VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Abhishek S Halagadagi (1BM22CS008)

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 **Abhishek Shivanand Halagadagi (1BM22CS008),** 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/Abhi-008-sh/MLGLab

Program 1

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

Screenshot

Date : 4/3/2025
Page No.:

Lab 1

```
import pandas as pd

data = {
    'Name' = ['Alice', 'Bob', 'Charlie'],
    'Age' = [25, 30, 35]
    'city' = ['New York', 'Los Angles', 'Chicago']
}

df = Pd.Dataframe(data)
df.head()
```

Output:

Name	Age	city
Alice	25	New York.

→ importing dataset from sklearn dataset

```
from sklearn.dataset import load_iris.
iris = load_iris()
df = Pd.dataframe(iris.data, columns=
        iris.feature_name)

df('target') = iris_target
print("sample data:")
df.head()
```

output:
sample data:

	Sepal length	sepal width	pet length	Petwidth
0	5.1	3.5	1.4	0.2

Date :
Page No.:

→ importing data from specified csv file.

```
file path = 'data.csv'
df = pd.read_csv(file_path)
print("scample data")
```

→ Document dataset

```
df = pd.read_csv('mobile_dataset-2025.csv')
print("Sample data")
df.head()
```

Sample data:

comp_name	model_name	weight	RAM
apple	iphone 16	174g	6GB

front_cam	Back cam	processor	Battery cap
12 MP	48 MP	A17 Bionic	3600mAh

size	launced_prize (India)	launched year
6.1 inches	79999	2024.

→ Exporting data.

1) df = Pd.read_csv('sample_sales_data.csv')

2) df.to_csv('output.csv', index = false)
   Print("Data saved to output.csv").

→ analysis of sales dataset

1) sales_df = Pd.read_csv('sales_data.csv')

Date :
Page No. :

2. Summerize sales by region.

```
sales_by_re = sales_df.groupby('Region').('Sale
   .sum()
```

```
Sales_by_re.
Region
 east    770
North   16400
South   3070.
 west    650.
```

3. Grouping by product & calculating total quantity sold

```
best selling_pre = sales_df.groupby('Produc
   ['Quantity']).sum().sort_values(ascendir
   = false)
```

```
best_selling_pre
Product
 mouse     24.
 laptop    17
 keyboard  16
 monitor   15
```

④ Saving data to a csv file

```
sales_by_re.to_csv('Sales_by_re.csv')
best_selling_re.to_csv('best_sell_re.csv')
print("\n Results saved to csv file")
```

results saved to csv file.

Code:

```
import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

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

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

print("First 5 rows of the dataset:")

print(data.head())

print("\nShape of the dataset:")

print(data.shape)

print("\nColumn names:")

print(data.columns)

hdfc_data = data['HDFCBANK.NS']

print("\nSummary statistics for HDFC Bank:")

print(hdfc_data.describe())

hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()

icici_data = data['ICICIBANK.NS']

print("\nSummary statistics for ICICI Bank:")

print(icici_data.describe())

icici_data['Daily Return'] = icici_data['Close'].pct_change()

kotak_data = data['KOTAKBANK.NS']

print("\nSummary statistics for Kotak Mahindra Bank:")

print(kotak_data.describe())
```

```
kotak_data['Daily Return'] = kotak_data['Close'].pct_change()
plt.figure(figsize=(14, 10))
plt.subplot(3, 2, 1)
hdfc_data['Close'].plot(title="HDFC Bank - Closing Price")
plt.subplot(3, 2, 2)
hdfc_data['Daily Return'].plot(title="HDFC Bank - Daily Returns", color='orange')
plt.subplot(3, 2, 3)
icici_data['Close'].plot(title="ICICI Bank - Closing Price")
plt.subplot(3, 2, 4)
icici_data['Daily Return'].plot(title="ICICI Bank - Daily Returns", color='orange')
plt.subplot(3, 2, 5)
kotak_data['Close'].plot(title="Kotak Mahindra Bank - Closing Price")
plt.subplot(3, 2, 6)
kotak_data['Daily Return'].plot(title="Kotak Mahindra Bank - Daily Returns", color='orange')
plt.tight_layout()
plt.show()


hdfc_data.to_csv('hdfc_bank_data.csv')
icici_data.to_csv('icici_bank_data.csv')
kotak_data.to_csv('kotak_bank_data.csv')


print("\nHDFC Bank data saved to 'hdfc_bank_data.csv'.")
print("ICICI Bank data saved to 'icici_bank_data.csv'.")
print("Kotak Bank data saved to 'kotak_bank_data.csv'.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot

Lab 2

Demonstrate various data pre-processing techniques

Housing.csv

```
import pandas as pd

i) df = pd.read_csv ('/content/housing.csv")

ii) Print ("Information of all coloums:" ).
    Print (df.head())

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

print ("\n Count of unique labels for 'Ocean
        Proximity' column")
print (df['ocean_proximity'].value_counts())

print ("\n Columns with missing values:").
missing_values = df.isnull().sum()
columns_with_missing_values = missing_values
                    [missing_values>0]
Print (colums_with_missing_values)
```

Output:

longitude	latitude	housing_median	total_room
-122.23	37.88	41.0	880.0

total_bedrooms	Population	households	median_income
129	322	126.0	8.3252

median_house_value	ocean_proximity
452600.0	Near Bay.

	longitude	latitude	housing median age
count	20640	20640	20640
mean	-119.569704	35.631861	28.639486
std	2.003532	2.135952	12.585558
min	-124.35	32.54	1.0
25%	-121.8	33.93	18
50%	-118.49	34.26	29
75%	-118.01	37.71	37
max	-114.31	41.95	52

count of unique labels for 'Ocean Proximate' column.

<1H Ocean	9136
INLAND	6551
NEAR OCEAN	2658
NEAR BAA	2290
ISLAND	5

columns with missing values.

total_bedrooms 207

Diabetes and Adult dataset

