VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

SIDDHARTH H G (1BM22CS276)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

Sep-2024 to Jan-2025

B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by **SIDDHARTH H G(1BM22CS276),** 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.

Lab Faculty Incharge Name: **Ms. Saritha A N** Assistant Professor Department of CSE, BMSCE	 **Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

Index

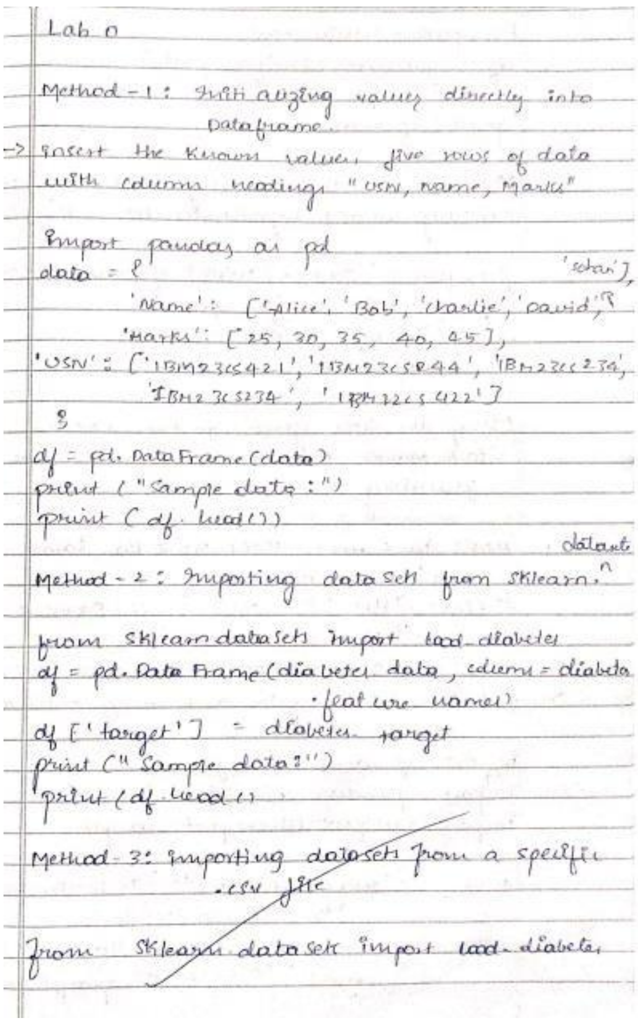
Github Link:

https://github.com/siddharthhg01/machine_learning

Program 1

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

Screenshot:



Lab 0

Method - 1: Initializing values directly into Dataframe.

-> Insert the known values, five rows of data with column headings "USN, name, Marks"

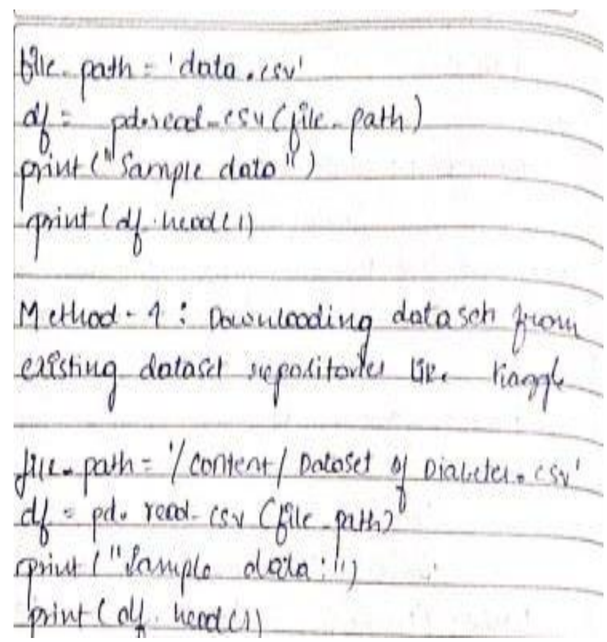
Import pandas as pd
data = {
'Name': ['Alice', 'Bob', 'Charlie', 'David', 'sohan'],
'Marks': [25, 30, 35, 40, 45],
'USN': ['1BM23CS421', '1BM23CS844', '1BM23CS234', '1BM23CS234', '1BM23CS422']
}
df = pd.DataFrame(data)
print ("Sample data:")
print (df.head())

Method - 2: Importing datasets from sklearn.datasets

from sklearn.datasets import load_diabetes
df = pd.DataFrame(diabetes.data, columns = diabetes.feature_names)
df ['target'] = diabetes.target
print ("Sample data:")
print (df.head())

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

from sklearn.datasets import load_diabetes



file_path = 'data.csv'
df = pd.read_csv(file_path)
print ("Sample data")
print (df.head())

Method - 4: Downloading datasets from existing dataset repositories like kaggle

file_path = '/content/Dataset of Diabetes.csv'
df = pd.read_csv(file_path)
print ("Sample data:")
print (df.head())

Code:

import pandas as pd

data = {

'Name': ['Alice', 'Bob', 'Charlie'],

'Age': [25, 30, 22],

```
    'Department': ['HR', 'Finance', 'IT']
  }

df = pd.DataFrame(data) df.to_csv('data.csv',
index=False) print("Sample data exported to
'data.csv'.")

imported_df = pd.read_csv('data.csv') print("\nImported
Data from 'data.csv':") print(imported_df)

imported_df['Age'] = imported_df['Age'] + 1
imported_df.to_csv('updated_data.csv', index=False)
print("\nUpdated data exported to 'updated_data.csv'.")
```

Program 2

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

Screenshot:

closing_prices = pd.DataFrame()
daily_returns = pd.DataFrame()
for ticker in tickers:
Stock_data = data[ticker]
closing_prices[ticker] = Stock_data["close"]
daily_returns [ticker] = Stock_data["close"].
pct_change()

Lab-1:

write the python code, consider the filename as "housing.csv"
(i) to load a csv file into the dataframe
(ii) to display the information of all columns
(iii) To display statistical information of all numerical
(iv) To display the count of unique labels for "ocean proximity" column
(v) To display which attributes (columns) in a dataset have missing values count greater than zero

Import pandas as pd
df = pd.read_csv("housing.csv")
(i) print (df.info)
(ii) print (df. describe())
(iii) df

Using the code given, do the exercise of "Stock Market Data Analysis", considering the following

1 HDFC Bank Ltd., ICICI Bank Ltd, Kotak Mahindra Bank Ltd
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
2. Start date: 2024-01-01, End date: 2024-12-30
3. plot the closing price and daily returns

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", groupby="ticker"

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMax
    Scaler, Standard Scaler

diabetes_df = pd.read_csv("/content/Dataset of
                    Diabetes.csv")
adult_income_df = pd.read_csv("adult.csv")

for column in diabetes_df.select_dtypes(
    include=['object']).columns:
  diabetes_df[column] = pd.to_numeric(
    diabetes_df[column], errors='coerce')

diabetes_df.fillna(diabetes_df.mean(),
        inplace=True)
adult_income_df = pd.get_dummies(adult_
 income_df, drop_first=True)

def remove_outliers(df, column):
   Q1 = df[column].quantile(0.25)
   Q3 = df[column].quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * QR
  upper_bound = Q3 + 1.5 * QR

for col in diabetes_df.select_dtypes(
 include=np.number).columns:
  diabetes_df = remove_outliers(
        diabetes_df, col).
```

Code:

import pandas as pd

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

print(df.head())

df.info

```
df.isnull.sum()

from sklearn.preprocessing import OneHotEncoder categorical_cols =

df.select_dtypes(include=['object']).columns print("Categorical columns:",

categorical_cols)

encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore') encoded_data

= encoder.fit_transform(df[categorical_cols])

encoded_df = pd.DataFrame(encoded_data,
columns=encoder.get_feature_names_out(categorical_cols))

df = pd.concat([df, encoded_df], axis=1)

df.drop(categorical_cols, axis=1, inplace=True) df.head()

Q1 = df['AGE'].quantile(0.25)  Q3 =

df['AGE'].quantile(0.75)

print(Q1,Q3)

IQR = Q3 - Q1

lower_bound   =   Q1   -   1.5   *   IQR

upper_bound   =   Q3   +   1.5   *   IQR

print(lower_bound,upper_bound)

outliers = df[(df['AGE'minmax_scaler = MinMaxScaler() standard_scaler =

StandardScaler()

numerical_features = ['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']

df[numerical_features] = minmax_scaler.fit_transform(df[numerical_features]) df[numerical_features] =

standard_scaler.fit_transform(df[numerical_features])

print(df)] < lower_bound) | (df['AGE'] > upper_bound)]

print(outliers['AGE'])
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:

