

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**

“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT
on**

Machine Learning (23CS6PCMAL)

Submitted by

Shilpa k m (1BM23CS419)

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 **Shilpa k m (1BM23CS419)**, 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. Sunayana S Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE
---	---

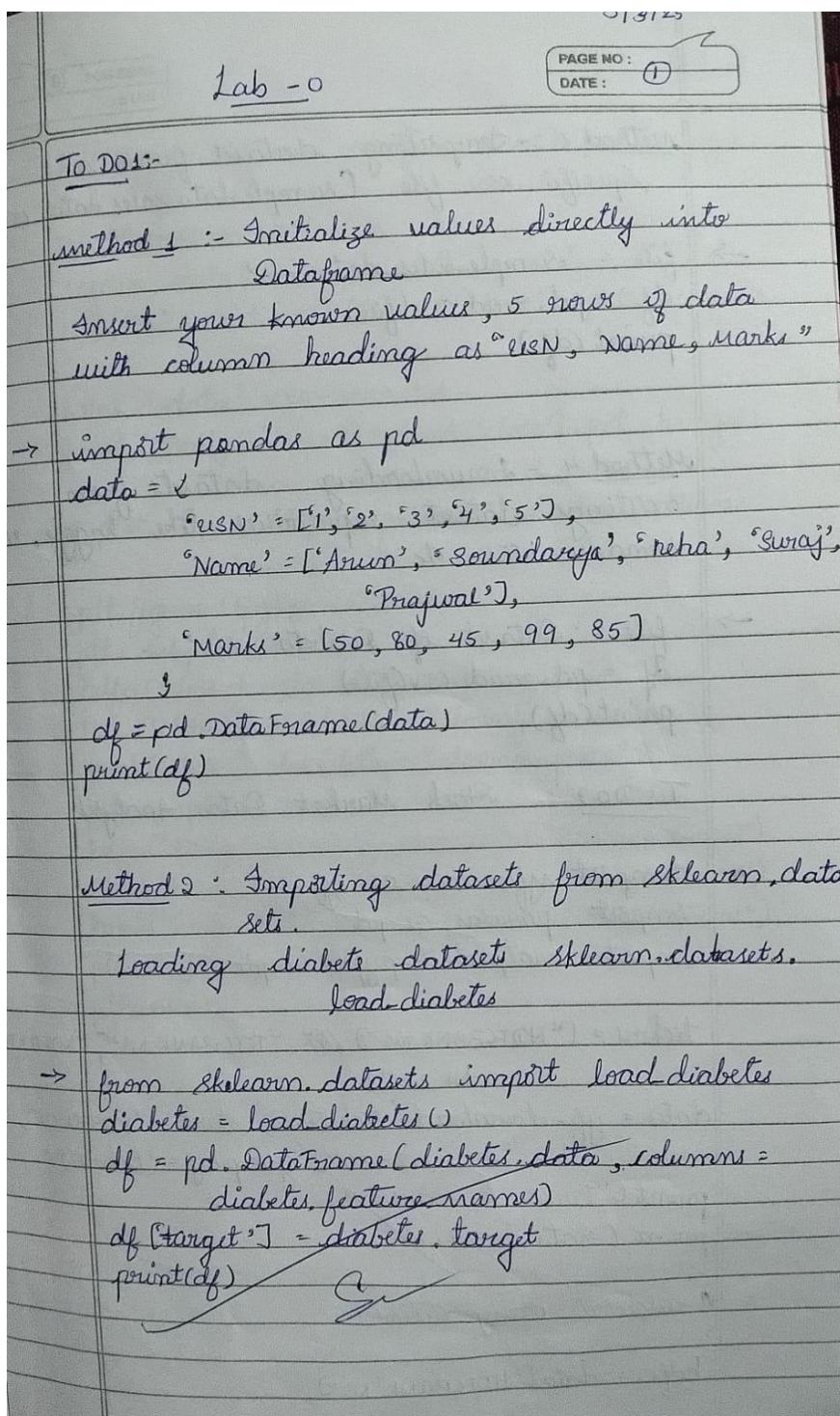
Index

Sl. No.	Date	Experiment Title	Page No.
1	21-2-2025	Write a python program to import and export data using Pandas library functions	4-7
2	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	8-10
3	10-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	11-16
4	17-3-2025	Build Logistic Regression Model for a given dataset	17-21
5	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample	22-24
6	7-4-2025	Build KNN Classification model for a given dataset	25-28
7	21-4-2025	Build Support vector machine model for a given dataset	29-33
8	5-5-2025	Implement Random forest ensemble method on a given dataset	34-36
9	5-5-2025	Implement Boosting ensemble method on a given dataset	37-39
10	12-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file	40-42
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method	43-46

Github Link: <https://github.com/SHILPA-45/ML.git>

Program 1

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



Method 3 :- Importing dataset from a specific .csv file (sample sales data.csv)

```
→ file = 'sample sales data.csv'
df = pd.read_csv(file)
print(df)
```

Method 4 :- Downloading datasets from existing dataset repositories like kaggle, UCI, Mendelv, KFEL etc

```
→ file = 'Dataset of Diabetes.csv'
df = pd.read_csv(file)
print(df)
```

To Do :- Stock Market Data Analysis

- Import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

```
tickers = ["HDFCBANK.NS", "ITC.NS", "KOTAKBA-NK.NS"]
```

```
data = yf.download(tickers, start = "2024-01-01",
end = "2024-12-31", groupby = 'tickers')
print("First 5 rows : ")
print(data.head())
```

calculate daily returns

```
hdfc = data["HDFCBANK.NS"]
```

Code:

```
import pandas as pd
data={
    'USN':['1','2','3','4','5'],
    'Name':['Arun', 'Soudarya','neha', 'suraj', 'Prajju'],
    'marks':[50, 80, 45, 99, 87]
}
df=pd.DataFrame(data)
print(df)

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

file_path='/sample_sales_data (1).csv'
df=pd.read_csv(file_path)
print("Sample data:")
print(df)

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

# Step 1: Import required libraries
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

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

# Fetch historical data for the last 1 year
data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')

# Display the first 5 rows of the dataset
print("First 5 rows of the dataset:")
print(data.head())
```

```

# Calculate daily returns
hdfc=data['HDFCBANK.NS']
hdfc['Daily Return'] = hdfc['Close'].pct_change()
print("\nSummary statistics for HDFC:")
print(hdfc.describe())

icic=data['ICICIBANK.NS']
icic['Daily Return'] = icic['Close'].pct_change()
print("\nSummary statistics for ICICI:")
print(icic.describe())

kotak=data['KOTAKBANK.NS']
kotak['Daily Return'] = kotak['Close'].pct_change()
print("\nSummary statistics for Kotak:")
print(kotak.describe())


# Plot the closing price and daily returns
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
hdfc['Close'].plot(title=" Closing Price")
icic['Close'].plot(title=" Closing Price")
kotak['Close'].plot(title=" Closing Price")
plt.subplot(2, 1, 2)
hdfc['Daily Return'].plot(title="Daily Returns" )
icic['Daily Return'].plot(title=" Daily Returns")
kotak['Daily Return'].plot(title="Daily Returns")
plt.tight_layout()
plt.show()

