→ Applied to BMI in the diabetes dataset and age in adult income dataset.
→ Preferred for models like linear regression.

3/3/25

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from io import StringIO

# Load dataset from provided content
df = pd.read_csv("/content/housing.csv")
print("Dataset loaded successfully!")
df.head()

# Perform describe() and info()
print("Data Information:")
print(df.info())
print("\nData Description:")
print(df.describe())

# Plot Histograms
df.hist(bins=50, figsize=(20,15))
plt.show()

# Create a Stratified Test Set
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit

# Random split
train_set_random, test_set_random = train_test_split(df, test_size=0.2, random_state=42)

# Stratified split based on income
df["income_cat"] = pd.cut(df["median_income"],
                          bins=[0., 1.5, 3.0, 4.5, 6., float("inf")],
                          labels=[1, 2, 3, 4, 5])
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(df, df["income_cat"]):
    strat_train_set = df.loc[train_index]
    strat_test_set  = df.loc[test_index]

# Drop the stratification column
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)

# Geographical Visualization
plt.figure(figsize=(10, 7))
plt.scatter(df["longitude"], df["latitude"], alpha=0.4,
            c=df["median_house_value"], cmap="jet", s=10)
plt.colorbar(label="Median House Value ($)")
```

```python
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Housing Prices by Location")
plt.grid(True)
plt.show()

# Feature Correlation
corr_matrix = df.corr(numeric_only=True)
corr_matrix["median_house_value"].sort_values(ascending=False)

# Plot correlation matrix heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix")
plt.show()

# Plot most correlated feature vs price
df.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.2)
plt.show()

# Data Cleaning: Fill missing total_bedrooms
median = df["total_bedrooms"].median()
df["total_bedrooms"] = df["total_bedrooms"].fillna(median)

# Combine Features to Improve Correlation
df["rooms_per_household"]      = df["total_rooms"] / df["households"]
df["bedrooms_per_room"]        = df["total_bedrooms"] / df["total_rooms"]
df["population_per_household"] = df["population"] / df["households"]

# Handle Categorical Data
from sklearn.preprocessing import OneHotEncoder
cat_encoder       = OneHotEncoder()
ocean_prox_1hot   = cat_encoder.fit_transform(df[["ocean_proximity"]])
ocean_prox_1hot.toarray()

##############################################################################
#                            ### ADDED SECTION ###                           #
#           Simple & Multiple Linear Regression with Visualization           #
##############################################################################

from sklearn.linear_model    import LinearRegression
from sklearn.metrics         import mean_squared_error, r2_score
from sklearn.impute          import SimpleImputer

# 1. Prepare feature matrix X and target vector y
X = df.drop(["median_house_value", "ocean_proximity"], axis=1)
y = df["median_house_value"]
```

```python
# 2. Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 3. Impute any remaining missing values using median strategy
imputer = SimpleImputer(strategy="median")
imputer.fit(X_train)
X_train = pd.DataFrame(imputer.transform(X_train), columns=X_train.columns)
X_test  = pd.DataFrame(imputer.transform(X_test),  columns=X_test.columns)

# 4. Simple Linear Regression using median_income
lin_reg = LinearRegression()
lin_reg.fit(X_train[["median_income"]], y_train)
y_pred_simple = lin_reg.predict(X_test[["median_income"]])

mse_simple = mean_squared_error(y_test, y_pred_simple)
r2_simple  = r2_score(y_test, y_pred_simple)
print(f"Simple Linear Regression:\n  MSE: {mse_simple:.2f}\n  R²: {r2_simple:.3f}\n")

# 4a. Plot Simple Regression
plt.figure(figsize=(8,6))
plt.scatter(X_test["median_income"], y_test, alpha=0.3, label="Actual")
plt.plot(
    X_test["median_income"], y_pred_simple,
    "r-", linewidth=2, label="Predicted"
)
plt.xlabel("Median Income")
plt.ylabel("Median House Value")
plt.title("Simple Linear Regression: Income vs. House Value")
plt.legend()
plt.show()