Lab-3:-

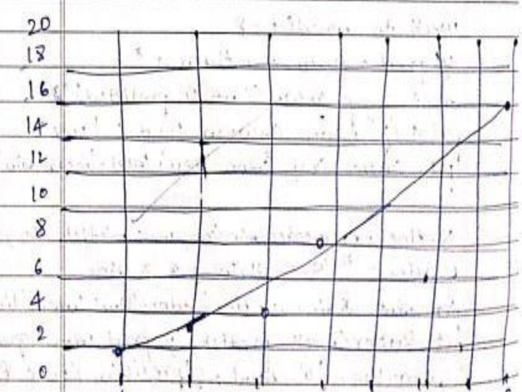
Linear Regression

```
Import numpy as np
Import matplotlib pyplot as plt

x = np.array([1,2,3,4])
y = np.array([1,3,4,8])

x_mean = np.mean(x)
y_mean = np.mean(y)
numerator = np.sum((x - x_mean)* (y - y_mean))
denominator = np.sum((x - x_mean)** 2)
theta_1 = numerator/denominator

theta_0 = y_mean - theta_1 * x_mean

print(f "Intercept (theta_0): {theta_0}")
print(f "Slope (theta_1): {theta_1}")

week to predict = 8
y_pred = theta_0 + theta_1 * x
mse = np.mean((y - y_predicted)** 2)
print(f "Mean Squared Error : {mse}")
plt. scatter (x,y, color='red', label='Actual data')

x_line = np.linspace(min(x), week_predict, 100)
y_line = theta_0 + theta_1 * x_line
plt. plot (x_line, y_line, color='blue', label='line')
plt. scatter(week_predict, y_pred, color='green'
 , marker='o', label = f'prediction (week {
   week_to_predict})')
plt. xlabel ("weeks")
plt. ylabel ("values")
plt. title("Linear Regression fit")
plt. legend()
plt. grid().
plt. show()
```

O/p:-

Intercept (theta_0): -1.5
Slope (theta_1): 2.2
predicted value for week 8: 16.1
Mean Squared Error : 0.44 999

O Actual data
— Fitted line
● prediction (week 8)



Using Matrix Method

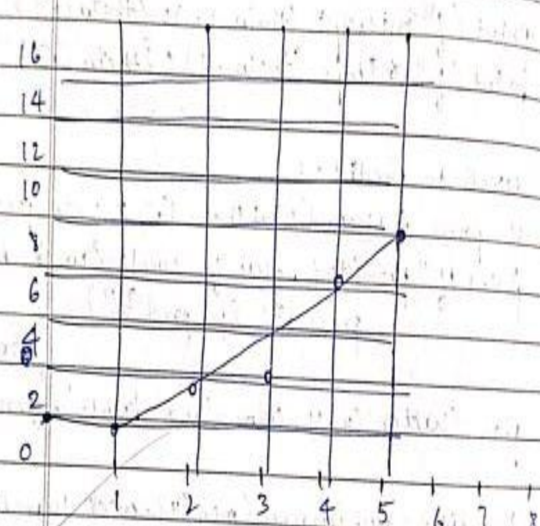
```
x = np.vstack ([np.ones (len(x)), x]).T
theta = np.linalg.inv (X.T @ X) @ X.T @ y

print (f"Intercept (theta_0): {theta[0]}")
print (f"Slope (theta_1) : {theta [1]}")

week_to_predict = 5
y_pred = theta [0] + theta [1] * week_predict
print (f" predicted value for week {week_to_predict}: {y_pred}")

plt. Scatter (x, y, color='red', label='actual data')

x_line = np. linspace (min (x), week_to_predict, 100)
y_line = theta[0] + theta [1] * x_line
plt.plot (x_line, y_line, color='blue', label='Fitted Line')

plt. scatter (week_to_predict, y_pred, color='green', marker='o', label=f'prediction (week {week_to_predict})

plt. x_label ("weeks")
plt.y label ("values")
plt. title ("Linear Regression Fit")
plt. legend ()
plt. grid()
plt. show()
```

O/P:-

Intercept (theta_0): -1.5

Slope (theta_1): 2.2000

predicted value for week 5: 9.5000



Code:

```
import pandas as pd

import numpy as np

from sklearn import linear_model import

matplotlib.pyplot as plt

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

plt.xlabel('area')

plt.ylabel('price')

plt.scatter(df.area,df.price,color='red',marker='+')

new_df = df.drop('price',axis='columns') new_df

price = df.price

price

reg = linear_model.LinearRegression()

reg.fit(new_df,price) reg.predict([[3300]])

reg.coef_ reg.intercept_

3300*135.78767123 + 180616.43835616432

reg.predict([[5000]])
```

```
df = pd.read_csv('/content/canada_per_capita_income.csv') new_df =
df.drop('per_capita_income',axis='columns')
reg = linear_model.LinearRegression() per_capita_income
= y = df['per_capita_income'].values
reg.fit(new_df,per_capita_income)
print(reg.coef_)
print(reg.intercept_)
predicted_income = reg.predict([[2020]])
print("predicted Income in the year 2020:" , predicted_income)
plt.scatter(df['year'], per_capita_income, color='blue', label='Data Points')
plt.plot(df['year'], reg.predict(new_df), color='red', label='Regression Line')
plt.xlabel('Year')
plt.ylabel('Per Capita Income (US$)') plt.title('Regression
Line: Per Capita Income vs Year') plt.legend()
plt.show()
df = pd.read_csv('/content/salary.csv')
df.YearsExperience.median()
df.YearsExperience = df.YearsExperience.fillna(df.YearsExperience.median()) reg
= linear_model.LinearRegression()
reg.fit(df.drop('Salary',axis='columns'),df.Salary)
print(reg.coef_)
print(reg.intercept_)
print("Predicted Salary of Person with 12 years of Experience: ",reg.predict([[12]]))
```

```
df = pd.read_csv('/content/hiring.csv') experience_map =

{

        'one':1,'two':2,'three':3,'four':4,'five':5,'six':6,'seven':7,'eight':8,'nine':9,'ten':10,'eleven':11,'twelve':12

         }

experience_map = df['experience'] = df['experience'].map(experience_map)

df.test_score = df.test_score.fillna(df.test_score.median())

df.experience = df.experience.fillna(df.experience.median()) reg

= linear_model.LinearRegression()

reg.fit(df.drop('salary',axis='columns'),df.salary) print(reg.coef_)

print(reg.intercept_)

print("Predicted Salary of Person with 2 years of Experience, 9 test score, 6 interview score:
",reg.predict([[2, 9, 6]]))

print("Predicted Salary of Person with 12 years of Experience, 10 test score, 10 interview score:
",reg.predict([[12, 10, 10]]))

df = pd.read_csv('/content/1000_Companies.csv')

experience_map = {

         'New York':1,'California':2,'Florida':3

         }

experience_map = df['State'] = df['State'].map(experience_map) reg

= linear_model.LinearRegression()

reg.fit(df.drop('Profit',axis='columns'),df.Profit)

print(reg.coef_)

print(reg.intercept_)

print(reg.predict([[91694.48, 515841.3, 11931.24,3]]))
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:

Lab - 4:

1) Consider a Binary classification problem where we want to predict whether a student will pass or fail based on their study hours. The logistic regression model has been trained, and the learned parameters are $a_0 = -5$ & $a_1 = 0.8$

(a) Write the logistic Regression

→ $a_0 = -5$ & $a_1 = 0.8$

⇒ $z = a_0 + a_1 \times x = -5 + (0.8) \times x$

$\therefore \boxed{p(pass|x) = \frac{1}{1 + e^{(-5 + 0.8x)}}}$

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

$p(pass|x)$ where $x = 7$

$\therefore \frac{1}{1 + e^{(5 - 0.8(7))}} = \frac{1}{1 + e^{(5 - 5.6)}}$

$= \boxed{\frac{1}{1 + e^{(-0.6)}} = 0.64}$

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

Student pass = 0.64 > 0.5 so the student will pass

2. Consider $z = [2, 1, 0]$ for three classes. Apply Softmax function to find the probability values of three classes.

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

for first class $z_k = 2$,

$$\text{Softmax}(z_k) = \frac{e^2}{e^2 + e^1 + e^0} = \boxed{0.665}$$

2nd class

$$\text{Softmax}(z_k) = \frac{e^1}{e^2 + e^1 + e^0} = \boxed{0.245}$$

3rd class

$$\text{Softmax}(z_k) = \frac{e^0}{e^2 + e^1 + e^0} = \boxed{0.090}$$

∴ probabilities = [0.665, 0.245, 0.090]

(3) After Building the logistic Regression models,

(i) For the data set file "HR-comma-sep.csv",

(a) The key variables that affect employee retention are:

(•) Employee with lower satisfaction level

(•) Extremely high or low working hours

(•) Lack of promotion

(•) Employees with low salaries having high turn over rate.

→ Accuracy → 80 85% and if dataset is imbalanced, accuracy alone can't be used.

2. Consider $z = [2, 1, 0]$ for three classes
Apply Softmax function to find the probability values of three classes.

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

for first class $z_k = 2$,

$$\text{Softmax}(z_k) = \frac{e^2}{e^2+e^1+e^0} = \boxed{0.665}$$

2nd class

$$\text{Softmax}(z_k) = \frac{e^1}{e^2+e^1+e^0} = \boxed{0.245}$$

3rd class

$$\text{Softmax}(z_k) = \frac{e^0}{e^2+e^1+e^0} = \boxed{0.090}$$

∴ probabilities = [0.665; 0.245; 0.090]

(3) After Building the logistic Regression models,

(i) For the data set file "HR-comma-sep.csv"

(a) The key variables that affect employee retention are:

(•) Employee with lower satisfaction level
(•) extremely high or low working hours
(•) Lack of promotion
(•) Employees with low salaries having high turn over rate.

→ Accuracy → 80 85% and if dataset is imbalanced, accuracy alone can't be used

Code:

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 OneHotEncoder, StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix #

Load dataset