```
import pandas as pd
import numpy as np
from sklearn preprocessing input minMaxScaler, Standard
-Sc
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn.impute
file.path = "diabetis.csv"
df = pd.read_csv (file-Path)
df-num = df.select_dtypes (include=['number'])
        copy()
imputer = simpleimputer (startegy='mean')
df-num.iloc (:, :) = imputer.fit_transform(df_numeric
df (df.num.colums) = df.numeric
Q1 = df.num.quantile (0.25)
Q3 = df num quantile (0.75)
IQR = Q3 - Q1
df = df [~ ((df-numeric < (Q1 -1.5 * IQR)) |
        (df-numeric > (Q3+1.5 * IQR))).aw(axis=1)
min-max-scaler = min maxscaler()
df minmax - pd.DataFrame (min_max-scaler.fit
        transform (df_num columns = df-num.columns)
standard-scaler = standardscaler()
df_standard = pd.DataFrame(standard.scaler.f
        _transform (df n um) columns=df mndumn
print("In proussed dataset (MinMax Scaled): "
Print ( df-minmax.head())
print ("In Proposed Dataset (standard scaled):")
print ( df_standard .head())
```

(a) Missing values are present in numerical columns if present which are replaced by the mean of the respective columns

(b) No categorical column, no encoding

c) Min max scaling transform data to is fixed range (0,1] using.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

used when: dataset does not follow a normal dist.
- features have diff ranges & used to be bound, standardization transform data have zero means & unit variance

$$x' = \frac{x - \mu}{\sigma}$$

used when: dataset follows a Gaussian dist many ML Algos assume normality

Adult dataset.

1) which columns have missing values?
workclass, occupation, native country
Handling missing values by

-> fill with mode for categorical data
-> drop rows if missing percentage is too high

2) Identify and encoding categorical columns
columns are: workclass, education, marital status, occupation, relationship, race encoding startegy.

-> One-hot encoding: for nominal categorical column
-> ordinal encoding: categorical column has a meaningful order.

3) Min Max & standardization (Difference Min Max)
Min Max: Formula:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- Features have known range.
- dataset does not have normal distribution.

standardization: Formula:

$$x' = \frac{x - \mu}{\sigma}$$

* features have unit scale.

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt

diabetes_data = pd.read_csv('/content/Dataset of Diabetes .csv')
adult_income_data = pd.read_csv('/content/adult.csv')

print("Diabetes Dataset:")
print(diabetes_data.head())

print("\nAdult Income Dataset:")
print(adult_income_data.head())

diabetes_numerical_cols = diabetes_data.select_dtypes(include=[np.number]).columns
diabetes_categorical_cols = diabetes_data.select_dtypes(include=[object]).columns

diabetes_imputer_num = SimpleImputer(strategy='median')
diabetes_data[diabetes_numerical_cols] =
diabetes_imputer_num.fit_transform(diabetes_data[diabetes_numerical_cols])

diabetes_imputer_cat = SimpleImputer(strategy='most_frequent')
diabetes_data[diabetes_categorical_cols] =
diabetes_imputer_cat.fit_transform(diabetes_data[diabetes_categorical_cols])

adult_income_numerical_cols = adult_income_data.select_dtypes(include=[np.number]).columns
adult_income_categorical_cols = adult_income_data.select_dtypes(include=[object]).columns

adult_income_imputer_num = SimpleImputer(strategy='median')
adult_income_data[adult_income_numerical_cols] =
adult_income_imputer_num.fit_transform(adult_income_data[adult_income_numerical_cols])

adult_income_imputer_cat = SimpleImputer(strategy='most_frequent')
adult_income_data[adult_income_categorical_cols] =
adult_income_imputer_cat.fit_transform(adult_income_data[adult_income_categorical_cols])

categorical_columns_adult = adult_income_data.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()

for col in categorical_columns_adult:
   adult_income_data[col] = label_encoder.fit_transform(adult_income_data[col])
```

```
def detect_and_remove_outliers(df):
    numerical_df = df.select_dtypes(include=[np.number])
    Q1 = numerical_df.quantile(0.25)
    Q3 = numerical_df.quantile(0.75)
    IQR = Q3 - Q1
    return df[~((numerical_df < (Q1 - 1.5 * IQR)) | (numerical_df > (Q3 + 1.5 * IQR))).any(axis=1)]

diabetes_data_cleaned = detect_and_remove_outliers(diabetes_data)
adult_income_data_cleaned = detect_and_remove_outliers(adult_income_data)

min_max_scaler = MinMaxScaler()

diabetes_numerical_cols = diabetes_data_cleaned.select_dtypes(include=[np.number]).columns
diabetes_data_normalized = diabetes_data_cleaned.copy()

diabetes_data_normalized[diabetes_numerical_cols] =
min_max_scaler.fit_transform(diabetes_data_cleaned[diabetes_numerical_cols])

adult_income_numerical_cols =
adult_income_data_cleaned.select_dtypes(include=[np.number]).columns
adult_income_data_normalized = adult_income_data_cleaned.copy()

adult_income_data_normalized[adult_income_numerical_cols] =
min_max_scaler.fit_transform(adult_income_data_cleaned[adult_income_numerical_cols])

standard_scaler = StandardScaler()

diabetes_data_standardized = diabetes_data_cleaned.copy()
diabetes_data_standardized[diabetes_numerical_cols] =
standard_scaler.fit_transform(diabetes_data_cleaned[diabetes_numerical_cols])

adult_income_data_standardized = adult_income_data_cleaned.copy()
adult_income_data_standardized[adult_income_numerical_cols] =
standard_scaler.fit_transform(adult_income_data_cleaned[adult_income_numerical_cols])
```

Program 3

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

Screenshot:

Lab - 5

instance	A_2	A_3	classification
1	Hot	high	No
2	Hot	high	No
6	cool	high	No
7	Hot	high	No
8	Hot	Normal	Yes

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

$$= 0.7219$$

for a_2,

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

$$= 0.8113$$

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

$$\text{Gain}\ (S, a_2) = 0.7219 - \frac{4}{5} \times 0.8113 - 0 = 0.07266$$

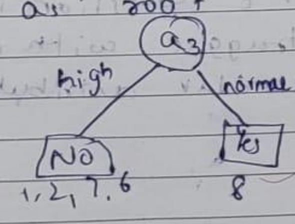
For a_3,

$$S_{high}\ (0+, 4-) = 0$$

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

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

$\therefore$ a_3 has highest gain value it is taken as root



1) what was the accuracy score for the iris dataset.
1 (100%)

$$\begin{bmatrix} 19 & 0 & 0 \\ 0 & 13 & 0 \\ 0 & 0 & 13 \end{bmatrix}$$

There are No miss classifications occurred.

2) The regression tree splits the data based on the most impactful feature influencing petrol consumption. Each split represents a decision rule on one of the input features. at the end of the tree, the predicted petrol consumption values are given.

Usually population Driver & Average income tend to be among the most important feature.

Unlike classification tree regression trees output assign based on numerical values. They minimize mean square error at each split.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
mean_absolute_error, mean_squared_error
from sklearn.preprocessing import LabelEncoder

iris = pd.read_csv("/content/iris (4).csv")
drug = pd.read_csv("/content/drug.csv")
petrol = pd.read_csv("/content/petrol_consumption.csv")

X_iris = iris.iloc[:, :-1]
y_iris = iris.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

print("Decision Tree Classification for IRIS Dataset:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
print("Classification Report:\n", classification_report(y_test, y_pred))

X_drug = drug.iloc[:, :-1]
y_drug = drug.iloc[:, -1]

le = LabelEncoder()

for col in X_drug.select_dtypes(include=['object']).columns:
    X_drug[col] = le.fit_transform(X_drug[col])

X_train, X_test, y_train, y_test = train_test_split(X_drug, y_drug, test_size=0.2, random_state=42)

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

print("\nDecision Tree Classification for Drug Dataset:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

X_petrol = petrol.iloc[:, :-1]
y_petrol = petrol.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X_petrol, y_petrol, test_size=0.2, random_state=42)

dtr = DecisionTreeRegressor()
dtr.fit(X_train, y_train)
y_pred = dtr.predict(X_test)

print("\nDecision Tree Regression for Petrol Consumption:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot

Lab 4.

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

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

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

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

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

$$(X^T X)^{-1} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$$

$$((X^T X)^{-1} X^T) = \begin{pmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{pmatrix}$$

Date :
Page No.:

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

$$\beta = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix} \begin{array}{l} \text{intercept} \\ \text{slope} \end{array}$$

$y = \beta_0 + \beta_1 x + \epsilon$

$y = -0.5 + 2.2x$

At $x = 5$

$y = -0.5 + 2.2(5)$

$y = 10.5$

11-3-25

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

hiring_data = pd.read_csv('hiring.csv')
print(hiring_data.head())
hiring_data = hiring_data.dropna()

experience_mapping = {
    'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'seven': 7, 'eight': 8,
    'nine': 9, 'ten': 10, 'eleven': 11, 'twelve': 12, 'thirteen': 13, 'fourteen': 14,
}

hiring_data['experience'] = hiring_data['experience'].replace(experience_mapping)
hiring_data['experience'] = pd.to_numeric(hiring_data['experience'], errors='coerce')

if hiring_data['experience'].isnull().any():
    print("Warning: There are still non-numeric values in the 'experience' column.")
    hiring_data = hiring_data.dropna(subset=['experience'])

X_hiring = hiring_data[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y_hiring = hiring_data['salary($)']

X_train_hiring, X_test_hiring, y_train_hiring, y_test_hiring = train_test_split(X_hiring, y_hiring,
test_size=0.2, random_state=42)

regressor_hiring = LinearRegression()
regressor_hiring.fit(X_train_hiring, y_train_hiring)

candidate_1 = np.array([[2, 9, 6]])
candidate_2 = np.array([[12, 10, 10]])

salary_1 = regressor_hiring.predict(candidate_1)
salary_2 = regressor_hiring.predict(candidate_2)

print(f"Predicted salary for candidate 1 (2 yr experience, 9 test score, 6 interview score):
{salary_1[0]}")
print(f"Predicted salary for candidate 2 (12 yr experience, 10 test score, 10 interview score):
{salary_2[0]}")
```

```
companies_data = pd.read_csv('/content/1000_Companies.csv')
print(companies_data.head())
companies_data = companies_data.dropna()

label_encoder = LabelEncoder()
companies_data['State'] = label_encoder.fit_transform(companies_data['State'])

X_companies = companies_data[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]
y_companies = companies_data['Profit']

X_train_companies, X_test_companies, y_train_companies, y_test_companies =
train_test_split(X_companies, y_companies, test_size=0.2, random_state=42)

regressor_companies = LinearRegression()
regressor_companies.fit(X_train_companies, y_train_companies)

input_data = np.array([[91694.48, 515841.3, 11931.24, label_encoder.transform(['Florida'])[0]]])
predicted_profit = regressor_companies.predict(input_data)

print(f"Predicted profit for the given inputs (Florida State): {predicted_profit[0]}")

y_pred_hiring = regressor_hiring.predict(X_test_hiring)
mae_hiring = mean_absolute_error(y_test_hiring, y_pred_hiring)
print(f"Mean Absolute Error for Salary Prediction: {mae_hiring}")