# 5. Multiple Linear Regression using all numerical features
multi_reg = LinearRegression()
multi_reg.fit(X_train, y_train)
y_pred_multi = multi_reg.predict(X_test)

mse_multi = mean_squared_error(y_test, y_pred_multi)
r2_multi  = r2_score(y_test, y_pred_multi)
print(f"Multiple Linear Regression:\n  MSE: {mse_multi:.2f}\n  R²: {r2_multi:.3f}\n")

# 5a. Plot Multiple Regression (Predicted vs Actual)
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred_multi, alpha=0.3)
plt.plot(
    [y_test.min(), y_test.max()],
    [y_test.min(), y_test.max()],
```

```python
    "k--", linewidth=2
)
plt.xlabel("Actual Median House Value")
plt.ylabel("Predicted Median House Value")
plt.title("Multiple Linear Regression: Predicted vs. Actual")
plt.show()

# 6. Inspect coefficients for multiple regression
coeff_df = pd.DataFrame({
    "Feature":     X.columns,
    "Coefficient": multi_reg.coef_
}).sort_values(by="Coefficient", ascending=False)

print("Top coefficients in multiple regression:")
print(coeff_df.head())
```

Date 10/3/25
Page 4

LAB-2

Demonstrate the steps to build machine learning model that predicts housing price using California housing price dataset.

1. Perform the describe and info steps
df.describe()
df.info()

2. Plot histogram of each feature
Indicate what does histogram indicate on median-income and house-median-age