```
file_path = "/content/HR_comma_sep.csv" df =

pd.read_csv(file_path)

# Exploratory Data Analysis (EDA)

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

sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Correlation Heatmap")

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

sns.countplot(x="left", data=df, palette="Set2")

plt.title("Employee Retention Distribution")

plt.xlabel("Left Company (1 = Yes, 0 = No)")

plt.ylabel("Count")

plt.show()
```

```
# Impact of salary on employee retention

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

sns.countplot(x="salary", hue="left", data=df, palette="muted")

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

plt.xlabel("Salary Level")

plt.ylabel("Count")

plt.legend(title="Left Company", labels=["Stayed", "Left"])

plt.show()

# Correlation between department and employee retention

plt.figure(figsize=(12,5))

sns.countplot(x="Department",   hue="left",   data=df,   palette="pastel")

plt.title("Correlation  between  Department  and  Employee  Retention")

plt.xlabel("Department")

plt.ylabel("Count") plt.xticks(rotation=45)

plt.legend(title="Left Company", labels=["Stayed", "Left"])

plt.show()

# Selecting important features

features = ["satisfaction_level", "time_spend_company", "number_project",
"average_montly_hours", "salary", "Department"]

X = df[features] y =

df["left"]

# One-hot encode categorical variables

X = pd.get_dummies(X, columns=["salary", "Department"], drop_first=True)
```

```
# Standardize numerical features

scaler = StandardScaler()

X.iloc[:, :4] = scaler.fit_transform(X.iloc[:, :4])

# Split dataset into training and testing sets (80-20 split)

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

Train logistic regression model

model = LogisticRegression()

model.fit(X_train, y_train)

# Predict and measure accuracy

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.2f}")

# Plot confusion matrix

plt.figure(figsize=(6,5))

sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Stayed", "Left"],
yticklabels=["Stayed", "Left"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()



zoo_data = pd.read_csv("/content/zoo-data.csv")
```

```
zoo_classes = pd.read_csv("/content/zoo-class-type.csv") #

Merge datasets on class_type if needed

    if 'class_type' in zoo_data.columns and 'class_type' in zoo_classes.columns:

        zoo_data = zoo_data.merge(zoo_classes, on='class_type', how='left')

# Separate features and target variable

X = zoo_data.drop(columns=['class_type', 'animal_name']) # Assuming 'animal_name' is
non-numeric

y = zoo_data['class_type']

# 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, stratify=y) #

Scale the features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Train Logistic Regression model

model = LogisticRegression(multi_class='ovr', solver='lbfgs', max_iter=200)

model.fit(X_train, y_train)

# Predictions

y_pred = model.predict(X_test) #

Evaluate accuracy

accuracy = accuracy_score(y_test, y_pred) print(f"Model

Accuracy: {accuracy:.2f}")
```

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred) plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y),
yticklabels=np.unique(y))

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()
```

Program 5

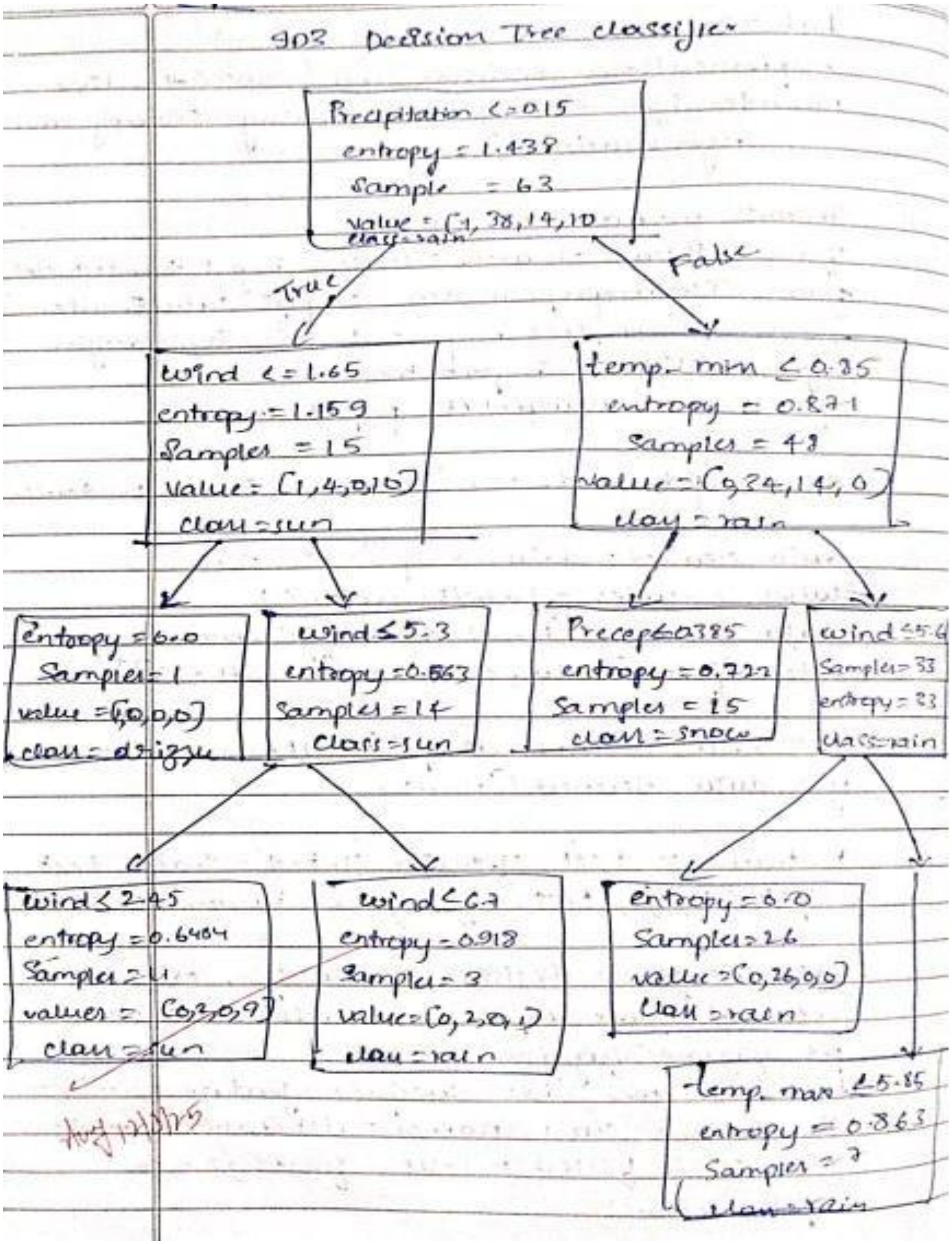
Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

Screenshot:

Lab - 2 :

Implementing Decision Tree :- ID3 (Iterative Dichotomizer 3) Algorithm using Entropy and Information

```
Import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt.

data = pd.read_csv('/content/Seattle-weather.csv')

data_cleaned = data.drop('date', axis=1)
label_encoder = LabelEncoder()
data_cleaned['weather'] = label_encoder.fit
_transform(data_cleaned['weather'])

x = data_cleaned.drop('weather', axis=1)
y = data_cleaned('weather')

x_train, x_test, y_train, y_test = train_test_
split(x, y, test_size = 0.2, random_state=42)

id3_classifier = DecisionTreeClassifier(criterion =
'entropy', max_depth = 4, random_state = 42)
plt.figure(figsize = (20,12))
tree.plot_tree(id3_classifier, feature_names =
x.columns, class_names = list(label_encoder.
classes_), filled = True, fontsize = 10)
plt.show()
```