```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Lab - 1

PAGE NO : 4
DATE :

To do 1 :- Housing.csv

- 1 # 1 To load .csv file into a Dataframe
`df = pd.read_csv("housing.csv")`
- 2 Display information of all columns
`print("Information of all columns")`
`print(df.info())`
3. Display statistical information of all numerical
`print(df.describe())`
4. To display the count of unique labels for "ocean proximity" column
`print(df['ocean proximity'].value_counts())`
5. To display which attributes (columns) in dataset have missing values count greater than zero.
`missing values = df.isnull().sum()`
`column missing val = missing values[missing values > 0]`
`print(column missing val)`

Data

Diabetes dataset :-

1. which columns in the dataset had missing value? How did you handle them?

- None of column had missing values but if had then numerical columns NAN can be replaced with median & categorical columns null can be replaced with mode
2. Which categorical columns did you identify in the dataset? How did you encode them
→ The diabetes dataset had gender, & class as categorical columns & adult dataset had workclass, education, marital-status, occupation relationship, race, gender, native-country & income as categorical column using ordinal encoder we can encode categorical columns
3. What is the difference between min-Max scaling and standardization? When would you one over the other?
→ Min-Max scaling, also called normalization transforms data to fit within a specific range (usually 0 to 1) by scaling based on min-max values in dataset.
Standardization set scales to data by subtracting mean and dividing.

Code:

```
#Handling Missing Values, Handling categorical data, Handling Outliers
import pandas as pd
df=pd.read_csv('/content/Dataset of Diabetes .csv')
missing_values=df.isnull().sum()
print(missing_values[missing_values > 0])

# Data Transformations: Min-max Scaler/Normalization , Standard Scaler
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, StandardScaler
file_path = "Dataset of Diabetes .csv"
df = pd.read_csv(file_path)
df["Gender"] = df["Gender"].str.upper()
ordinal_encoder = OrdinalEncoder(categories=[["M", "F"]])
df["Gender_Encoded"] = ordinal_encoder.fit_transform(df[["Gender"]])
onehot_encoder = OneHotEncoder(sparse_output=False, drop="first")
encoded_class = onehot_encoder.fit_transform(df[["CLASS"]])
encoded_class_df = pd.DataFrame(encoded_class,
columns=onehot_encoder.get_feature_names_out(["CLASS"]))
df_encoded = pd.concat([df, encoded_class_df], axis=1)
df_encoded.drop(["Gender", "CLASS"], axis=1, inplace=True)
num_cols = ["AGE", "Urea", "Cr", "HbA1c", "Chol", "TG", "HDL", "LDL", "VLDL", "BMI"]
scaler = StandardScaler()
df_encoded[num_cols] = scaler.fit_transform(df_encoded[num_cols])
print(df_encoded.head())
df_encoded.to_csv("cleaned_diabetes_dataset.csv", index=False)
df_encoded_copy1=df_encoded
df_encoded_copy2=df_encoded
df_encoded_copy3=df_encoded
Q1 = df_encoded_copy1['BMI'].quantile(0.25)
Q3 = df_encoded_copy1['BMI'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_encoded_copy1['BMI'] = np.where(df_encoded_copy1['BMI'] > upper_bound, upper_bound,
np.where(df_encoded_copy1['BMI'] < lower_bound, lower_bound,
df_encoded_copy1['BMI']))

print(df_encoded_copy1.head())
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

19/3/25

Lab - 3

PAGE NO : 9
DATE :

Logistic Regression

Consider a binary classification problem where we want to predict whether a student will pass or fail based on]

Linear Regression

1) Apply linear regression technique to predict the 5th week sales

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.DataFrame({'x': x, 'y': y2})
```

$x \cdot y = []$

$x2 = []$

```
for i, j in zip(x, y):
    xy.append(i * j)
for i in x:
    x2.append(i ** 2)
```

$x_mean = \sum(x) / \text{len}(x)$

$y_mean = \sum(y) / \text{len}(y)$

$xy_mean = \sum(xy) / \text{len}(xy)$

$x2_mean = \sum(x2) / \text{len}(x2)$

$a_1 = (xy_mean - x_mean * y_mean) / (x2_mean - x_mean * x2)$

$a_0 = y_mean - a_1 * x_mean$

print(f"{}a_0 = {}a_0, a_1 = {}a_1{})")

`x_input = int(input("Enter the value of x:"))`

`y_pred = a0 + a1 * x_input`

point (`"predicted y for x = x_input is "`)
`Ly_inputs.append(y_pred)`)

`plt.scatter(x, y, color = 'blue', label = 'datapoint')`

`plt.plot([a0 + a1 * xi for xi in x], color = 'red',
label = 'Regression line')`

`plt.xlabel('x')`

`plt.ylabel('y')`

`plt.title('Linear Regression')`

`plt.legend()`

`plt.grid(True)`

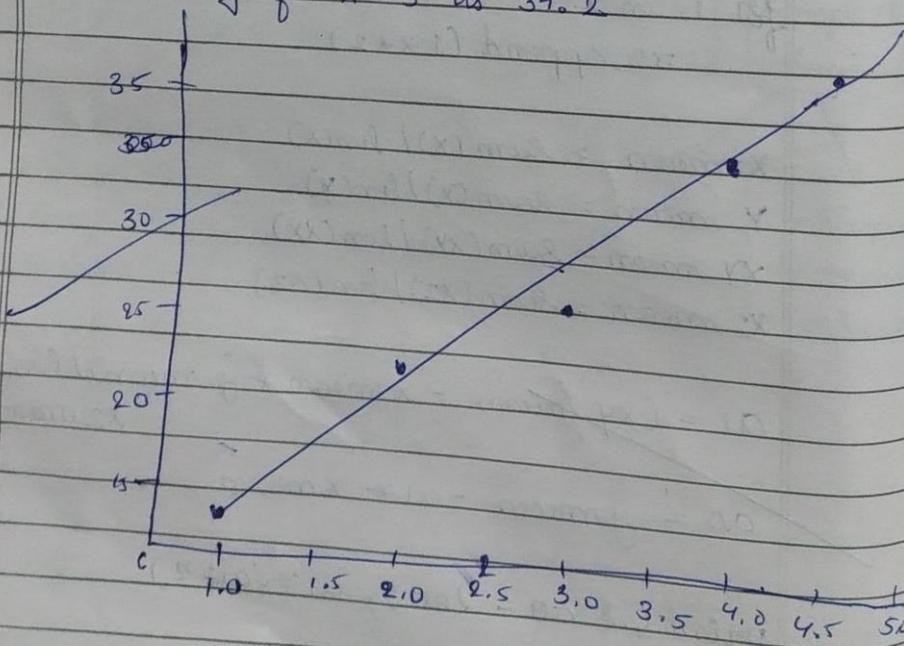
`plt.show()`

Output :-

$$a_0 = 6.1999 \quad a_1 = 5.6000$$

Enter the value of $x : 5$

Predicted y for $x = 5$ is 34.2



Q) Find linear regression of data of week & product sales. use matrix approach for finding linear regression