```
import matplotlib.pyplot as plt
df.hist(figsize(12,8), bins = 50 , edgecolor="black")
plt.tight_layout()      // automatically adjust
                        the spacing b/w subplots
                        to prevent overlapping.
```

Observations:

median-income : Right-skewed, most values within 2 to 5.

house-median-age : Evenly spread, peaks around 10-20 and 30-40, max at 52.

3. Demonstrate the process of creating a test set.
Write the difference b/w random and stratified test-set

Date __/__/__
Page 5

Random-Sampling (train-test-split)	Stratified Sampling (StratifiedShuffleSplit)
• Splits the dataset randomly without any specific feature distribution.	• ensures that distribution of an important feature ("income-cat") is preserved in both training and test sets.
• Can lead to imbalances in key features, meaning that important subgroups (e.g., diff income levels) may not be represented properly in the test set.	• Prevents over representation or under representation of certain categories, leading to more reliable test set.

4. What does the graph indicate w.r.t housing prices and location?

Higher house prices are near the coastal regions. Lower prices appear inland.

5. Which feature correlates to the maximum. Plot the graph for that with housing price and analyse what the graph indicate

Median income has the highest positive correlation with house price.

6. List the features that could be combined to improve correlation

df["rooms-per-household"] = df["total-rooms"]/df["households"]

Date / / Page 6

df["population-per-household"] = df["population"] / df["households"]

7. Datasets already have all the values ; so no need to be cleaned.

8. Is there any categorical data that needs to be converted to numerical ?

Yes, Ocean Proximity needs to be converted to numerical data.
Machine learning models can not directly process categorical data, so we convert it into numerical form using One-Hot Encoding.

One-Hot Encoding creates binary columns for each unique category in the categorical feature. eg:- if 'ocean-proximity' has categories:
"NEAR OCEAN", "LESS THAN 1H OCEAN", "NEAR BAY", "INLAND", "ISLAND",
One-Hot encoding will convert it into four separate columns with 1s and 0s indicating the category.

9. Importance of feature Scaling.

- → Brings all numerical values to a common scale.
- → Improves model performance by avoiding features dominating others.

Date / / Page 7

10. explain ~~how~~ "Custom Transform, feature scaling and encoding" and how it works.

The pipeline consists of three main types:

Custom Transformation (Handling Missing Values) - SimpleImputer (strategy = "median") replaces missing values with the median to prevent data loss.

Feature Scaling - StandardScaler() standardizes numerical features to have a mean of 0 and std dev of 1, ensuring all features contribute equally to the model.

Encoding categorical data - One hot encoder (handle-unknown = "gnore") converts categorical into numerical.

The pipeline first fills missing values in numerical cols using SimpleImputer and then it scales numerical features to have a mean 0 and then categorical.

Date 7/3/25 Page 8

LAB-2

Find linear regression of the data of ~~week and~~ :-

Diameter(X) in inches	Price(Y) in Dollars
8	10
10	13
12	16

Que. Predict the price of 20 inch pizza using the data given :

$$\beta = ((X^TX)^{-1}X^T)Y \qquad X = \begin{bmatrix} 1 & 8 \\ 1 & 10 \\ 1 & 12 \end{bmatrix}$$

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

We want to fit the model $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$

1. (X^TX) :

$$\underset{2\times3}{\begin{bmatrix} 1 & 1 & 1 \\ 8 & 10 & 12 \end{bmatrix}} \times \underset{3\times2}{\begin{bmatrix} 1 & 8 \\ 1 & 10 \\ 1 & 12 \end{bmatrix}} = \underset{2\times2}{\begin{bmatrix} 3 & 30 \\ 30 & 308 \end{bmatrix}}$$

2. $(X^TX)^{-1}$

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \text{ then } A^{-1} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

A^{-1} exist only when $ad - bc \neq 0$

Date / / Page 9

$$\begin{bmatrix} 3 & 30 \\ 30 & 308 \end{bmatrix}^{-1} = \frac{1}{(3\times308) - (900)}\begin{bmatrix} 308 & -30 \\ -30 & 3 \end{bmatrix}$$

$$= \frac{1}{24}\begin{bmatrix} 308 & -30 \\ -30 & 3 \end{bmatrix} = \begin{bmatrix} 12.83 & -1.25 \\ -1.25 & 0.125 \end{bmatrix}$$

3. $((X^TX)^{-1}X^T) = \underset{2\times2}{\begin{bmatrix} 12.83 & -1.25 \\ -1.25 & 0.125 \end{bmatrix}} \times \underset{2\times3}{\begin{bmatrix} 1 & 1 & 1 \\ 8 & 10 & 12 \end{bmatrix}}$

$$= \begin{bmatrix} 2.83 & 0.33 & -2.17 \\ -0.25 & 0 & 0.25 \end{bmatrix}$$

4. $((X^TX)^{-1}X^T) \cdot Y$

$$\underset{2\times3}{\begin{bmatrix} 2.83 & 0.33 & -2.17 \\ -0.25 & 0 & 0.25 \end{bmatrix}} \underset{3\times1}{\begin{bmatrix} 10 \\ 13 \\ 16 \end{bmatrix}}$$

$$= \begin{bmatrix} -2.13 \\ 1.5 \end{bmatrix} \qquad \beta_0 = -2.13, \quad \beta_1 = 1.5$$

Final Regression eq : $y = -2.13 + 1.5x$

Prediction for a 20 inch pizza

$$y = -2.13 + 1.5(20)$$
$$= -2.13 + 30 = 27.87$$

Date / /
Page 10

Q. For hiring.csv, what is the predicted salary for a candidate with 12 years of experience, 10 test score and 10 interview score?

The predicted salary is $58,227.64.

Yes, "state was encoded using One-hot encoding ('OneHotEncoder(drop='first')').

This converts state into binary columns and remove one to avoid redundancy.

Q. For 1000_companies.csv. Did you encode categorical variables (e.g state)? If yes, how? Did you scale features? If yes, why?

No feature scaling was not applied because linear regression does not need it.
• The model directly weight to each feature.

17-3-25

Program 4

Build Logistic Regression Model for a given dataset

```
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

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

# Scatter plot: Employee satisfaction vs Retention
plt.scatter(df.satisfaction_level, df.left, marker='+', color='red')
plt.xlabel("Satisfaction Level")
plt.ylabel("Left (1) / Stayed (0)")
plt.title("Impact of Satisfaction Level on Employee Retention")
plt.show()

# Define features (X) and target (y)
X = df[['satisfaction_level']]
y = df['left']

# Split dataset (90% train, 10% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.9,
random_state=10)

# Train logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_predicted = model.predict(X_test)

# Model Accuracy
print(f"Model Accuracy: {model.score(X_test, y_test):.4f}")

# Probability predictions
print("Predicted Probabilities:")
print(model.predict_proba(X_test))

# Predict for a specific satisfaction level (e.g., 0.4)
predicted_status = model.predict([[0.4]])
print(f"Prediction for Satisfaction Level 0.4: {'Left' if predicted_status[0]
== 1 else 'Stayed'}")