y_pred_companies = regressor_companies.predict(X_test_companies)
mae_companies = mean_absolute_error(y_test_companies, y_pred_companies)
print(f"Mean Absolute Error for Profit Prediction: {mae_companies}")
```

Program 5

Build Logistic Regression Model for a given dataset

Screenshot

Lab - 3

Date: 18/03/2025
Page No.:

(1) $a_0 = -5$

$a_1 = 0.8$

i) $f(x) = \dfrac{1}{1 + \exp(-(-5 + 0.8x))}$

ii) $f(7) = \dfrac{1}{1 + \exp(-(-5 + 0.8 \times 7))}$

$= 0.6457$

iii) if $f(x) < 0.5$ then the student is ~~pass~~ fail

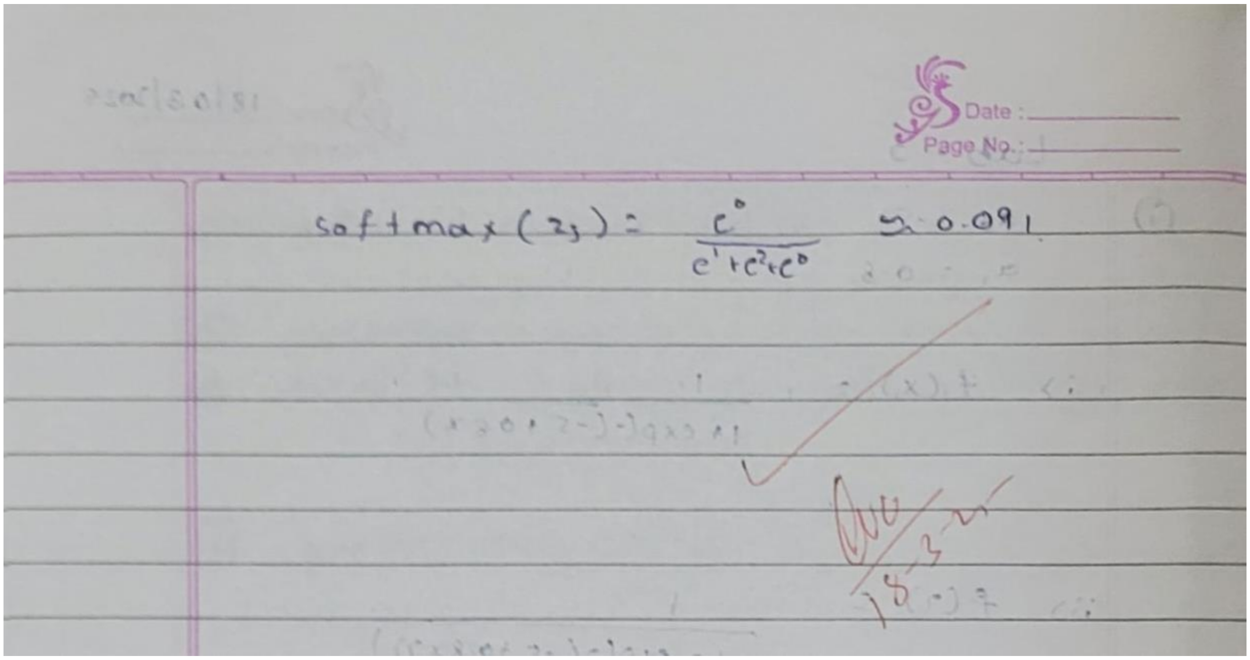
$f(x) \geq 0.5$ then the student is pass.

(2) $z = [2, 1, 0]$

$$\text{softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}$$

$$\text{softmax}(z_1) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}} = \frac{e^2}{e^2 + e^1 + e^0} \approx 0.6652$$

$$\text{softmax}(z_2) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}} \approx \frac{e^1}{e^2 + e^1 + e^0} \approx 0.245$$



Date :

Page No. :

$$\text{softmax}(z_3) = \frac{e^0}{e^1 + e^2 + e^0} \approx 0.091$$

Code:

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

file_path = 'HR_comma_sep.csv'
data = pd.read_csv(file_path)

print(data.info())

print(data.head())

print(data.describe())

plt.figure(figsize=(8, 5))
sns.countplot(x='salary', hue='left', data=data)
plt.title('Impact of Salary on Employee Retention')
plt.xlabel('Salary')
plt.ylabel('Count')
plt.legend(title='Employee Retention', labels=['Stayed', 'Left'])
plt.show()

plt.figure(figsize=(10, 6))
```

```
sns.countplot(x='Department', hue='left', data=data)
plt.title('Impact of Department on Employee Retention')
plt.xlabel('Department')
plt.ylabel('Count')
plt.legend(title='Employee Retention', labels=['Stayed', 'Left'])
plt.xticks(rotation=45)
plt.show()

data_encoded = pd.get_dummies(data, columns=['salary', 'Department'], drop_first=True)

print(data_encoded.info())

X = data_encoded.drop('left', axis=1)
y = data_encoded['left']

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

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

logreg = LogisticRegression(max_iter=1000)

logreg.fit(X_train_scaled, y_train)

y_pred = logreg.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Logistic Regression Model: {accuracy * 100:.2f}%")

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=['Stayed', 'Left'],
yticklabels=['Stayed', 'Left'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot

Lab 6.

Person	Age	Salary lk	Target
A	18	50	N
B	23	55	N
C	24	70	N
D	41	60	Y
E	43	70	Y
F	38	40	Y
x	35	100	?

$(x_2, y_2) = (35, 100)$

for $(18, 50)$ $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

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

$= 52.81$

for $(23, 55)$ $= \sqrt{(35 - 23)^2 + (100 - 55)^2}$

$= 46.57$

for $(24, 70)$ $= \sqrt{(35 - 24)^2 + (100 - 70)^2}$

$= 31.95$

for $(41, 60)$ $= \sqrt{(35 - 41)^2 + (100 - 60)^2}$

$= 40.44$

for $(43, 70)$ $= \sqrt{(35 - 43)^2 + (100 - 70)^2}$

$= 31.04$

for $(38, 40)$ $= \sqrt{(35 - 38)^2 + (100 - 40)^2}$

$= 60.07$

Person	Age	Salary lL	Target	distance	Rank
A	18	50	N	52.81	5
B	23	55	N	46.57	4
C	24	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	6.
X	35	100	Y		

k = 3, 2 - Y 1 - N.

k = 1 - Y

k = 2 - N

k = 3 - Y.

So, X (35, 100) is Y.

Code:

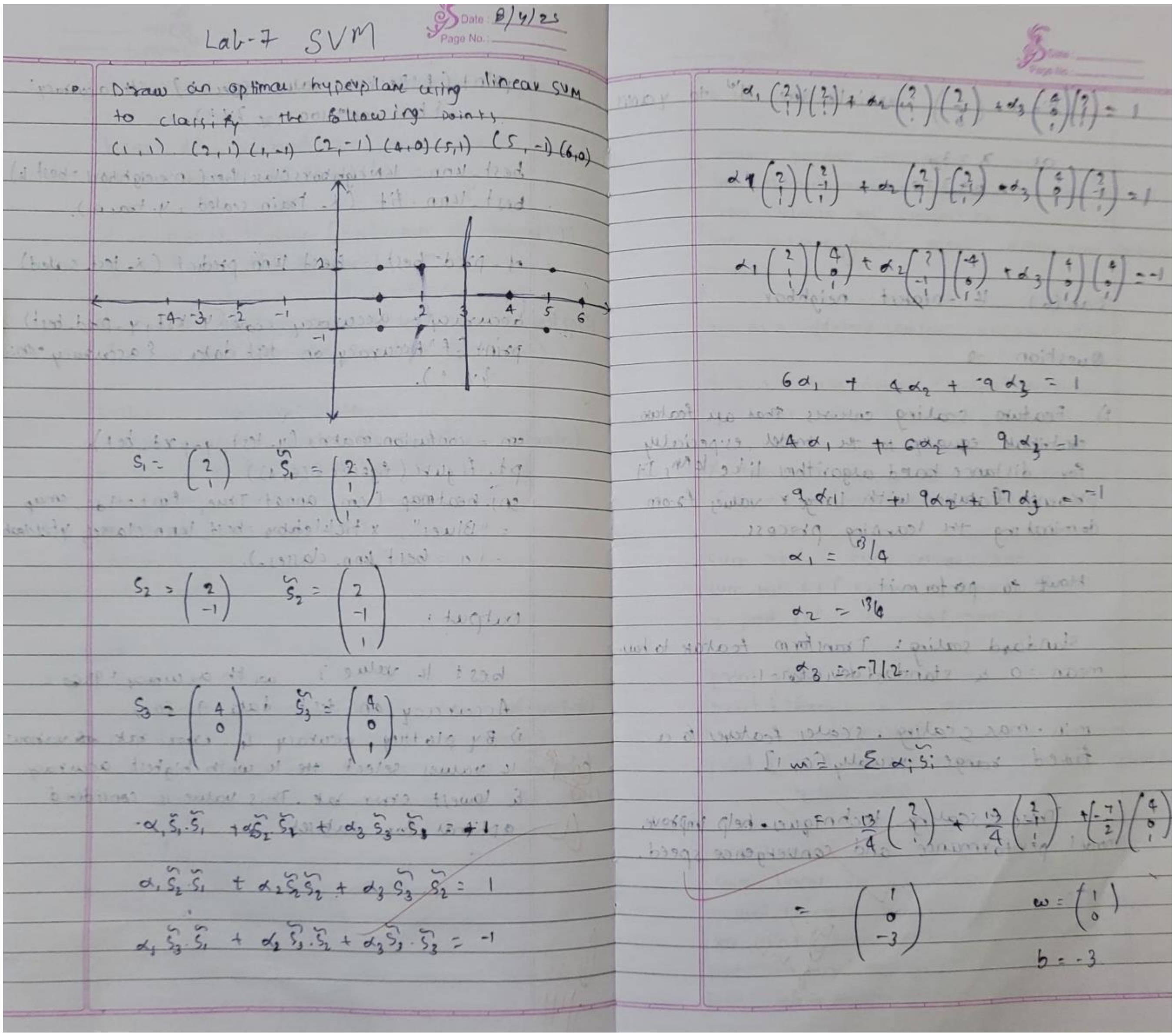
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot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

iris_df = pd.read_csv('/content/iris (3).csv')

print(iris_df.head())

X_iris = iris_df.drop(columns=['species'])
y_iris = iris_df['species']

X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train_iris = scaler.fit_transform(X_train_iris)
X_test_iris = scaler.transform(X_test_iris)

knn_iris = KNeighborsClassifier(n_neighbors=3)

knn_iris.fit(X_train_iris, y_train_iris)

y_pred_iris = knn_iris.predict(X_test_iris)

accuracy_iris = accuracy_score(y_test_iris, y_pred_iris)
print(f"Accuracy on Iris test data: {accuracy_iris * 100:.2f}%")

cm_iris = confusion_matrix(y_test_iris, y_pred_iris)
sns.heatmap(cm_iris, annot=True, fmt="d", cmap="Blues", xticklabels=knn_iris.classes_,
yticklabels=knn_iris.classes_)
plt.title("Confusion Matrix for Iris Dataset")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print("Classification Report for Iris Dataset:")
print(classification_report(y_test_iris, y_pred_iris))

diabetes_df = pd.read_csv('diabetes.csv')
print(diabetes_df.head())
```