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

```
def matrix(x, y, n):
    ax = np.ones((n, 2))
    [ax[:, 1]] = x
    ay = np.array(y).reshape(-1, 1)
```

```
xt = ax.T
A = np.dot(xt, ax)
detA = np.linalg.det(A)
```

```
if detA != 0:
    Ainv = np.linalg.inv(A)
    a = np.dot(Ainv, np.dot(xt, ay))
    return a.flatten()
else:
```

return "Inverse doesn't exist"

```
n = int(input("Enter the number of observations"))
x = [int(input("Enter the value of x [{}]: ".format(i))) for i in range(n)]
y = [int(input("Enter the value of y [{}]: ".format(i))) for i in range(n)}
result = matrix_operation(x, y, n)
if isinstance(result, str):
    print(result)
```

else :

$a_0, a_1 = \text{result}$

print(f" Slope (a1) = {a1}, Intercept (a0) = {a0})

plt.scatter(x, y, color='blue', label='Data Points')

plt.plot(x, a0 + a1 * np.array(x), color='red',
label='Regression line')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Linear regression')

plt.legend()

plt.grid(True)

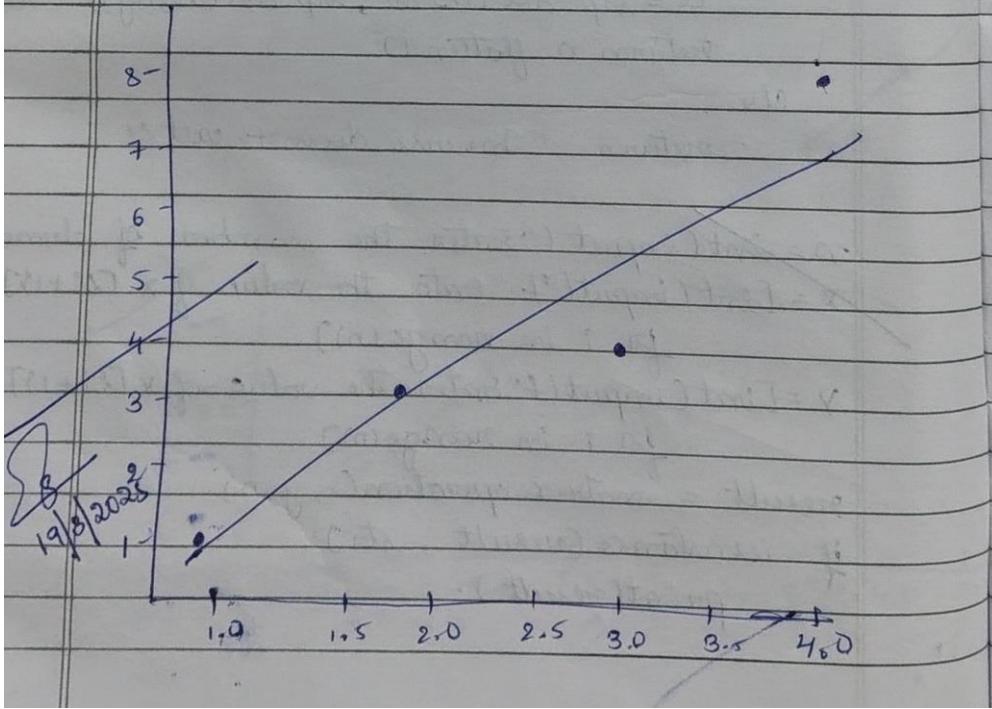
plt.show()

Output :- Enter the number of element : 4

Enter the values of x : [1, 2, 3, 4]

Enter the values of y : [1, 3, 4, 8]