# Logistic function
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

# Custom prediction function
m, b = model.coef_[0][0], model.intercept_[0]
def prediction_function(satisfaction):
    z = m * satisfaction + b
```

```
    y = sigmoid(z)
    return y

satisfaction_test = 0.4
print(f"Sigmoid Prediction for Satisfaction Level {satisfaction_test}:
{prediction_function(satisfaction_test):.4f}")
```

```python
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Load the Zoo dataset
file_path = "/content/zoo-data.csv"
zoo_data = pd.read_csv(file_path)

# Drop the 'animal_name' column as it is not a relevant feature
X = zoo_data.drop(['animal_name', 'class_type'], axis=1)  # Features
y = zoo_data['class_type']  # Target variable

# Split the dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the Logistic Regression model for multi-class classification
model = LogisticRegression(multi_class='multinomial', solver='lbfgs',
max_iter=200)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Multinomial Logistic Regression model:
{accuracy:.2f}")

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Adjust display labels to match actual present labels in the test set
unique_classes_in_test = sorted(y_test.unique())

# Display confusion matrix
cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=unique_classes_in_test)
cm_display.plot(cmap='Blues', xticks_rotation=45)
plt.show()
```

Date 7/3/25
Page 13

Program 3

Build logistic regression model for given dataset

Pseudocode

BEGIN
LOAD dataset

PREPROCESS data:
- Handle missing values
- Encode categorical features if needed
- Normalise features
- Split into: x-train, y-train
  - x-test, y-test

DEFINE logistic Regression model:

y = sigmoid(z)
where $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots \beta_n x_n$
$sigmoid(z) = \frac{1}{1+e^{(-z)}}$

TRAIN the model using Gradient Descent:

Initialise $\beta = [0, 0, \dots 0]$ for all weights

For epoch = 1 to max-epochs:
z = x-train · β
y-pred = sigmoid(z)

Compute gradient:
gradient = (1/n) * $X\text{-train}^T$ · (y-pred - y-train)

Update weights:

β = β - learning rate * gradient

PREDICT on test data:

z-test = x-test * β
y-prob = sigmoid(z-test)
y-pred = 1 if y-prob ≥ 0.5 else 0

Evaluate model:
Accuracy = (number of correct predictions) / (Total predictions)

Confusion matrix:

- TP: True Positive
- TN: True Negative
- FP: False Positive
- FN: " Negative

END

Program 5

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

```
import numpy as np
import pandas as pd
from collections import Counter

class Node:
    def __init__(self, feature=None, value=None, label=None):
        self.feature = feature  # Attribute to split on
        self.value = value      # Value of the attribute
        self.label = label      # Label if it’s a leaf node
        self.children = {}      # Dictionary of child nodes


def entropy(y):
    counts = np.bincount(y)
    probabilities = counts / len(y)
    return -np.sum([p * np.log2(p) for p in probabilities if p > 0])

def information_gain(X, y, feature):
    total_entropy = entropy(y)
    values, counts = np.unique(X[:, feature], return_counts=True)
    weighted_entropy = sum((counts[i] / sum(counts)) * entropy(y[X[:,
feature] == v]) for i, v in enumerate(values))
    return total_entropy - weighted_entropy

def best_feature_to_split(X, y):
    gains = [information_gain(X, y, i) for i in range(X.shape[1])]
    return np.argmax(gains)

def id3(X, y, features):
    if len(set(y)) == 1:
        return Node(label=y[0])
    if len(features) == 0:
        return Node(label=Counter(y).most_common(1)[0][0])
    best_feature = best_feature_to_split(X, y)
    node = Node(feature=features[best_feature])
    feature_values = np.unique(X[:, best_feature])
    for value in feature_values:
        sub_X = X[X[:, best_feature] == value]
        sub_y = y[X[:, best_feature] == value]
        if len(sub_y) == 0:
            node.children[value] =
Node(label=Counter(y).most_common(1)[0][0])
        else:
            node.children[value] = id3(np.delete(sub_X, best_feature,
axis=1), sub_y, features[:best_feature] + features[best_feature+1:])
    return node

def print_tree(node, depth=0):
    if node.label is not None:
```

```python
        print(f"{'  ' * depth}Leaf: {node.label}")
        return
    print(f"{'  ' * depth}Feature: {node.feature}")
    for value, child in node.children.items():
        print(f"{'  ' * depth}Value: {value}")
        print_tree(child, depth + 1)