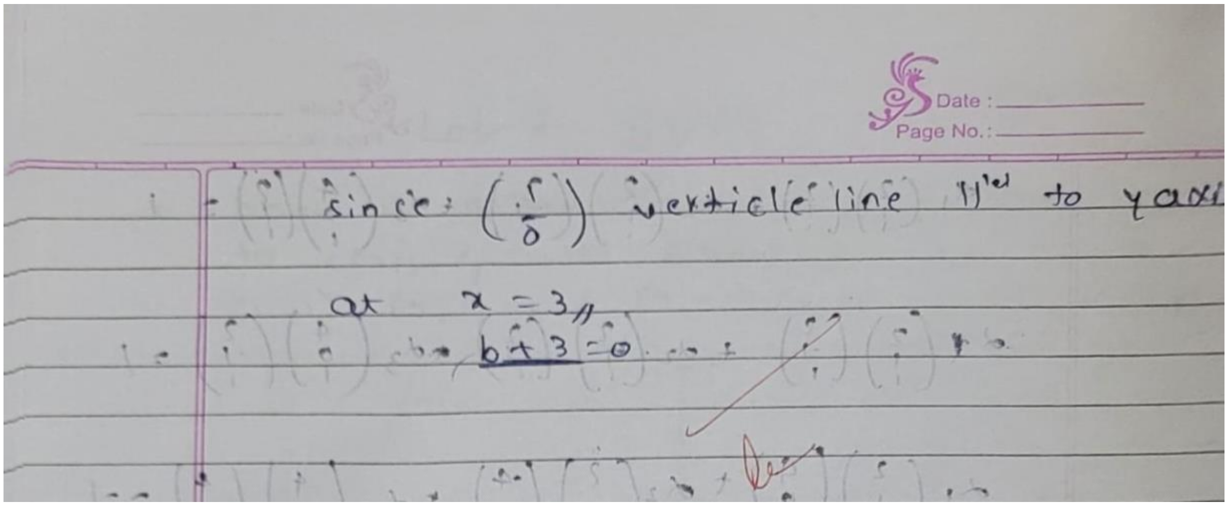
```
X_diabetes = diabetes_df.drop(columns=['Outcome'])
y_diabetes = diabetes_df['Outcome']

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

scaler = StandardScaler()
X_train_diabetes = scaler.fit_transform(X_train_diabetes)
X_test_diabetes = scaler.transform(X_test_diabetes)

knn_diabetes = KNeighborsClassifier(n_neighbors=5)

knn_diabetes.fit(X_train_diabetes, y_train_diabetes)

y_pred_diabetes = knn_diabetes.predict(X_test_diabetes)

accuracy_diabetes = accuracy_score(y_test_diabetes, y_pred_diabetes)
print(f"Accuracy on Diabetes test data: {accuracy_diabetes * 100:.2f}%")

cm_diabetes = confusion_matrix(y_test_diabetes, y_pred_diabetes)
sns.heatmap(cm_diabetes, annot=True, fmt="d", cmap="Blues", xticklabels=knn_diabetes.classes_,
yticklabels=knn_diabetes.classes_)
plt.title("Confusion Matrix for Diabetes Dataset")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print("Classification Report for Diabetes Dataset:")
print(classification_report(y_test_diabetes, y_pred_diabetes))
```