Slope (a1) = 2.20 , Intercept (a0) = -1.5



Code:

```
import pandas as pd
import matplotlib.pyplot as plt

# Data
x = [1, 2, 3, 4, 5]
y = [12, 18, 22, 28, 35]
df = pd.DataFrame({'x': x, 'y': y})

# Calculate required values
xy = []
x2 = []
for i, j in zip(x, y):
    xy.append(i * j)
for i in x:
    x2.append(i ** 2)

x_mean = sum(x) / len(x)
y_mean = sum(y) / len(y)
xy_mean = sum(xy) / len(xy)
x2_mean = sum(x2) / len(x2)

# Calculate coefficients a0 and a1
a1 = (xy_mean - x_mean * y_mean) / (x2_mean - x_mean ** 2)
a0 = y_mean - a1 * x_mean

# Output the coefficients
print(f"a0 = {a0}, a1 = {a1}")

# Predict y for a given x value
x_input = int(input("Enter the value of x: "))
y_pred = a0 + a1 * x_input
print(f"Predicted y for x = {x_input} is: {y_pred}")

# Plotting
plt.scatter(x, y, color='blue', label='Data Points') # Scatter plot for the data points
plt.plot(x, [a0 + a1 * xi for xi in x], color='red', label='Regression Line') # Regression line
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

Code:

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

def matrix_operations(x, y, n):
    ax = np.ones((n, 2))
    ax[:, 1] = x
    ay = np.array(y).reshape(-1, 1)

    xt = ax.T
    A = np.dot(xt, ax)
    det_A = np.linalg.det(A)

    if det_A != 0:
        A_inv = np.linalg.inv(A)
        a = np.dot(A_inv, np.dot(xt, ay))
        return a.flatten() # Returns [a0, a1]
    else:
        return "Inverse doesn't exist"

# Input
n = int(input("Enter the number of elements: "))
x = [int(input(f"Enter the value of x[{i+1}]: ")) for i in range(n)]
y = [int(input(f"Enter the value of y[{i+1}]: ")) for i in range(n)]

# Perform matrix operation
result = matrix_operations(x, y, n)
if isinstance(result, str):
    print(result)
else:
    a0, a1 = result
    print(f"Slope (a1) = {a1}, Intercept (a0) = {a0}")

# Plotting
plt.scatter(x, y, color='blue', label='Data Points')
plt.plot(x, a0 + a1 * np.array(x), color='red', label='Regression Line')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

Program 4

Build Logistic Regression Model for a given dataset

Lab - 4

02/04/25
PAGE NO :
DATE :

Logistic regression

Q 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, & the learned parameters are $a_0 = -5$ (intercept) & $a_1 = 0.8$ [co-efficient for study hours]

@ Write the logistic regression equation for this problem.

$$\rightarrow P(y=1|x) = \frac{1}{1+e^{(-5+0.8x)}}$$

⑤ Calculate the probability for 7 hours of study

$$x = 7$$
$$P(y=1|7) = \frac{1}{1+e^{(-5+0.8(7))}}$$
$$P(y=1|7) = \frac{1}{1+e^{(-5+5.6)}}$$
$$P(y=1|7) = \frac{1}{1+e^{-0.6}}$$
$$P(y=1|7) = \frac{1}{1+e^{-0.6}}$$
$$e^{-0.6} \approx 0.5488$$

$$P(y=1|z) = \frac{1}{1+0.5488} = \frac{1}{1.5488}$$

$$P(y=1|z) \approx 0.6457$$

hence, the probability that student who studies for z hours will pass is
 $0.6457 \approx 64.57\%$.

- ② Determine the predicted class (pass/fail) for this student based on threshold of 0.5

→ if $P(y=1|z) \geq 0.5$ (or $P(y=0|z) \leq 0.5$) is predict class is pass

if $P(y=1|z) \leq 0.5$ is predict class is fail

so $P(y=1|z) \approx 0.6457$ is greater than 0.5

∴ Predicted class is "Pass"

- ③ Consider $z = [2, 1, 0]$ for 3 classes. Apply softmax funtn to find probability values

$$P(y=i|z) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad (\text{where } z_1=2, z_2=1, z_3=0)$$

$$e^{z_1} = e^2 \approx 7.389$$

$$e^{z_2} = e^1 \approx 2.718$$

$$e^{z_3} = e^0 \approx 1$$

$$\sum_j e^{z_j} = e^2 + e^1 + e^0 = 7.389 + 2.718 + 1 = 11.097$$

$$P(y=1|z) = \frac{e^{z_1}}{\sum_j e^{z_j}} = \frac{7.389}{11.107} \approx 0.665$$

$$P(y=2|z) = \frac{e^{z_2}}{\sum_j e^{z_j}} = \frac{0.718}{11.107} \approx 0.065$$

$$P(y=3|z) = \frac{e^{z_3}}{\sum_j e^{z_j}} = \frac{1}{11.107} \approx 0.090$$

vector $z = [2, 1, 0]$

$$P(y=1) \approx 0.665$$

$$\cancel{P(y=2) \approx 0.065}$$

$$\cancel{P(y=3) \approx 0.090}$$

$$u = 8.12$$

$$v = 0.16$$

$$w = 2.04$$

\rightarrow mean u &

\rightarrow std. dev. v .

\rightarrow variance w

\rightarrow each of these having a

standard deviation of 1

and all words are with 0

and 1 as minimum value

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Load dataset
file_path = "/content/zoo-data.csv"
df = pd.read_csv(file_path)
# Drop the animal_name column as it is not a feature
df = df.drop(columns=["animal_name"])
# Define features and target
X = df.drop(columns=["class_type"])
y = df["class_type"]
# 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)
# Train multinomial logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=500)
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
labels = np.unique(y)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
import pandas as pd
from matplotlib import pyplot as plt
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# Load HR dataset
```

```

df = pd.read_csv("/content/HR_comma_sep.csv")
df.head()

plt.scatter(df.satisfaction_level, df.left, marker='+', color='red')

# Splitting dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[['satisfaction_level']], df.left, train_size=0.9, random_state=10)
X_train.shape
X_test

# Training logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
X_test
y_test
y_predicted = model.predict(X_test)
y_predicted
model.score(X_test, y_test)
model.predict_proba(X_test)
y_predicted = model.predict([[0.5]])

# model.coef_ indicates value of m in y=m*x + b equation
model.coef_

# model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

# Define sigmoid function and do the math manually
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

def prediction_function(satisfaction_level):
    z = model.coef_[0][0] * satisfaction_level + model.intercept_[0]
    y = sigmoid(z)
    return y
satisfaction_level = 0.35
probability = prediction_function(satisfaction_level)
print(f"Probability of leaving: {probability:.2f}")

if probability >= 0.5:
    print("The employee will leave.")
else:
    print("The employee will stay.")

```

Program 5

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

Lab - 02

PAGE NO : 5
DATE :

Implement ID3 [Iterative Dichotomiser 3] algorithm on Tennis weather database

```
import pandas as pd
data = pd.read_csv('tennis.csv')
data

from sklearn.preprocessing import
LabelEncoder
outlook = LabelEncoder()
temp = LabelEncoder()
humidity = LabelEncoder()
wind = LabelEncoder()
play = LabelEncoder()
data['outlook'] = outlook.fit_transform(data['outlook'])
data['temp'] = outlook.fit_transform(data['temp'])
data['humidity'] = outlook.fit_transform(data['humidity'])
data['wind'] = outlook.fit_transform(data['wind'])
data['play'] = outlook.fit_transform(data['play'])
data

features cols = ['outlook', 'temp', 'humidity', 'wind']
X = data[features cols]
Y = data.play
print(X)
print(Y)
print(y)

from sklearn.model_selection import train_test
Split
x_train, x_test, y_train, y_test = train_test_split
(x, y, test_size = 0.2)
```

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy')
classifier.fit(x_train, y_train)
classifier.predict(x_test)
x_test
y_test
classifier.score(x_test, y_test)
```

```
from sklearn import tree
tree.plot_tree(classifier)
clf = DecisionTreeClassifier(criterion = 'entropy')
clf.fit(x, y)
importances = clf.feature_importances_
features_importance_df = pd.DataFrame({
    'Feature': x.columns,
    'Information Gain': importances
})
feature_importance_df = feature_importance_df.sort_values(
    by = 'Information Gain', ascending = False)
print(feature_importance_df)
```

Output :-

features	Information gain
0 outlook	0.369629
3 windy	0.244909
2 humidity	0.211237
1 temp	0.151929

Code:

```
import pandas as pd
data=pd.read_csv('tennis.csv')
data
from sklearn.preprocessing import LabelEncoder
outlook=LabelEncoder()
temp=LabelEncoder()
humidity=LabelEncoder()
wind=LabelEncoder()
play=LabelEncoder()
data['outlook']=outlook.fit_transform(data['outlook'])
data['temp']=outlook.fit_transform(data['temp'])
data['humidity']=outlook.fit_transform(data['humidity'])
data['windy']=outlook.fit_transform(data['windy'])
data['play']=outlook.fit_transform(data['play'])
data
features_cols=['outlook','temp','humidity','windy']
x=data[features_cols]
y=data.play
print(x)
print()
print(y)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train,y_test=train_test_split(x,y,test_size=0.2)
from sklearn.tree import DecisionTreeClassifier
classifier=DecisionTreeClassifier(criterion='entropy')
classifier.fit(x_train,y_train)
classifier.predict(x_test)
x_test
y_test
classifier.score(x_test,y_test)
from sklearn import tree
tree.plot_tree(classifier)
clf = DecisionTreeClassifier(criterion='entropy') # Using entropy to calculate information gain
clf.fit(x, y)
importances = clf.feature_importances_
# Create a DataFrame to see the feature importances
feature_importance_df = pd.DataFrame({
    'Feature': x.columns,
    'Information Gain': importances
})
# Sort by importance
feature_importance_df = feature_importance_df.sort_values(by='Information Gain', ascending=False)
print(feature_importance_df)
```

Program 6

Build KNN Classification model for a given dataset

Lab - 06

214125
PAGE NO.:
DATE:

KNN classification

• Consider the following dataset for $k=3$ & test data $(x, 35, 100)$ as (person, Age, salary) value using knn classifier model & predict the target

Person	x	y	Target	distance	distance
A	18	50	N	$\sqrt{(35-18)^2 + (100-50)^2} = 52.8$	
B	23	55	N	$\sqrt{(35-23)^2 + (100-55)^2} = 46.6$	
C	24	70	N	$\sqrt{(35-24)^2 + (100-70)^2} = 31.9$	
D	41	60	Y	$\sqrt{(35-41)^2 + (100-60)^2} = 40.5$	
E	43	70	Y	$\sqrt{(35-43)^2 + (100-70)^2} = 31.1$	
F	38	40	Y	$\sqrt{(35-38)^2 + (100-40)^2} = 60.1$	
X	35	100	?		

$31.3 = N$
 $31.9 = Y$
 $40.5 = Y$

1 N from C
 2 Y from E & D

The majority is Y

~~The predicted target for $x = (35, 100)$ is Y [par]~~

- For this data set
- ⑨ How to choose the k value? Demonstrate using accuracy rate & error rate.

Steps:

- 1) Split the data into training & testing sets
- 2) Train the model with various values of k
- 3) Evaluate accuracy & error rate for each k
- 4) Plot accuracy vs k & error rate vs k and select k with the highest accuracy or lowest error rate

$$\text{accuracy} = \frac{\text{No. of correct predictions}}{\text{Total prediction}}$$

$$\text{Error rate} = 1 - \text{Accuracy}$$

• For diabetes data set

What is the purpose of feature scaling?
How to perform it?

Purpose: KNN uses ~~diabetes~~ distance, so features with larger values dominate. Scaling ensures fairness.

Perform: Standardization (Z-score normalization):
subtract mean, divide by standard deviation.

24/2025

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Load dataset
file_path = "/content/diabetes.csv"
df = pd.read_csv(file_path)
# Define features and target
X = df.drop(columns=["Outcome"])
y = df["Outcome"]
# 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)
# Feature scaling (Standardization)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Initialize and train KNN classifier with k=5
k = 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
# Make predictions
y_pred = knn.predict(X_test)
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
conf_matrix = confusion_matrix(y_test, y_pred)
labels = ["Non-Diabetic", "Diabetic"]
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

```

from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
df = pd.read_csv('/content/heart.csv')
# Define features and target
X = df.drop(columns=['target']) # Assuming 'target' is the classification column
y = df['target']
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Find the best K value
k_values = range(1, 21)
accuracy_scores = []
for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy_scores.append(accuracy_score(y_test, y_pred))
best_k = k_values[np.argmax(accuracy_scores)]
print(f'Best K value: {best_k}')
# Train model with best K
best_model = KNeighborsClassifier(n_neighbors=best_k)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy with best K ({best_k}): {accuracy:.4f}')
print("Classification Report:")
print(classification_report(y_test, y_pred))
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix - KNN (K={best_k})')
plt.show()
# Plot K values vs. Accuracy
plt.plot(k_values, accuracy_scores, marker='o')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.title('K Value vs Accuracy')
plt.show()

```

Program 7

Build Support vector machine model for a given dataset

Lab - 06

PAGE NO :
DATE :

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

class SVM:
    def __init__(self, learning_rate=0.001, lambda=0,
                 n_iters=1000):
        self.b = learning_rate
        self.lambda = lambda
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, x, y):
        y = np.where(y <= 0, -1, 1)
        n_samples, n_features = x.shape
        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(x):
                condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1
                if condition:
                    self.w -= self.b * (2 * self.lambda * self.w)
                else:
                    self.w -= self.b * (2 * self.lambda * self.w -
                                         np.dot(x_i, y[idx])) + y[idx]
                    self.b += y[idx]

    def predict(self, x):
        return np.sign(np.dot(x, self.w) + self.b)
```

```
def predict(self, x):
    approx = np.dot(x, self.w) + self.b
    return np.sign(approx)
```

```
def visualization(self, X, Y, new_point=None,
                  prediction=None):
```

```
def view_get_hyperplane(x, w, b, offset):
    return (-w[0] * x + b + offset) / w[1]
```

```
for i, sample in enumerate(X):
```

```
if Y[i] == 1:
```

```
plt.scatter(sample[0], sample[1], marker='o',
            color='blue', label='class +1' if i == 0 else '')
```

```
x0 = np.linspace(np.min(X[:, 0]) - 1, np.max(X[:, 0]) + 1, 100)
```

```
x1 = get_hyperplane(x0, self.w, self.b, 0)
```

```
ax.plot(x0, x1, 'k-', label='Decision')
```

```
ax.plot(x0, x1_m, 'k--', label='margins')
```

```
ax.plot(x0, x1_p, 'k--')
```

```
if __name__ == "main":
```

```
X = np.array([
```

```
[1, 7]
```

```
[2, 8]
```

```
[3, 8]
```

```
[8, 1]
```

```
[9, 2]
```