# Example dataset
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain',
'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast',
'Rain'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool',
'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal',
'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong',
'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes',
'Yes', 'Yes', 'Yes', 'Yes', 'No']
})

X = data.iloc[:, :-1].apply(lambda col: pd.factorize(col)[0]).to_numpy()
y = pd.factorize(data['PlayTennis'])[0]
features = list(data.columns[:-1])

decision_tree = id3(X, y, features)

print_tree(decision_tree)
```

LAB-4

Date 24.3.25
Page 17

DECISION Trees Using ID3 Algorithm

Pseudocode

BEGIN
LOAD dataset
PREPROCESS data :
- Handle missing values
- Encode categorical variables (if needed)
- split data into :
X-train, y-train, X-test, y-test

FUNCTION InformationGain (S, A):

(c) $IG(S,A) = Entropy(S) - \Sigma \left(\frac{|S_v|}{|S|}\right) \times Entropy(S_v)$

(b) Entropy (S, Sv)

where :
- S is the full dataset
- Sv is the subset of S where attribute A = Value v

FUNCTION Entropy(S):

$Entropy(S) = -\Sigma\, p_i \log_2 (p_i)$ — (a)

where p_i is proportion of class i in set S.

FUNCTION
CONSTRUCT ID3 :

- Compute Entropy (S) eqn (a) for whole training dataset.

- Compute entropy (S, Sv) eq (b) and Information gain (S,A) eq (c) for each attribute in the training set.

Date
Page 18

- Choose the attribute for which entropy is minimum and Information gain is maximum.

- The best split attribute is placed as the root node

- The root node is branched into subtrees with each subtree as an outcome of test condition of the root node. Accordingly training set is also split into subsets.

- Recursively apply same opn for subset of training set with remaining attributes until a leaf node is derived or no more training ~~set~~ instances are available in the subset.

Program 6

Build KNN Classification model for a given dataset.

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

# Function to train and evaluate KNN model
def knn_classification(data_path, target_column, dataset_name, k=5):
    # Load dataset
    df = pd.read_csv(data_path)

    # Split features and target
    X = df.drop(columns=[target_column])
    y = df[target_column]

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

    # Feature scaling for better performance
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Train KNN model
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Evaluate model
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy of KNN on {dataset_name} dataset: {accuracy:.4f}')
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix - {dataset_name}')
    plt.show()

# Run KNN classification on both datasets
knn_classification('/content/iris.csv', 'species', 'Iris', k=5)

knn_classification('/content/diabetes.csv', 'Outcome', 'Diabetes', k=5)
```

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

# Load dataset
df = pd.read_csv('/content/heart (1).csv')

# Define features and target
X = df.drop(columns=['target'])  # Assuming 'target' is the classification
column
y = df['target']

# Split data
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 = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Find the best K value
k_values = range(1, 21)
accuracy_scores = []
for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy_scores.append(accuracy_score(y_test, y_pred))

best_k = k_values[np.argmax(accuracy_scores)]
print(f'Best K value: {best_k}')

# Train model with best K
best_model = KNeighborsClassifier(n_neighbors=best_k)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy with best K ({best_k}): {accuracy:.4f}')
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix - KNN (K={best_k})')
plt.show()
```

```python
# Plot K values vs. Accuracy
plt.plot(k_values, accuracy_scores, marker='o')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.title('K Value vs Accuracy')
plt.show()
```

LAB - 5

Date 7/4/25
Page 19

K- Nearest Neighbours Classification

Pseudocode