Program 7

Build Support vector machine model for a given dataset

Screenshot





since $\left(\frac{r}{\delta}\right)$ verticle line $\parallel^{el}$ to y axi

at $x = 3$

$b+3=0$

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve
from sklearn.preprocessing import LabelEncoder, label_binarize
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

df = pd.read_csv("/content/letter-recognition.csv")

top_classes = df['letter'].value_counts().head(5).index.tolist()
df = df[df['letter'].isin(top_classes)]

X = df.iloc[:, 1:]
y = df.iloc[:, 0]

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

y_bin = label_binarize(y_encoded, classes=np.unique(y_encoded))
n_classes = y_bin.shape[1]

X_train, X_test, y_train, y_test_bin = train_test_split(X, y_bin, test_size=0.2, random_state=42)

svm_model = SVC(kernel='linear', probability=True)
svm_model.fit(X_train, y_train.argmax(axis=1))
y_score = svm_model.predict_proba(X_test)

y_pred = svm_model.predict(X_test)
```

```
y_true = y_test_bin.argmax(axis=1)

print("Accuracy:", accuracy_score(y_true, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))

plt.figure()
for i in range(n_classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    auc = roc_auc_score(y_test_bin[:, i], y_score[:, i])
    plt.plot(fpr, tpr, label=f"{label_encoder.inverse_transform([i])[0]} AUC={auc:.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve (Top 5 Classes)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()