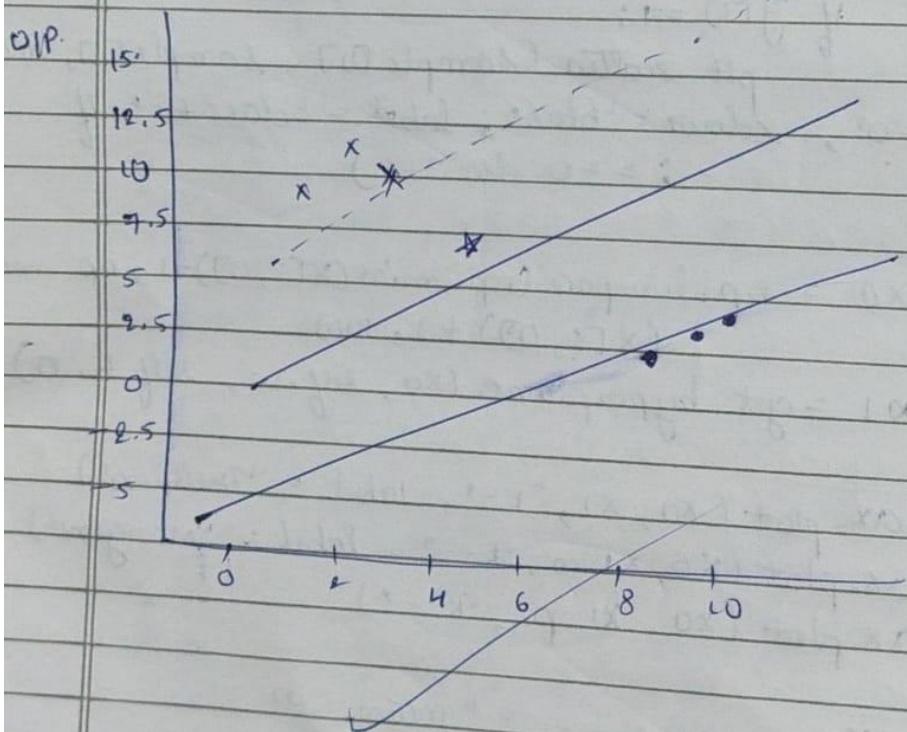
```
[10, 2]
```

$y = \text{np.array}([0, 0, 0, 1, 1, 1])$

$\text{new_point} = \text{np.array}([5, 5])$

SVM, visualize (x, y , new_new_point = new_point
prediction = prediction)

point ($+$ "New point" new_point) & classify
as : $\begin{cases} \text{'class 1'} & \text{if prediction} == 1 \\ \text{'class 0'} & \end{cases}$



Code:

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

class SVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, X, y):
        y = np.where(y <= 0, -1, 1) # Convert labels to -1 and 1
        n_samples, n_features = X.shape
        self.w = np.zeros(n_features)
        self.b = 0
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y[idx]))
                    self.b += self.lr * y[idx]
        def predict(self, X):
            approx = np.dot(X, self.w) + self.b
            return np.sign(approx)

    def visualize(self, X, y, new_point=None, prediction=None):
        def get_hyperplane(x, w, b, offset):
            return (-w[0] * x + b + offset) / w[1]
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)
        for i, sample in enumerate(X):
            if y[i] == 1:
                plt.scatter(sample[0], sample[1], marker='o', color='blue', label='Class +1' if i == 0 else "")
            else:
                plt.scatter(sample[0], sample[1], marker='x', color='red', label='Class -1' if i == 0 else "")

        # Plot decision boundary
        x0 = np.linspace(np.min(X[:, 0])-1, np.max(X[:, 0])+1, 100)
        x1 = get_hyperplane(x0, self.w, self.b, 0)
        x1_m = get_hyperplane(x0, self.w, self.b, -1)
```

```

x1_p = get_hyperplane(x0, self.w, self.b, 1)
ax.plot(x0, x1, 'k-', label='Decision Boundary')
ax.plot(x0, x1_m, 'k--', label='Margins')
ax.plot(x0, x1_p, 'k--')

if new_point is not None:
    color = 'green' if prediction == 1 else 'orange'
    label = f'New Point: Class {"1" if prediction == 1 else "0"}'
    plt.scatter(new_point[0], new_point[1], c=color, s=100, edgecolors='black', label=label, marker='*')
    ax.legend()
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title("SVM with New Point Prediction")
    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    # Training data
    X = np.array([
        [1, 7],
        [2, 8],
        [3, 8],
        [8, 1],
        [9, 2],
        [10, 2]
    ])
    y = np.array([0, 0, 0, 1, 1, 1]) # 0 -> -1, 1 -> +1

    # New point to classify
    new_point = np.array([[5, 5]])

    # Train and predict
    svm = SVM()
    svm.fit(X, y)
    prediction = svm.predict(new_point)[0]

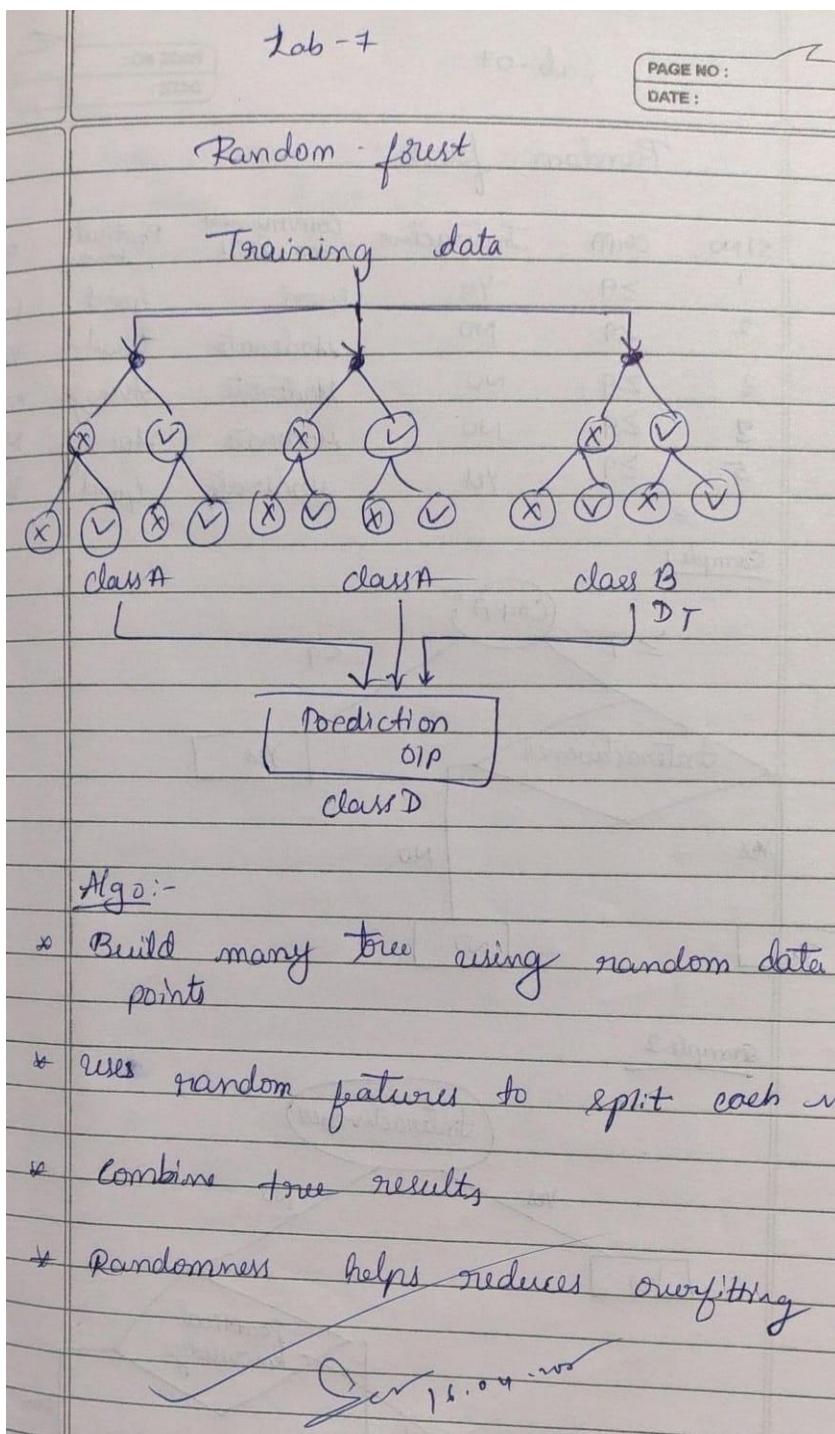
    # Visualize
    svm.visualize(X, y, new_point=new_point[0], prediction=prediction)