```
BEGIN
  LOAD dataset
  PREPROCESS data:
     - Handle missing values
     - Encode categorical variables
     - Normalise features
     - Split data into training and testing set

  DEFINE KNN classifier:
     - Choose number of neighbours (K)

  FUNCTION EuclideanDistance(x1, x2):
      distance = sqrt( Σ (x1_i - x2_i)^2 )
      RETURN distance

  FUNCTION PredictClass(test-sample, training-data, K):
      FOR each sample in training-data:
         Compute distance = EuclideanDistance(testsample, train sample)
         store(distance, class-label)

      Sort all distances in asc order
      Select top k nearest neighbours
      Count the freq of class labels among k neighbours.
      Return the class label with highest freq.

  EVALUATE model:
      Accuracy = (no of correct predictions) / (Total Prediction)
END
```

Page 11

Consider the following dataset, for K=3 and test data (X, 35,100) as (Person, Age, SalaryK) solve using Knn classifier model and predict the target

Person	Age	SalaryK	Target	Distance
A	18	50	N	$\sqrt{(35-18)^2+(100-50)^2} = 52.8$
B	23	55	N	$\sqrt{(35-23)^2+(100-55)^2} = 46.6$
C	24	70	N	$\sqrt{(35-24)^2+(100-70)^2} = 31.9$
D	41	60	Y	$\sqrt{(35-41)^2+(100-60)^2} = 40.4$
E	43	70	Y	$\sqrt{(35-43)^2+(100-70)^2} = 31.0$
F	38	40	Y	$\sqrt{(35-38)^2+(100-40)^2} = 60.1$
X	35	100	?	

K=3, The 3 closest entries are:

E (40,70,Y) - 31.0
C (24,70,N) - 31.9
D (41,60,Y) - 40.4

} Majority Target: Y

Conclusion: Using KNN with K=3, the predicted target for (35,100) is $\boxed{Y}$

- For Iris Dataset
  How to choose the K value? Demonstrate using accuracy rate.

Ans The value of K determines how many neighbours are used for prediction.
K can vary from 1 to 20 and we can observe:

Program 7

Build Support vector machine model for a given dataset

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

# Define the Linear SVM class
class LinearSVM:
    def __init__(self, learning_rate=0.001, reg_strength=0.1,
num_iterations=1000):
        self.learning_rate = learning_rate
        self.reg_strength = reg_strength
        self.num_iterations = num_iterations

    def fit(self, X, y):
        # Initialize weights and bias
        num_samples, num_features = X.shape
        self.W = np.zeros(num_features)  # Weights
        self.b = 0  # Bias

        # Gradient Descent
        for _ in range(self.num_iterations):
            # Compute the margin (decision function)
            margins = 1 - y * (np.dot(X, self.W) + self.b)
            # Compute gradient
            dw = -2 * np.dot(X.T, (y * (margins > 0))) / num_samples + 2 *
self.reg_strength * self.W
            db = -2 * np.sum(y * (margins > 0)) / num_samples

            # Update weights and bias
            self.W -= self.learning_rate * dw
            self.b -= self.learning_rate * db



    def predict(self, X):
        # Make predictions
        return np.sign(np.dot(X, self.W) + self.b)

# Generate toy data (binary classification)
np.random.seed(42)
num_samples = 100
X = np.random.randn(num_samples, 2)
y = np.ones(num_samples)
y[X[:, 0] < X[:, 1]] = -1  # Assign different class based on condition

# Train the Linear SVM
svm = LinearSVM(learning_rate=0.001, reg_strength=0.1, num_iterations=1000)
svm.fit(X, y)

# Predict
y_pred = svm.predict(X)
```

```python
# Visualize the decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100), np.linspace(ylim[0],
ylim[1], 100))
Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
plt.title("Linear SVM Decision Boundary")
plt.show()

# Print accuracy (simple comparison)
accuracy = np.mean(y_pred == y)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

LAB - 6

Date 21/4/25
Page 20

Support Vector Machine

BEGIN
LOAD dataset
PREPROCESS dataset:
- Handle missing value
- Encode categorical variables (if any)
- Scale/Normalise features

DEFINE SVM Model:
- Choose kernel type: linear, polynomial, Gaussian etc.
- Choose regularization parameter, C
- For non-linear data, select appropriate kernel and parameters

TRAIN SVM:
- Find the optimal hyperplane that maximise the margin b/w classes
- Obj: minimise $(1/2) \cdot \|w\|^2$

- If soft margin SVM:
Allow some slack using parameter C:
Minimise $(1/2) \cdot \|w\|^2 + C \cdot \Sigma \xi_i$

PREDICT class label
For each sample x: ~~compute~~
compute decision function:
$f(x) = w \cdot x + b$
Predict class: $y = +1$ if $f(x) \geq 0$ else -1

EVALUATE:
Accuracy = (No of correct predictions) / (Total predictions)

END

Program 8

Implement Random forest ensemble method on a given dataset.

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

# Load the iris dataset from CSV
df = pd.read_csv("/content/iris (2).csv")

# Assuming last column is the label
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Split into training and test sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# 1. Train RF Classifier with default n_estimators=10
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
accuracy_default = accuracy_score(y_test, y_pred_default)

print(f"Default RF Accuracy (n_estimators=10): {accuracy_default:.4f}")

# 2. Fine-tune: Try different numbers of trees (1 to 100)
best_accuracy = 0
best_n = 0
accuracies = []

for n in range(1, 101):
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

    if acc > best_accuracy:
        best_accuracy = acc
        best_n = n

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

# Plot accuracy vs. number of trees
plt.figure(figsize=(10, 6))
plt.plot(range(1, 101), accuracies, marker='o')
plt.title("Accuracy vs Number of Trees in Random Forest")
plt.xlabel("Number of Trees (n_estimators)")
plt.ylabel("Accuracy")
plt.grid(True)
```

```
plt.show()
```