macro_auc = roc_auc_score(y_test_bin, y_score, average="macro")
print("Macro AUC Score:", macro_auc)
```

Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot

Date : 15/04/2025
Page No.:

Lab-8 (Random Forest)

1) Decision Tree | Random Forest
- single tree. | Multiple.
- less accurate. | More accurate
- Fast to train | slow to train
- single prediction | majority rate.

2) Parameters of random forest classification
-> n-estimate : no. of trees in forest
-> critision : to nearest quality of split
-> max-depth : min depth of tree
-> min-sample-split : min sample req to split rate.
-> Max feature : no. of features to look for best fit.

3) Algo

1) training dataset.

2) for n time.
 - randomly select sample with replacement.
 - Grow a decision tree.
 - at each split choose random subset of feature.
 - split nodes using best features

3) Aggregate prediction.
 - classification - majority vote.
 - regression - avg.

4) O/P prediction.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn import preprocessing

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

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

for column in X.columns:
    if X[column].dtype == 'object':
      le = preprocessing.LabelEncoder()
      X[column] = le.fit_transform(X[column])

if y.dtype == 'object':
  le = preprocessing.LabelEncoder()
  y = le.fit_transform(y)

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

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

y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot

Page No.:

Lab-9 (Ada boosting)

Boosting:

combines multiple weak learners to create a strong learners. It works by training model sequentially where each model focuses on error model by previous one.

Parameters:

estimator	The base model.
n-estimator	no. of weak learners
learning rate	shrinks contributers of each learner
algorithm	'SAMME' or SAMME.R'.
random.state	for reproducability.

Algorithm

1) start with equal wts for all training sample
2) train a weak model.
3) cal errors & update sample wts
4) add weak model to ensemble with a wt based on its accuracy
5) Repeat for n estimators
6) final prediction.

Code:

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

iris = load_iris()
X = iris.data
y = iris.target

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

results = []

n_estimators_list = [10, 50, 100]
learning_rates = [0.01, 0.1, 1]

for n in n_estimators_list:
    for lr in learning_rates:
        tree_base = DecisionTreeClassifier(max_depth=1)
        model = AdaBoostClassifier(estimator=tree_base, n_estimators=n, learning_rate=lr,
random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Base': 'DecisionTree',
            'n_estimators': n,
            'learning_rate': lr,
            'Accuracy': acc
        })

for n in n_estimators_list:
    for lr in learning_rates:
        log_reg_base = LogisticRegression(max_iter=1000)
        model = AdaBoostClassifier(estimator=log_reg_base, n_estimators=n, learning_rate=lr,
random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Base': 'LogisticRegression',
```

```
        'n_estimators': n,
        'learning_rate': lr,
        'Accuracy': acc
    })

results_df = pd.DataFrame(results)
print(results_df)

import seaborn as sns
plt.figure(figsize=(12, 6))
sns.barplot(x='n_estimators', y='Accuracy', hue='Base', data=results_df, ci=None)
plt.title('AdaBoost Accuracy with Different Estimators and n_estimators')
plt.show()
```

Program 10

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

Screenshot

Date: 18/31 9 (40)

```
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_train)
y_pred = kmeans.predict(X_test)
print(f'Predicted clusters for test Data:{y_pred}')
```

output:

Predicted clusters for Test Data:-

[1 0 1 0 2 0 1 1 0 0]

Q1) Algorithm.

1) select number k to decide the no. of clusters.
2) select random k points or centroids.
3) Assign each data point to their closet centroid, which will form the predefined k cluster.
4) Calculate the variance & place a new centroid of each cluster.
5) Repeat the third steps, which means reassign each datapoint to new closest centroid of each cluster.
6) If any reassignment occure then go to step-4 else go to Finish.
7) The model is ready.

Q2) How to determine no. of clusters..

Elbow method

Silhoutte score

Gaps statistic

Q2) formulas for sum of squared error plot SSE vs no. of cluster,

$$SSE = \sum_{k=1}^{K} \sum_{i \in k} || x_i - \mu_k ||^2$$

The plot of SSE vs no. of cluster

Step 1: Run the k-means algorithm for a range of cluster values.

2) calculate SSE for each no. of clusters
3) Plot SSE vs no. of clusters.
4) calculate the variance & place a new centroid of each cluster.
5) Repet the step 3, which means reasisn each datapoin to the new closet centroid of each clusters.
6) if any reassignment occurs then go to Finish.
7) The model is ready.

Q) Elbow Technique.

i) Run k means for range of values of k.
ii) compute the within clusters sum of squares each value of k.
iii) Plot the WCSS against the no. of cluster.
iv) Look for the elbow point the value of k at which the rate of decrease in was slower significantly.

Q) Discuss all parameters used in kmeans():

i) n-cluster: No of cluster to form.

ii) init = k means + random an array of shape.

iii) n-init: No of times the k-means algorithm will be run with different centroid seeds.

iv) max-iter: Max no of iterations of the k-means ~~are~~

v) tol: Relative tolerance with regards to inertia to declare convergence.

vi) Random state: controls the regards to inertia to declare convergence.

vii) Algorithm: k-means implementation

Q) cluster 8 points (with (x,y) representing location) into 3 clusters: $A_1(2,10)$, $A_2(7,5)$, $A_3(8,4)$, $A_4(5,8)$, $A_5(7,5)$, $A_6(6,4)$, $A_7(1,2)$, $A_8(4,9)$

initial clusters: $A_1(2,10)$; $A_2(5,8)$; $A_2(1,2)$

→ Calculation of distance b/w point $A(2,10)$ & $C_1(2:10)$

$P(A_1, C_1) = |x_2 - x_1| + |y_2 - y_1| = 0$

$C_2(5,8)$:

$P(A, C_2) = 5$

$C_3(1,2)$:

$P(A_1, C_3) = |2-1| + |10-2| = 8+1 = 9$

we calculate distance of other points from each of the center of 3 cluster

for cluster 01

- we have only one point $A_1(2,10)$ in cluster 01
- so cluster center remains the same.

iteration -1

Given pts	Distance from C_1	Dist from C_2	Dist from C_3	Point Belongs to
$A_1(2,10)$	0	5	9	C_1
$A_2(2,5)$	5	6	4	C_3
$A_3(8,4)$	12	7	9	C_2
$A_4(5,8)$	5	0	10	C_2
$A_5(7,5)$	10	5	9	C_2
$A_6(6,4)$	10	5	7	C_2
$A_7(1,2)$	9	10	0	C_3
$A_8(4,9)$	3	2	10	C_2

For 02

centre of cluster 02:

$((8+5+7+6+4)/5, (4+8+5+4+9)/5) = (6,6)$

for cluster 03:

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

This is completion of iteration 01

iteration 02.