    # Print prediction
    print(f"New point {new_point[0]} classified as: {'Class 1' if prediction == 1 else 'Class 0'}")

```

Program 8

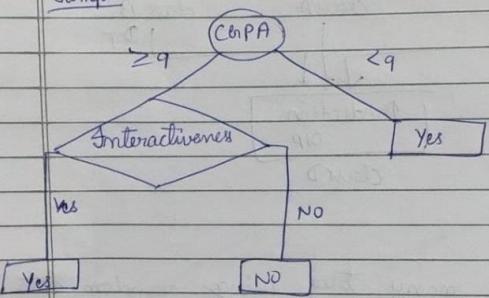
Implement Random forest ensemble method on a given dataset



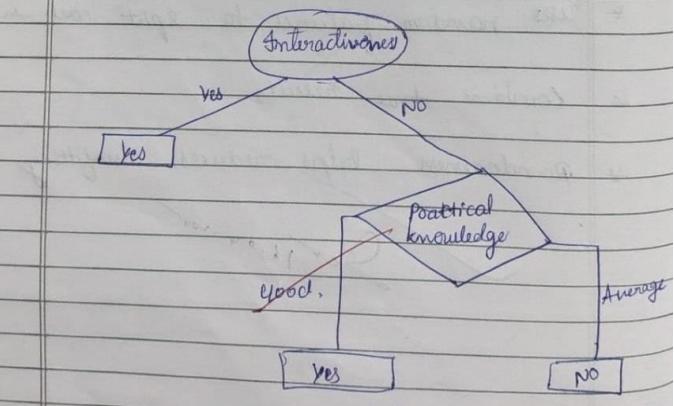
Random forest

S1 NO.	CePA	Interactive	Communication Skill	Poetical knowl	Job type
1	≥ 9	Yes	Good	Good	Yes
2	< 9	No	Moderate	Good	Yes
3	≥ 9	No	Moderate	Average	No
4	≥ 9	No	Moderate	Average	No
5	≥ 9	Yes	Moderate	Good	Yes

Sample 1



Sample 2



Best accuracy : 1.0

number of Trees : 1

Confusion matrix

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

Code:

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

# Load the iris dataset
iris = load_iris()

# Convert to DataFrame
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target

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

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

best_score = 0
best_n = 0
best_cm = None

# Try different numbers of trees
for n in range(1, 101):
    clf = RandomForestClassifier(n_estimators=n, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    if acc > best_score:
        best_score = acc
        best_n = n
        best_cm = confusion_matrix(y_test, y_pred)

print(f"Best Accuracy: {best_score}")
print(f"Number of Trees: {best_n}")
print("Confusion Matrix:")
print(best_cm)
```

Program 9

Implement Boosting ensemble method on a given dataset

Lab - 08																																							
PAGE NO : DATE :																																							
<u>AdaBoost</u>																																							
<table border="1"> <thead> <tr> <th>C6PA</th> <th>Interviewer</th> <th>Practical knowledge</th> <th>Communication Skill</th> <th>Job profile</th> </tr> </thead> <tbody> <tr> <td>≥ 9</td> <td>Yes</td> <td>Good</td> <td>Good</td> <td>Yes</td> </tr> <tr> <td>< 9</td> <td>No</td> <td>Good</td> <td>Moderate</td> <td>Yes</td> </tr> <tr> <td>≥ 9</td> <td>No</td> <td>Average</td> <td>Moderate</td> <td>No</td> </tr> <tr> <td>< 9</td> <td>No</td> <td>Average</td> <td>Good</td> <td>No</td> </tr> <tr> <td>≥ 9</td> <td>Yes</td> <td>Good</td> <td>Moderate</td> <td>Yes</td> </tr> <tr> <td>≥ 9</td> <td>Yes</td> <td>Good</td> <td>Moderate</td> <td>Yes</td> </tr> </tbody> </table>					C6PA	Interviewer	Practical knowledge	Communication Skill	Job profile	≥ 9	Yes	Good	Good	Yes	< 9	No	Good	Moderate	Yes	≥ 9	No	Average	Moderate	No	< 9	No	Average	Good	No	≥ 9	Yes	Good	Moderate	Yes	≥ 9	Yes	Good	Moderate	Yes
C6PA	Interviewer	Practical knowledge	Communication Skill	Job profile																																			
≥ 9	Yes	Good	Good	Yes																																			
< 9	No	Good	Moderate	Yes																																			
≥ 9	No	Average	Moderate	No																																			
< 9	No	Average	Good	No																																			
≥ 9	Yes	Good	Moderate	Yes																																			
≥ 9	Yes	Good	Moderate	Yes																																			
<u>2 wrong</u>																																							
<table border="1"> <thead> <tr> <th>C6PA</th> <th>Predicted job offer</th> <th>Job profile</th> </tr> </thead> <tbody> <tr> <td>≥ 9</td> <td>Yes</td> <td>Yes</td> </tr> <tr> <td>< 9</td> <td>No</td> <td>Yes</td> </tr> <tr> <td>≥ 9</td> <td>Yes</td> <td>No</td> </tr> <tr> <td>< 9</td> <td>No</td> <td>No</td> </tr> <tr> <td>≥ 9</td> <td>Yes</td> <td>Yes</td> </tr> <tr> <td>≥ 9</td> <td>Yes</td> <td>Yes</td> </tr> </tbody> </table>					C6PA	Predicted job offer	Job profile	≥ 9	Yes	Yes	< 9	No	Yes	≥ 9	Yes	No	< 9	No	No	≥ 9	Yes	Yes	≥ 9	Yes	Yes														
C6PA	Predicted job offer	Job profile																																					
≥ 9	Yes	Yes																																					
< 9	No	Yes																																					
≥ 9	Yes	No																																					
< 9	No	No																																					
≥ 9	Yes	Yes																																					
≥ 9	Yes	Yes																																					
<u>2 wrong prediction</u>																																							
<u>4 correct prediction</u>																																							
(i)	$\epsilon_{C6PA} = \frac{2}{6} = \frac{1}{3} = 0.33$																																						
(ii)	$\begin{aligned} \Delta \epsilon_{C6PA} &= \frac{1}{2} \ln \left(\frac{1 - \epsilon_{C6PA}}{\epsilon_{C6PA}} \right) \\ &= \frac{1}{2} \ln \left(\frac{1 - 0.33}{0.33} \right) \\ &= 0.347 \end{aligned}$																																						