Code:

import pandas as pd

```
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

file_path = "/content/iris.csv"

df = pd.read_csv(file_path)

# Separate features and target

X = df.drop(columns=['species']) y =

df['species']

# Encode target labels

y = LabelEncoder().fit_transform(y)

# Split data into training (80%) and testing (20%)

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

Create and train the DecisionTree classifier

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

# Make predictions on the test set y_pred =

clf.predict(X_test)

# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
# Generate confusion matrix

cm = confusion_matrix(y_test, y_pred) #
```

Plot confusion matrix

plt.figure(figsize=(6,5))

sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=df['species'].unique(), yticklabels=df['species'].unique())

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

```
label_encoders = {}

    for column in df.columns:

        le = LabelEncoder()

        df[column] = le.fit_transform(df[column])

        label_encoders[column] = le
```

Split the dataset into features and target X =

df.drop('species', axis=1)

y = df['species']

Initialize the Decision Tree Classifier with entropy as the criterion clf =

DecisionTreeClassifier(criterion='entropy')

Train the classifier

clf.fit(X_train, y_train) #

Make predictions

y_pred = clf.predict(X_test) #

Evaluate the classifier

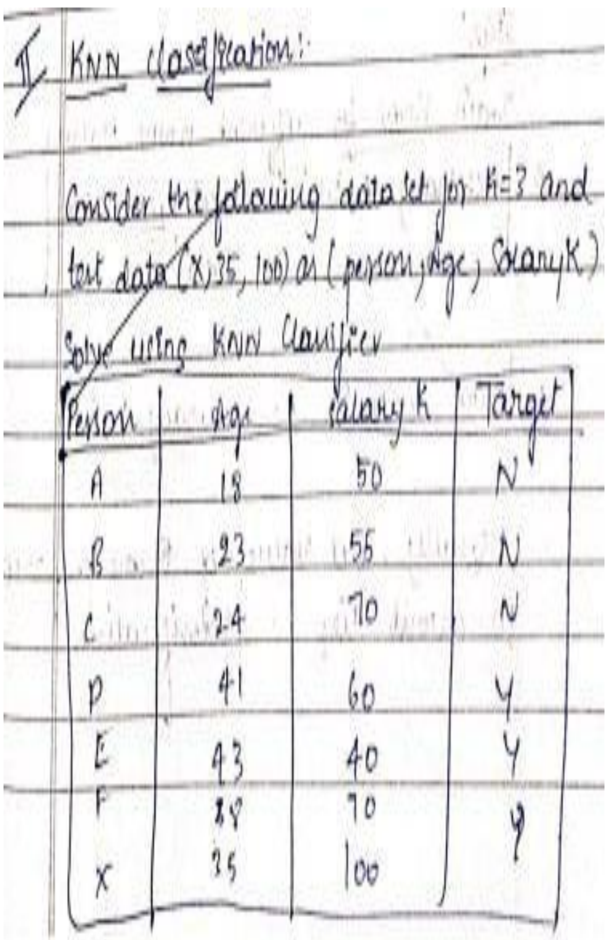
accuracy = accuracy_score(y_test, y_pred)

```
print(f'Accuracy: {accuracy:.2f}')

print(classification_report(y_test, y_pred, target_names=['Iris-setosa',
'Iris-versicolor','Iris-virginica']))

# Optionally, visualize the decision tree

from sklearn.tree import plot_tree import

matplotlib.pyplot as plt

plt.figure(figsize=(12,8))

plot_tree(clf, filled=True, feature_names=X.columns, class_names=['Setosa', 'Versicolor','Virginica'])

plt.show()

file_path = "/content/drug.csv" df =

pd.read_csv(file_path)

# Encode categorical features categorical_cols

= ['Sex', 'BP', 'Cholesterol']

df[categorical_cols] = df[categorical_cols].apply(LabelEncoder().fit_transform) #

Separate features and target

X = df.drop(columns=['Drug']) y =

df['Drug']

# Encode target labels

y = LabelEncoder().fit_transform(y)

# Split data into training (80%) and testing (20%)

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

Create and train the DecisionTree classifier

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

# Make predictions on the test set y_pred =
```

Program 6

Build KNN Classification model for a given dataset.

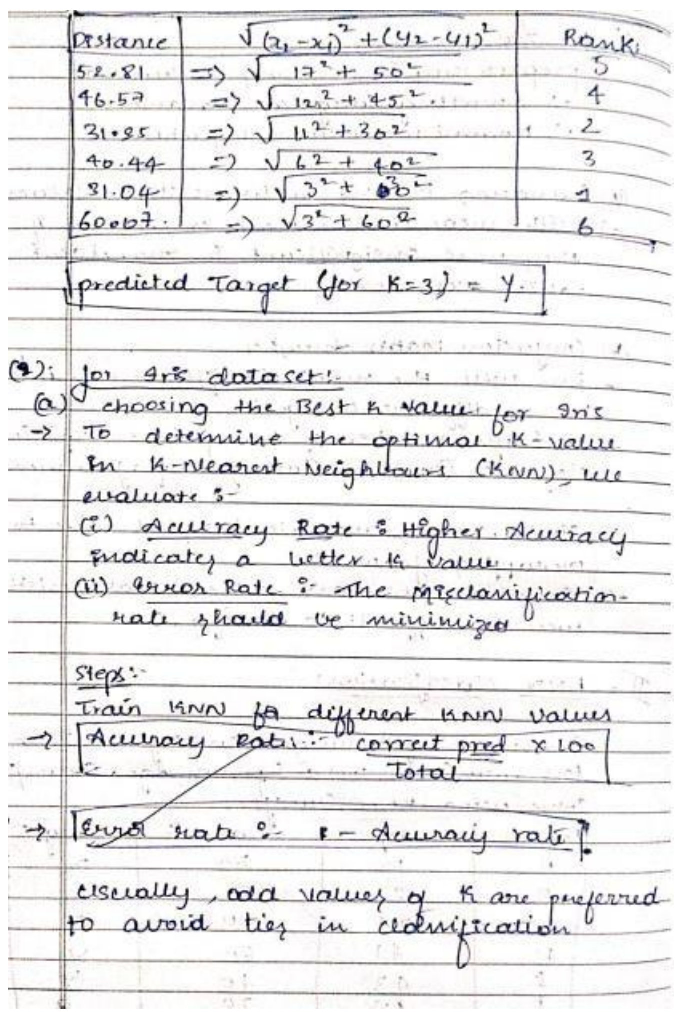
Screenshot:



I. KNN classification:

Consider the following data set for K=3 and test data (X, 35, 100) as (person, Age, Salary K)
Solve using KNN Classifier

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



Distance	$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$	Rank
52.81	$\Rightarrow \sqrt{17^2+50^2}$	5
46.57	$\Rightarrow \sqrt{12^2+45^2}$	4
31.95	$\Rightarrow \sqrt{11^2+30^2}$	2
40.44	$\Rightarrow \sqrt{6^2+40^2}$	3
31.04	$\Rightarrow \sqrt{3^2+30^2}$	1
60.07	$\Rightarrow \sqrt{3^2+60^2}$	6

predicted Target (for K=3) = Y.

(2): for Iris dataset:
(a) choosing the Best K value for Iris
→ To determine the optimal K-value in K-Nearest Neighbours (KNN), we evaluate:-
(i) Accuracy Rate: Higher Accuracy indicates a better K value.
(ii) Error Rate: The misclassification rate should be minimized

Steps:-
Train KNN for different KNN values
→ Accuracy Rate: $\frac{\text{correct pred}}{\text{Total}} \times 100$
→ Error rate: 1 − Accuracy rate

Usually, odd values of K are preferred to avoid ties in classification

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 confusion_matrix, classification_report, accuracy_score import

matplotlib.pyplot as plt

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

df.head()

# Separate features and labels X

= df.drop('species', axis=1) y =

df['species']

# Split the 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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Find the best k value by plotting error rate

error_rate = []


for i in range(1, 31):

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train, y_train)


pred_i = knn.predict(X_test)

 error_rate.append(np.mean(pred_i != y_test)) #

Plotting error rates
plt.figure(figsize=(12,6))

   plt.plot(range(1,31), error_rate, color='blue', linestyle='dashed', marker='o',
```

```
            markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate') plt.show()
# Choose k with minimum error
optimal_k = error_rate.index(min(error_rate)) + 1
print(f"Optimal K value: {optimal_k}")
# Train the model with optimal k
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
# Predict the test set results
y_pred = knn.predict(X_test) #
Evaluate the model
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
class_report = classification_report(y_test, y_pred)


acc_score = accuracy_score(y_test, y_pred)
print("\nClassification Report:\n", class_report)
print("\nAccuracy Score:", acc_score)


 diabetes_df = pd.read_csv('/content/diabetes.csv')
# Display first few rows
diabetes_df.head()
# Separate features and target
X = diabetes_df.drop('Outcome', axis=1)
# Assuming 'Outcome' is the target variable based on common diabetes datasets
y = diabetes_df['Outcome']
# 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
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Finding the best k value
error_rate = []
for i in range(1, 31):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
```