Given Pts	Dist from C_1	Dist from C_2	Dist from C_3	Point Belongs to cluster
$A_1(2,10)$	0	8	7	C_1
$A_2(2,5)$	5	5	2	C_3
$A_3(8,4)$	12	4	7	C_2
$A_4(5,8)$	5	3	8	C_2
$A_5(7,5)$	10	2	7	C_2
$A_6(6,4)$	10	2	5	C_2
$A_7(1,2)$	9	9	2	C_3
$A_8(4,9)$	3	5	8	C_1

for C_1 cluster. $((2+4)/2, (10+9)/2) = (3, 9.5)$

for C_2 centre $= ((8+5+7+6)/4, (4+8+5+4)/4) = (6.5, 5.25)$

for C_3 centre $= ((2+1)/2, (5+2)/2) = (1.5, 3.5)$

iteration 03:

Given Pts	Dist from C_1	Dist from C_2	Dist from C_3	Point Belongs to cluster
$A_1(2,10)$	1.5	9.25	7	C_1
$A_2(2,5)$	5.5	4.75	2	C_3
$A_3(8,4)$	10.5	2.75	7	C_2
$A_4(5,8)$	3.5	4.75	8	C_1
$A_5(7,5)$	8.5	0.75	7	C_2
$A_6(6,4)$	8.5	1.75	5	C_2
$A_7(1,2)$	9.5	8.75	2	C_3
$A_8(8,9)$	1.5	6.25	8	C_1

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

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

for C_3 $((2+1)/2) = 1.5, 3.5)$

iteration 04

Given Pts	Dist from C_1	Dist from C_2	Dist from C_3	Points Belongs to cluster
$A_1(2,10)$	1.66	10.07	8	C_1
$A_2(2,5)$	5.66	5.67	2	C_3
$A_3(8,4)$	9.34	1.33	7	C_2
$A_4(5,8)$	2.34	5.67	8	C_1
$A_5(7,5)$	7.34	0.67	7	C_2
$A_6(6,4)$	7.34	1.33	6	C_2
$A_7(1,2)$	9.66	8.37	2	C_3
$A_8(8,9)$	0.34	7.66	8	C_1

→ centre of clusters are (3.66, 9) (7, 4.33) (1.5, 3.5)

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

data = {
    'Name': [f'Person_{i+1}' for i in range(50)],
    'Age': np.random.randint(18, 70, size=50),
    'Income': np.random.randint(20000, 120000, size=50)
}

df = pd.DataFrame(data)

df.to_csv('income.csv', index=False)

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

X = df[['Age', 'Income']]

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

X_train, X_test = train_test_split(X_scaled, test_size=0.2, random_state=42)

sse = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train)
    sse.append(kmeans.inertia_)

plt.plot(k_range, sse, marker='o')
plt.title('SSE vs Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Errors (SSE)')
plt.show()

optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_train)
y_pred = kmeans.predict(X_test)

print(f'Predicted Clusters for Test Data: {y_pred}')
```

Program 11

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

Screenshot

Date : 6/5/2025

Page No.:

Lab - 11

1) Calculate mean
2) Calculation of covariance matrix
3) Eigenvalues of the covariance matrix
4) Computation of the eigenvectors - Unit eigen-vectors
5) Computation of first principal component
6) Geometrical meaning of first principle components.

Given the data in Table, reduce the dimension from 2 to 1 using the principle component Analysis Algorithm.

Feature	example1	2	3	4
x_1	4	8	13	7
x_2	11	4	5	14

step 1: calculate mean.

$$\bar{x}_1 = \frac{4+8+13+7}{4} = 8$$

$$\bar{x}_2 = \frac{11+4+5+14}{4} = 8.5$$

step 2: calculate the covariance matrix.

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

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

$$= 14.$$

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

$$= \frac{1}{3}\left((11-8.5)(4-8) + (4-8.5)(8-8) + (5-8.5)(12-8) + (14-8.5)(7-8)\right)$$

$$= -11$$

$$\text{cov}(x_2, x_1) = \text{cov}(x_1, x_2) = -11$$

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

$$= \frac{1}{3}\left((11-8.5)^2 + (4-8.5)^2 + (5-8.5)^2 + (14-8.5)^2\right)$$

$$= 23.$$

=) The covariance matrix is

$$S = \begin{bmatrix} \text{cov}(x_1, x_1) & \text{cov}(x_1, x_2) \\ \text{cov}(x_2, x_1) & \text{cov}(x_2, x_2) \end{bmatrix}$$

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

step 3:

$$0 = \det(S - \lambda I)$$

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

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

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

solving the characteristic equation we get

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

$$= 30.3849,\ 6.6151$$

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

step 4: computation of the eigenvector.

$$\lambda = \lambda_1.$$

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

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

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

$$= \begin{bmatrix} (14-\lambda_1)u_1 - 11u_2 \\ -11u_1 + (23-\lambda_1)u_2 \end{bmatrix}$$

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

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

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

$$u_1 = 11t, \quad u_2 = (14-\lambda)t$$

taking t is any real number

taking $t = 1$

$$U_1 = \begin{bmatrix} 11 \\ 14 - \lambda_1 \end{bmatrix}$$

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

$$= \sqrt{11^2 + (14 - 30.3849)^2}$$

$$= 19.7348$$

$$\Rightarrow \quad e^1 = \begin{bmatrix} 11 / \|U_1\| \\ (14 - \lambda_1) / \|U_1\| \end{bmatrix}$$

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

=> by computing similar steps we get

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

step 5: computation of first principal component

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

$$e_1^T \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix} = (0.5574 \; -0.83) \begin{bmatrix} x_{1k} - \bar{x} \\ x_{2k} - \bar{x} \end{bmatrix}$$

$$= 0.5574 (x_{1k} - \bar{x}_1) + 0.8303 (x - \bar{x}_2)$$

Code:

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

df = pd.read_csv('heart (2).csv')

z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
df_no_outliers = df[(z_scores < 3).all(axis=1)]

df_cleaned = df_no_outliers.copy()
for col in df_cleaned.select_dtypes(include='object').columns:
    df_cleaned[col] = LabelEncoder().fit_transform(df_cleaned[col])

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

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

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

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

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

pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42,
stratify=y)
```

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