LAB - 7

Date 5/5/25
Page 21

Random forest ensemble method

Pseudocode

BEGIN
LOAD dataset
PREPROCESS data:
- Handle missing value
- Encode categorical variable
- Split the data into training and testing set

DEFINE RANDOM forest:
- Choose number of trees (n-trees)
- choose max-depth, min samples split etc.

FOR each tree it from 1 to n-trees:
- Draw a bootstrap sample from training data
- Train decision tree (e.g. using ID3) on this sample
- For each node, use random subset of features to find best split

FOR each ~~test~~ sample:
- Predict class using each tree
- Final prediction = majority vote from all trees

EVALUATE:
Accuracy = Correct predictions / Total Prediction

END

Program 9

Implement Boosting ensemble method on a given dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

# Step 1: Load the dataset
df = pd.read_csv("/content/income.csv")

# Step 2: Split into features and target
X = df.drop(columns=['income_level'])
y = df['income_level']

# Step 3: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Step 4: AdaBoost with 10 estimators
model_10 = AdaBoostClassifier(n_estimators=10, random_state=42)
model_10.fit(X_train, y_train)
y_pred_10 = model_10.predict(X_test)
accuracy_10 = accuracy_score(y_test, y_pred_10)
conf_matrix_10 = confusion_matrix(y_test, y_pred_10)

print("Accuracy with 10 estimators:", round(accuracy_10, 4))
print("Confusion Matrix (10 estimators):\n", conf_matrix_10)

# Step 5: Fine-tune number of trees (1 to 50)
best_accuracy = 0
best_n = 0
accuracies = []

for n in range(1, 51):
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

    if acc > best_accuracy:
        best_accuracy = acc
        best_n = n

print(f"\nBest Accuracy: {round(best_accuracy, 4)} with n_estimators =
{best_n}")

# Step 6: Plot accuracy vs. number of estimators
plt.figure(figsize=(10, 6))
plt.plot(range(1, 51), accuracies, marker='o', linestyle='-', color='blue')
plt.title('Accuracy vs Number of Trees (n_estimators)')
```