```
    error_rate.append(np.mean(pred_i != y_test))

# Plotting error rates

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

plt.plot(range(1,31), error_rate, color='blue', linestyle='dashed', marker='o',

         markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')

plt.show()

# Choose optimal k

optimal_k = error_rate.index(min(error_rate)) + 1

print(f"Optimal K value: {optimal_k}")

# Train the model with optimal k

knn = KNeighborsClassifier(n_neighbors=optimal_k)

knn.fit(X_train, y_train)

# Predict the test set results

y_pred = knn.predict(X_test)

# Evaluate the model

conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8,6))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')
```

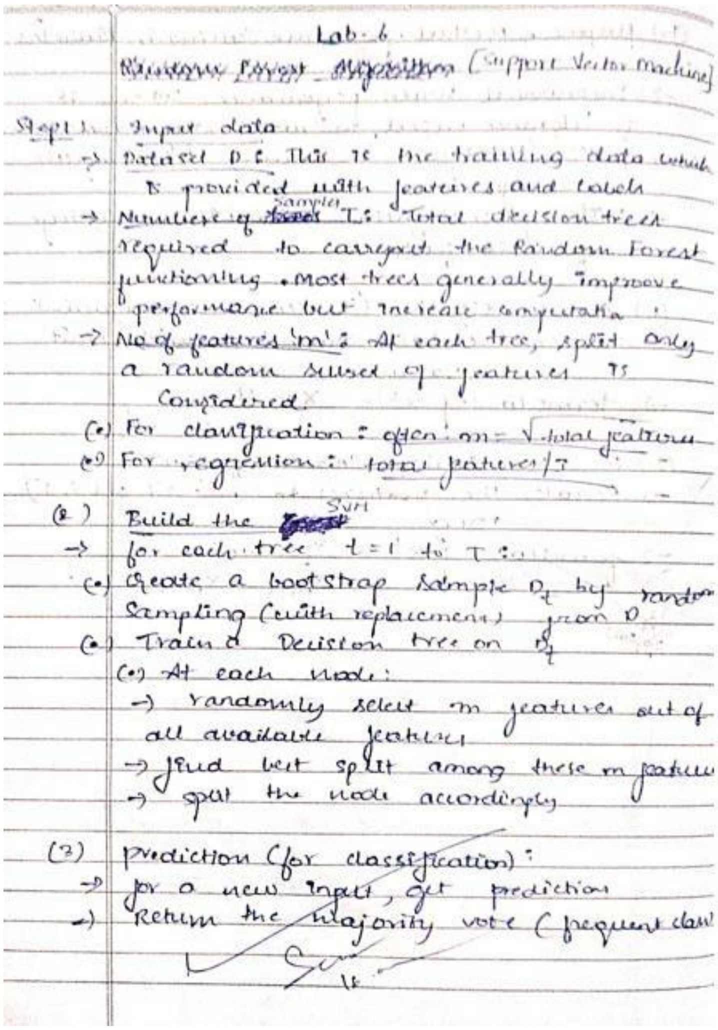
```
plt.show()

acc_score = accuracy_score(y_test, y_pred)

print("\nAccuracy Score:", acc_score)


heart_df = pd.read_csv('/content/heart.csv')

# Display first few rows

heart_df.head()

# Separate features and target

X = heart_df.drop('target', axis=1)

y = heart_df['target']

# Split the dataset

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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Find the best k value

error_rate = []

acc_scores = []

for i in range(1, 31):

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train, y_train)

    pred_i = knn.predict(X_test)

    acc_scores.append(accuracy_score(y_test, pred_i))

    error_rate.append(np.mean(pred_i != y_test))
```

```
plt.figure(figsize=(12,6))
plt.plot(range(1,31), acc_scores, color='green', linestyle='dashed', marker='o',
    markerfacecolor='blue', markersize=10)
plt.title('Accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.show()
optimal_k = acc_scores.index(max(acc_scores)) + 1
print(f"Optimal K value: {optimal_k}")
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
# Predict the test set results
y_pred = knn.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)
acc_score = accuracy_score(y_test, y_pred)
print("\nAccuracy Score:", acc_score)
```

Program 7

Build Support vector machine model for a given dataset.

Screenshot:



Lab-6

~~Random Forest Algorithm~~ (Support Vector machine)

Step 1: Input data

→ Dataset D: This is the training data which is provided with features and labels

→ Number of Samples T: Total decision trees required to carryout the Random Forest functioning. Most trees generally improve performance but increase computation.

→ No of features 'm': At each tree, split only a random subset of features is considered.

(•) For classification: often m = √total features

(•) For regression: total features/3

(2) Build the SVM

→ for each tree t = 1 to T:

(•) Create a bootstrap sample D_t by random sampling (with replacement) from D

(•) Train a Decision tree on D_t

(•) At each node:

→ randomly select m features out of all available features

→ find best split among these m features

→ split the node accordingly

(3) Prediction (for classification):

→ for a new input, get prediction

→ Return the majority vote (frequent class)

Code:

```
import pandas as pd import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
```

```
import matplotlib.pyplot as plt

df1=pd.read_csv("/content/iris.csv")



print("Iris\n",df1.head())

X_iris = df1.drop('species', axis=1) y_iris

= df1['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)

# Linear Kernel SVM

svm_linear = SVC(kernel='linear', random_state=42)

svm_linear.fit(X_train_iris, y_train_iris)

# RBF Kernel SVM

svm_rbf = SVC(kernel='rbf', random_state=42)

svm_rbf.fit(X_train_iris, y_train_iris)

y_pred_linear = svm_linear.predict(X_test_iris)

y_pred_rbf = svm_rbf.predict(X_test_iris)

# Accuracy and Confusion Matrix for Linear Kernel accuracy_linear =

accuracy_score(y_test_iris, y_pred_linear) conf_matrix_linear =

confusion_matrix(y_test_iris, y_pred_linear)

# Accuracy and Confusion Matrix for RBF Kernel accuracy_rbf

= accuracy_score(y_test_iris, y_pred_rbf) conf_matrix_rbf =

confusion_matrix(y_test_iris, y_pred_rbf) # Display Results

print(f"Linear Kernel Accuracy: {accuracy_linear}")

print(f"RBF Kernel Accuracy: {accuracy_rbf}")

# Confusion Matrices
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(conf_matrix_linear, annot=True, fmt='d', cmap='Blues', ax=ax1)

ax1.set_title("Linear Kernel Confusion Matrix")
```

```
ax1.set_xlabel('Predicted')

ax1.set_ylabel('Actual')

sns.heatmap(conf_matrix_rbf, annot=True, fmt='d', cmap='Blues', ax=ax2)

ax2.set_title("RBF Kernel Confusion Matrix")

ax2.set_xlabel('Predicted')

ax2.set_ylabel('Actual') plt.show()


df2=pd.read_csv("/content/letter-recognition.csv")

print("Letter-Recognition\n",df2.head())

X_letter = df2.drop('letter', axis=1) y_letter

= df2['letter']

y_letter = y_letter.astype('category').cat.codes

X_train_letter, X_test_letter, y_train_letter, y_test_letter = train_test_split(X_letter, y_letter,
test_size=0.2, random_state=42)

# Linear Kernel SVM for Letter Recognition

svm_linear_letter = SVC(kernel='linear', random_state=42, probability=True)

svm_linear_letter.fit(X_train_letter, y_train_letter)

# RBF Kernel SVM for Letter Recognition

svm_rbf_letter = SVC(kernel='rbf', random_state=42, probability=True)

svm_rbf_letter.fit(X_train_letter, y_train_letter)

y_pred_linear_letter = svm_linear_letter.predict(X_test_letter) y_pred_rbf_letter

= svm_rbf_letter.predict(X_test_letter) accuracy_linear_letter =

accuracy_score(y_test_letter, y_pred_linear_letter)

conf_matrix_linear_letter = confusion_matrix(y_test_letter, y_pred_linear_letter) accuracy_rbf_letter

= accuracy_score(y_test_letter, y_pred_rbf_letter)
```

conf_matrix_rbf_letter = confusion_matrix(y_test_letter, y_pred_rbf_letter) print(f"Linear Kernel Accuracy (Letter-recognition): {accuracy_linear_letter}") print(f"RBF Kernel Accuracy (Letter-recognition): {accuracy_rbf_letter}")

Confusion Matrices

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(25, 12)) sns.heatmap(conf_matrix_linear_letter, annot=True, fmt='d', cmap='Blues', ax=ax1) ax1.set_title("Linear Kernel Confusion Matrix")

ax1.set_xlabel('Predicted')

ax1.set_ylabel('Actual')

sns.heatmap(conf_matrix_rbf_letter, annot=True, fmt='d', cmap='Blues', ax=ax2) ax2.set_title("RBF Kernel Confusion Matrix")

ax2.set_xlabel('Predicted')

ax2.set_ylabel('Actual')

plt.show()

Plotting ROC curve for Linear Kernel

fpr, tpr, thresholds = roc_curve(y_test_letter, svm_linear_letter.predict_proba(X_test_letter)[:, 1], pos_label=1)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

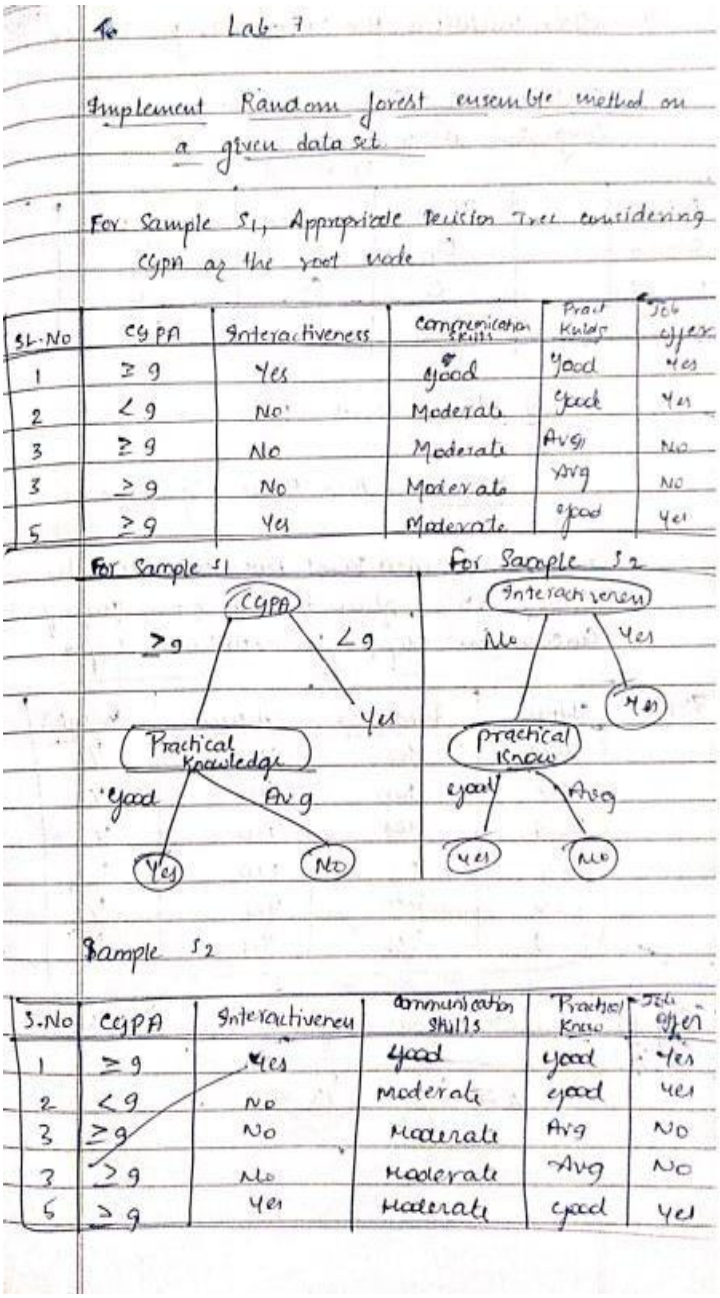
plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc='lower right')

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:



Lab 7

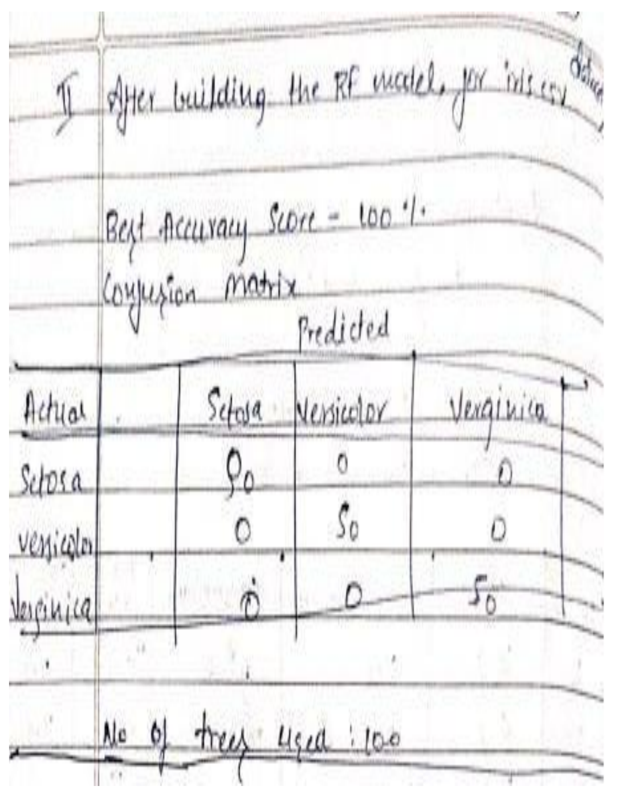
Implement Random forest ensemble method on a given data set

For Sample S_1, Appropriate Decision Tree considering CGPA as the root node

Sl-No	CGPA	Interactiveness	Communication Skills	Pract Knowledge	Job offers
1	≥ 9	Yes	good	good	Yes
2	< 9	No	Moderate	good	Yes
3	≥ 9	No	Moderate	Avg	No
3	≥ 9	No	Moderate	Avg	No
5	≥ 9	Yes	Moderate	good	Yes

Sample S_2

S-No	CGPA	Interactiveness	Communication Skills	Practical Know	Job offers
1	≥ 9	Yes	good	good	Yes
2	< 9	No	moderate	good	Yes
3	≥ 9	No	Moderate	Avg	No
3	≥ 9	No	Moderate	Avg	No
5	≥ 9	Yes	Moderate	good	Yes



After building the RF model, for iris csv dataset

Best Accuracy Score = 100 %

Confusion Matrix

		Predicted	
Actual	Setosa	Versicolor	Verginica
Setosa	50	0	0
Versicolor	0	50	0
Verginica	0	0	50

No of trees used : 100

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, classification_report import

matplotlib.pyplot as plt

import seaborn as sns #

Load the dataset

file_path = '/content/iris.csv' data =

pd.read_csv(file_path)

print("Columns:", data.columns)

# Assume last column is target, others are features X

= data.iloc[:, :-1]

y = data.iloc[:, -1] #

Split dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42) # 1

Build Random Forest 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) score_default

= accuracy_score(y_test, y_pred_default)
```

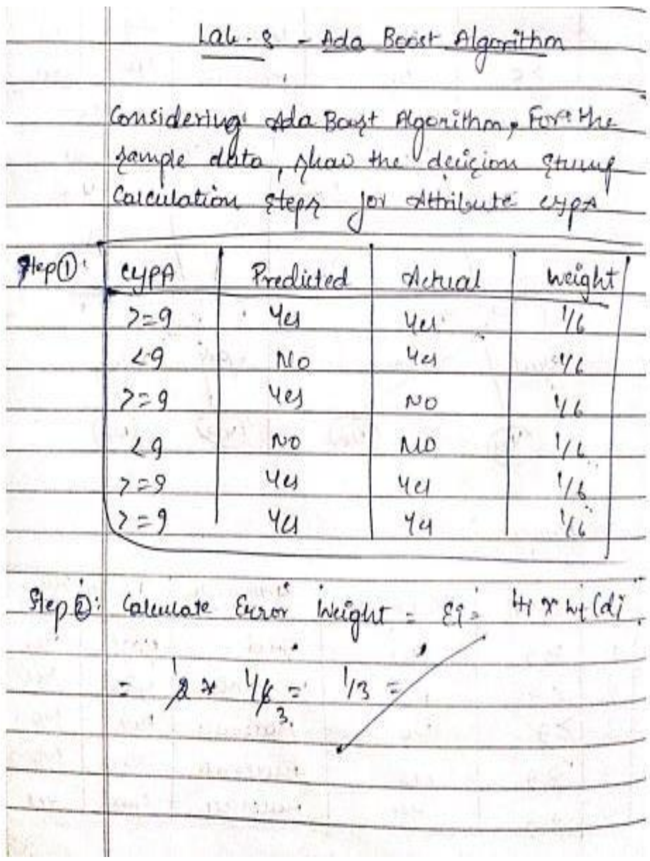
```
# Show confusion matrix
cm = confusion_matrix(y_test, y_pred_default)
print("\nConfusion Matrix (default 10 trees):")
print(cm)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix (n_estimators=10)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
print("\nClassification Report:")
print(classification_report(y_test, y_pred_default)) #
Show feature importance
importances = rf_default.feature_importances_
feature_names = X.columns
feat_importances = pd.Series(importances, index=feature_names)
feat_importances.sort_values().plot(kind='barh', figsize=(8,6))
plt.title('Feature Importances (n_estimators=10)')
plt.show() best_score =
0
best_n = 0
scores = []
```

```
n_values = range(1, 101, 5)  # Try from 1 to 100 in steps of 5 for

n in n_values:

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

        rf.fit(X_train, y_train)

        y_pred = rf.predict(X_test)

        score = accuracy_score(y_test, y_pred)

        scores.append(score)

        print(f"n_estimators={n}, Accuracy: {score:.4f}")

        if score > best_score:

            best_score = score

            best_n = n

print(f"\nBest accuracy {best_score:.4f} achieved with n_estimators={best_n}") #

Plot scores vs. number of trees

plt.figure(figsize=(10,6)) plt.plot(n_values,

scores, marker='o')

plt.xlabel('Number of Trees (n_estimators)')

plt.ylabel('Accuracy Score')

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

plt.grid(True)

plt.show()
```

Program 9

Implement Boosting ensemble method on a dataset.