```python
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.grid(True)
plt.tight_layout()
plt.show()
```

LAB - 8

Date 5/5/25
Page 22

Boosting Ensemble Method (e.g. AdaBoost)

BEGIN
LOAD dataset
PREPROCESS dataset :
- Handle missing values
- encode categorical variables
- split the dataset into training and testing

INITIALISE :
- Initialise ~~the~~ equal weights to all the training ~~data~~ samples

FOR $t = 1$ to n-estimators :
- Train weak learners (decision stump) using weighted data
- Compute error

$$\varepsilon_i = \sum_{j=1}^{N} H_i(d_j) \times wt(d_j)$$

where $H_i(d_j) = 0$ for correct pred
$= 1$ - wrong -

- Compute wt of each weak classifier

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

- Calculate normalising factor

$Z(e_i)$ = wt (correct classified instance) x
feature : No of correct classifier x $e^{-\alpha_{ei}}$
+ wt (wrong classified instance) x
No of wrong classifier x $e^{\alpha_{ei}}$

- update the wt of data instances :

$$wt(d_j)_{i+1} = \frac{wt(d_j) \text{ of correct instance} \times e^{-\alpha_{ei}}}{Z_{ei}}$$

Date __/__/__
Page 23

$$wt(d_j)_{i+1} = \frac{wt(d_j) \text{ wrong instance} \times e^{\alpha_{ei}}}{Z_{ei}}$$

COMPUTE final predicted value

$$\sum_{i=1}^{m} \alpha_i \times H_i(d_j)$$

wt of all features

Yes = 1
No = 0

END

Program 10

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

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

# Load the dataset
df = pd.read_csv("/content/iris (2).csv")

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

# Optional: Standardize the data
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:
    model = KMeans(n_clusters=k, random_state=42, n_init=10)
    model.fit(X_scaled)
    inertia.append(model.inertia_)

# Plot the elbow graph
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.title('Elbow Plot for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()
```

Date 12/5/25
Page 24

LAB-9

K-Means Clustering

Pseudocode

BEGIN
  load dataset
  PREPROCESS data :
    - Handle missing values.
    - normalise features (important for dist)

  INPUT: No of clusters (K)

  INTIALISE :
    - Randomly select K points as intial centroids

  REPEAT until Convergence :
    FOR each data point x_i :
      Assign x_i to the nearest centroid
      Euclidean Distance = sqrt $\left(\Sigma(x_i - \mu_i)^2\right)$

    FOR each cluster :
      Update centroid = mean of all assigned points

  OUTPUT: cluster assignments for each point, final centroids

END

Program 11

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

```
import pandas as pd
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.decomposition import PCA
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv("/content/heart (1).csv")  # Update to match your file path
if needed

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

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

# Encode categorical columns
for col in categorical_cols:
    if X[col].nunique() == 2:
        X[col] = LabelEncoder().fit_transform(X[col])
    else:
        X = pd.get_dummies(X, columns=[col])

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

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

# Train and evaluate models (without PCA)
print("□ Accuracy without PCA:")
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"{name}: {accuracy_score(y_test, y_pred):.4f}")

# Apply PCA (reduce to 5 components)
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)
```

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

# Train and evaluate models (with PCA)
print("\n□ Accuracy with PCA:")
for name, model in models.items():
    model.fit(X_train_pca, y_train_pca)
    y_pred_pca = model.predict(X_test_pca)
    print(f"{name}: {accuracy_score(y_test_pca, y_pred_pca):.4f}")
```

LAB-10

Date 12/5/25
Page 25

Principal Component Analysis (PCA)

Pseudocode

BEGIN
LOAD dataset
PREPROCESS data:
- Handle missing values
- Normalise features (zero mean, unit variance)

Compute mean $\bar{X}_1$, $\bar{X}_2$
COMPUTE covariance matrix:

$$S = \begin{bmatrix} cov(X_1,X_1) & cov(X_1,X_2) \\ cov(X_2,X_1) & cov(X_2,X_2) \end{bmatrix} \quad COV = (1/n) \times (X^T \cdot X)$$

COMPUTE eigen values and eigen vectors of covariance matrix

SELECT top k eigen vectors with largest eigen values

PROJECT data to lower dimensions:
X_reduced = X · W
where W is matrix of selected eigen vectors

OUTPUT:
- Reduced dataset (X_reduced)

END.

19-5-2025