Screenshot:



Lab - 8 - Ada Boost Algorithm

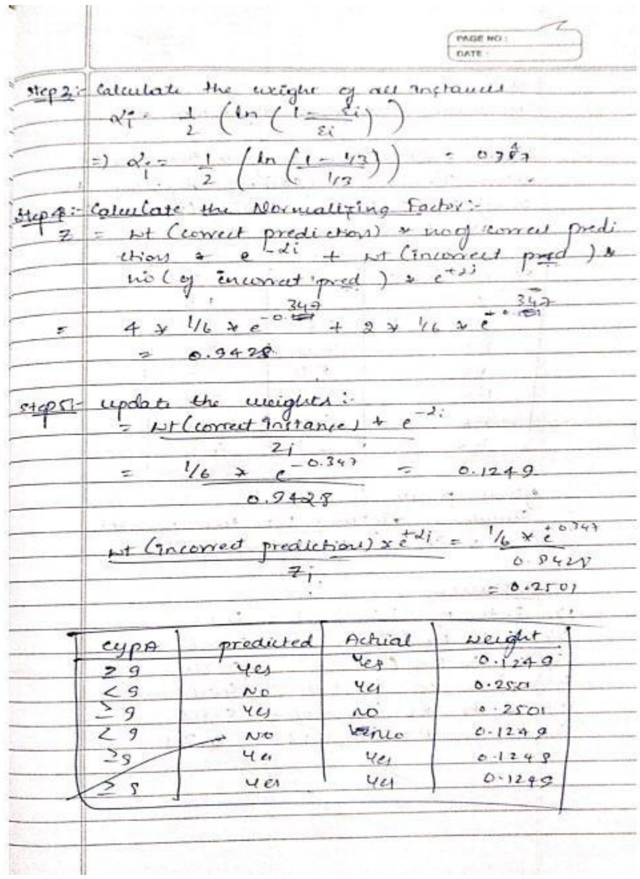
Considering Ada Boost Algorithm, For the sample data, show the decision stump Calculation steps for Attribute cgpa

Step 1:

cgpa	Predicted	Actual	weight
>=9	Yes	Yes	1/6
<9	No	Yes	1/6
>=9	Yes	NO	1/6
<9	NO	NO	1/6
>=9	Yes	Yes	1/6
>=9	Yes	Yes	1/6

Step 2: Calculate Error Weight = ε_i = Σ wt(d_i)

$= \cancel{2} \times \frac{1}{\cancel{6}_3} = \frac{1}{3}$



Step 3: Calculate the weight of all instances

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

$\Rightarrow \alpha_1 = \frac{1}{2}\left(\ln\left(\frac{1-1/3}{1/3}\right)\right) = 0.347$

Step 4: Calculate the Normalizing Factor:

Z = wt (correct predictions) × no of correct predictions × $e^{-\alpha_i}$ + wt (incorrect pred) × no (of incorrect pred) × $e^{+\alpha_i}$

$= 4 \times 1/6 \times e^{-0.347} + 2 \times 1/6 \times e^{0.347}$

$= 0.9428$

Step 5: update the weights:

$= \frac{\text{wt (correct instance)} \times e^{-\alpha_i}}{Z_i}$

$= \frac{1/6 \times e^{-0.347}}{0.9428} = 0.1249$

$\frac{\text{wt (incorrect prediction)} \times e^{+\alpha_i}}{Z_i} = \frac{1/6 \times e^{0.347}}{0.9428}$

$= 0.2501$

cgpa	predicted	Actual	weight
≥9	yes	yes	0.1249
<9	No	yes	0.2501
≥9	yes	NO	0.2501
<9	NO	NO	0.1249
≥9	yes	yes	0.1249
≥9	yes	yes	0.1249

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report import

matplotlib.pyplot as plt

import seaborn as sns #

Load the dataset
```

```
file_path = '/content/income.csv'

data = pd.read_csv(file_path)

# Inspect columns

print("Columns:", data.columns)

# Assume last column is target, others are features X

= data.iloc[:, :-1]

y = data.iloc[:, -1]


# Split dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42) # 1

Build AdaBoost with n_estimators=10

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

ada_default.fit(X_train, y_train)

y_pred_default = ada_default.predict(X_test)


score_default = accuracy_score(y_test, y_pred_default)

print(f"n_estimators=10, Accuracy: {score_default:.4f}") #

Show confusion matrix

cm = confusion_matrix(y_test, y_pred_default)

print("\nConfusion Matrix (n_estimators=10):")

print(cm)

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
```

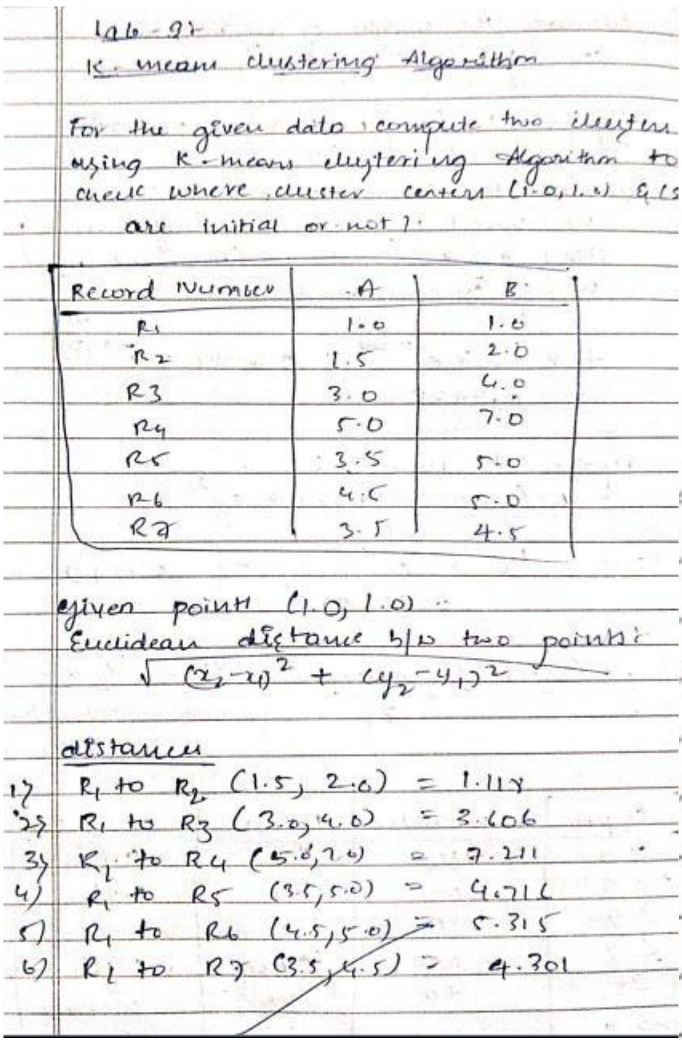
```
plt.title('Confusion Matrix (n_estimators=10)')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# Show classification report  print("\nClassification

Report:") print(classification_report(y_test,

y_pred_default)) # 2 Fine-tune number of estimators

best_score = 0

best_n = 0

scores = []

n_values = range(10, 201, 10)  # Try from 10 to 200 in steps of 10 for n

in n_values:

    ada = AdaBoostClassifier(n_estimators=n, random_state=42)
    ada.fit(X_train, y_train)

    y_pred = ada.predict(X_test)

    score = accuracy_score(y_test, y_pred)

    scores.append(score)

    print(f"n_estimators={n}, Accuracy: {score:.4f}")

    if score > best_score:

        best_score = score

        best_n = n

print(f"\nBest accuracy {best_score:.4f} achieved with n_estimators={best_n}") #

Plot scores vs. number of estimators
```

```
plt.figure(figsize=(10,6))
plt.plot(n_values, scores, marker='o') plt.xlabel('Number

of Estimators (n_estimators)') plt.ylabel('Accuracy

Score')

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

plt.grid(True)

plt.show()
```

Program 10

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

Screenshot:



lab - 9:

K. mean clustering Algorithm

For the given data compute two cluster using K-means clustering Algorithm to check where cluster centers (1.0,1.0) & (5... are initial or not?

Record Number	A	B
R1	1.0	1.0
R2	1.5	2.0
R3	3.0	4.0
R4	5.0	7.0
R5	3.5	5.0
R6	4.5	5.0
R7	3.5	4.5

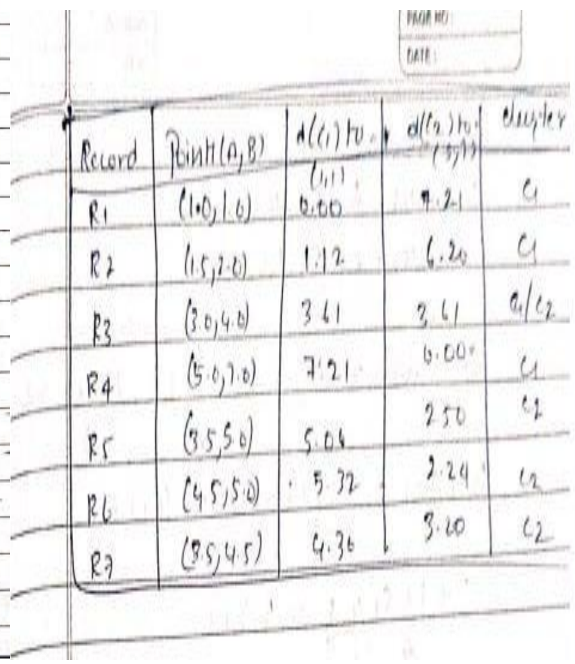
given point (1.0, 1.0):

Euclidean distance b/w two points:

$\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$

distance

1) R_1 to R_2 (1.5, 2.0) = 1.118
2) R_1 to R_3 (3.0, 4.0) = 3.606
3) R_1 to R_4 (5.0, 7.0) = 7.211
4) R_1 to R_5 (3.5, 5.0) = 4.716
5) R_1 to R_6 (4.5, 5.0) = 5.315
6) R_1 to R_7 (3.5, 4.5) = 4.301



PAGE NO:

DATE:

Record	Point(A,B)	d(c1) to (1,1)	d(c2) to (5,7)	cluster
R1	(1.0,1.0)	0.00	7.21	C1
R2	(1.5,2.0)	1.12	6.20	C1
R3	(3.0,4.0)	3.61	3.61	C1/C2
R4	(5.0,7.0)	7.21	0.00	C2
R5	(3.5,5.0)	5.04	2.50	C2
R6	(4.5,5.0)	5.32	2.24	C2
R7	(3.5,4.5)	4.36	3.10	C2

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

```
from scipy import stats

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score from

sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

df1=pd.read_csv("iris.csv")

df1.head()

df = df1.drop(['sepal_length','sepal_width','species'],axis=1)

scaler = StandardScaler()

scaled_df = scaler.fit_transform(df) wcss =

[]

for i in range(1, 11):

    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)

    kmeans.fit(scaled_df)

    wcss.append(kmeans.inertia_)
```

```
plt.plot(range(1, 11), wcss)

plt.title('Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)

pred_y = kmeans.fit_predict(scaled_df)

df['cluster'] = pred_y

plt.scatter(df['petal_length'],    df['petal_width'],    c=df['cluster'])

plt.title('Clusters of Iris Flowers')

plt.xlabel('Petal      Length')

plt.ylabel('Petal       Width')

plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:

Lab-10

Implementing Dimensionality reduction using principle component Analysis

Given the data, reduce the dimension from 2 to 1 using PCA compute for first PCA.

Feature	Example 1	Example 2	Example 3	Example 4
x_1	4	8	13	7
x_2	11	4	5	14

given:-

2 features x_1, x_2 with 4 examples:

$$\begin{bmatrix} 4 & 8 & 13 & 7 \\ 11 & 4 & 5 & 14 \end{bmatrix}$$

Eigen Values : $\lambda_1 = 30.3849$, $\lambda_2 = 6.6151$

Eigen Vectors: $e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$, $e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

Goal:- Reduce dimensionally from 2D to 1D

1) center the data :-

mean: $\mu x_1 = 4+8+13+7/4 = 8.0$

$\mu x_2 = 11+4+15+14/4 = 8.5$

Centered data:

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

Step 2: Project the centered data on the first principal component:

use dot product:

$z_i = e_1^T \cdot \text{centered\_data}_T$

compute each projection

Let $e_1 = [0.5574, -0.8307]^T$

(•) Example 1 : $z = (0.5574)(-4) + (-0.8303)(2.5) = -2.2296 - 2.07575 = -4.30535$

(•) Example 2 : $z = (0.5574)(0) + (-0.8303)(-4.5) = 0 + 3.73635 = 3.73635$

(•) Example 3 : $z = (0.5574)(5) + (-0.8303)(-3.5) = 2.787 + 2.90605 = 5.69305$

(•) Example 4 : $z = (0.5574)(-1) + (-0.8303)(5.5) = -0.5574 - 4.5665 = -5.12405$

Final answer:-

Projected data in 1D along first principal component

$$\begin{bmatrix} -4.30535 \\ 3.73635 \\ 5.69305 \\ -5.12405 \end{bmatrix}$$ //

7/5/2025

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from scipy import stats

import seaborn as sns
```

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

df1=pd.read_csv("heart.csv")

df1.head()

text_cols = df1.select_dtypes(include=['object']).columns

label_encoder = LabelEncoder()

for col in text_cols:

  df1[col] = label_encoder.fit_transform(df1[col])

print(df1.head())

X = df1.drop('HeartDisease', axis=1)

y = df1['HeartDisease']

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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

```
# Support Vector Machine
svm_model = SVC(kernel='linear', random_state=42)

svm_model.fit(X_train, y_train)

svm_predictions = svm_model.predict(X_test)

svm_accuracy = accuracy_score(y_test, svm_predictions)

print(f"SVM Accuracy: {svm_accuracy}")

# Logistic Regression

lr_model = LogisticRegression(random_state=42)

lr_model.fit(X_train, y_train)

lr_predictions = lr_model.predict(X_test)

lr_accuracy = accuracy_score(y_test, lr_predictions)

print(f"Logistic Regression Accuracy: {lr_accuracy}")

# Random Forest

rf_model = RandomForestClassifier(random_state=42)

rf_model.fit(X_train, y_train)

rf_predictions = rf_model.predict(X_test)

rf_accuracy = accuracy_score(y_test, rf_predictions)

print(f"Random Forest Accuracy: {rf_accuracy}")

models = {

"SVM": svm_accuracy,

"Logistic Regression": lr_accuracy,

"Random Forest": rf_accuracy
```

```
    }
best_model = max(models, key=models.get)
print(f"\nBest Model: {best_model} with accuracy {models[best_model]}")
pca = PCA(n_components=0.95)

X_train_pca = pca.fit_transform(X_train)

X_test_pca = pca.transform(X_test)

svm_model_pca = SVC(kernel='linear', random_state=42)

svm_model_pca.fit(X_train_pca, y_train)

svm_predictions_pca = svm_model_pca.predict(X_test_pca)

svm_accuracy_pca = accuracy_score(y_test, svm_predictions_pca)

print(f"SVM Accuracy (with PCA): {svm_accuracy_pca}")

lr_model_pca = LogisticRegression(random_state=42)

lr_model_pca.fit(X_train_pca, y_train)

lr_predictions_pca = lr_model_pca.predict(X_test_pca)

lr_accuracy_pca = accuracy_score(y_test, lr_predictions_pca)

print(f"Logistic Regression Accuracy (with PCA): {lr_accuracy_pca}")

rf_model_pca = RandomForestClassifier(random_state=42)

rf_model_pca.fit(X_train_pca, y_train)

rf_predictions_pca = rf_model_pca.predict(X_test_pca)

rf_accuracy_pca = accuracy_score(y_test, rf_predictions_pca)

print(f"Random Forest Accuracy (with PCA): {rf_accuracy_pca}")

models_pca = {

"SVM": svm_accuracy_pca,
```

```
"Logistic Regression": lr_accuracy_pca,

"Random Forest": rf_accuracy_pca

    }

best_model_pca = max(models_pca, key=models_pca.get)

print(f"\nBest Model (with PCA): {best_model_pca} with accuracy
{models_pca[best_model_pca]}")
```