$$\begin{aligned}
 \text{(ii) } Z_{\text{true}} &= \text{wt}[\text{correct instance}] \times [\text{exp of correct classifier}] \\
 &\quad + \text{wt}[\text{incorrect instance}] \times [\text{exp of incorrect classifier}] \\
 &= \frac{1}{6} \times 4 \times e^{-0.347} + \frac{1}{6} \times 2 \times e^{+0.347} \\
 &= 0.9428 \quad [\text{normalize factor}]
 \end{aligned}$$

$$\begin{aligned}
 \text{(iv) } \text{wt}(d_j)_{\text{ini}} &= \frac{\text{wt}(d_j) \text{ CgPA of correct instance} \times e^{-\text{dcgpa}}}{Z_{\text{CgPA}}} \\
 &= \frac{\frac{1}{6} \times e^{-0.347}}{0.9428} \\
 &= 0.1249 \quad [\text{correct weight}]
 \end{aligned}$$

v. w^{new} for incorrect instance

$$\begin{aligned}
 &= \frac{\frac{1}{6} \times e^{+0.347}}{0.9428} \\
 &= 0.2501
 \end{aligned}$$

CgPA	Initial weight	updated weight
>9	$\frac{1}{6}$	0.1249
<9	$\frac{1}{6}$	0.2501
>9	$\frac{1}{6}$	0.1249
<9	$\frac{1}{6}$	0.2501
>9	$\frac{1}{6}$	0.1249
>9	$\frac{1}{6}$	0.1249

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
# Load dataset
df = pd.read_csv("/content/income.csv")
# Define features and target
X = df.drop("income_level", axis=1)
y = df["income_level"]
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 1: AdaBoost with 10 estimators
model = AdaBoostClassifier(n_estimators=10, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
base_accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print(f"Accuracy with 10 estimators: {base_accuracy:.4f}")
print("Confusion Matrix:\n", conf_matrix)
# Step 2: Fine-tuning n_estimators
estimator_range = range(10, 201, 10)
scores = []
for n in estimator_range:
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    scores.append(acc)
    print(f"n_estimators={n}, Accuracy={acc:.4f}")
# Plot accuracy vs. number of estimators
plt.plot(estimator_range, scores, marker='o')
plt.title("Accuracy vs. Number of Estimators (AdaBoost)")
plt.xlabel("Number of Estimators")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
# Best score and configuration
best_n = estimator_range[np.argmax(scores)]
best_score = max(scores)
print(f"Best accuracy: {best_score:.4f} achieved with {best_n} estimators.")
```

Program 10

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

		PAGE NO :	DATE :	
Lab-09				
<u>k-means</u>				
cluster centers are $(1.0, 1.0)$ & $(5.0, 7.0)$				
execute two iteration				
Record no	A	B		
R ₁	1.0	1.0		
R ₂	1.5	2.0		
R ₃	3.0	4.0		
R ₄	5.0	7.0		
R ₅	3.5	5.0		
R ₆	4.5	5.0		
R ₇	3.5	4.5		
Distance = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$				
Iteration 1				
Record	Point(A,B)	dist C ₁ (1,1)	dist C ₂ (5,7)	Assign cluster
R ₁	(1,1)	0	7.21	C ₁
R ₂	(1.5,2.0)	1.12	6.10	C ₁
R ₃	(3,4)	3.61	4.24	C ₁
R ₄	(5,7)	7.21	0	C ₂
R ₅	(3.5,5)	4.72	2.50	C ₂
R ₆	(4.5,5)	5.32	2.24	C ₂
R ₇	(3.5,4.5)	4.30	2.92	C ₂
$C_1 = \frac{(1+1.5+3)}{3}, \frac{(1+2+4)}{3}$				
$= \frac{5.5}{3}, \frac{7.0}{3}$				
$= 1.83, 2.33$				
$C_2 = \frac{(5+3.5+4.5+3.5)}{4}, \frac{(7+5+5+4.5)}{4}$				
$= \left(\frac{16.5}{4}, \frac{21.5}{4} \right)$				
$= 4.12, 5.37$				

Iteration 2

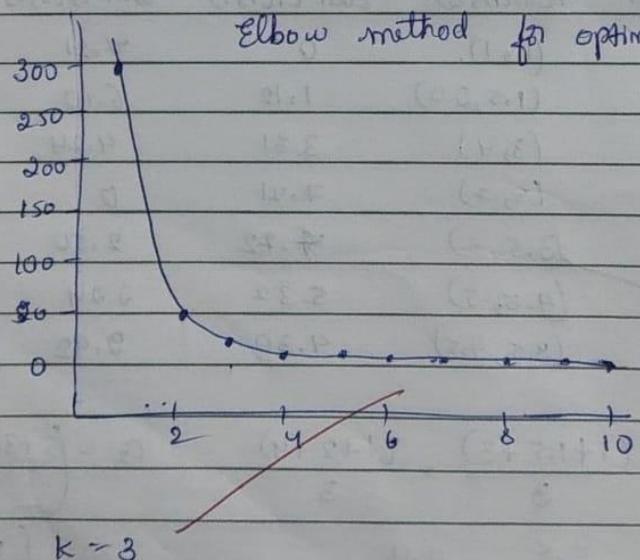
Record no	$C_1(1+83, 2.33)$	$C_2(4.12, 5.37)$	duster
$R_1(1, 1)$	1.57	5.37	C1
$R_2(1.5, 2)$	0.47	4.27	C1
$R_3(3, 4)$	2.04	1.77	C2
$R_4(5, 7)$	5.64	1.85	C2
$R_5(3.5, 5)$	3.14	0.72	C2
$R_6(4.5, 5)$	3.78	0.53	C2
$R_7(3.5, 4.5)$	2.74	1.04	C2

$$C_1 = \frac{(1+1.5)}{2} \quad \frac{(1+2)}{2} \quad C_2 = \frac{(3+5+3.5+4.5+3.5)}{5},$$

$$C_1 = 1.25, 1.5 \quad C_2 = \frac{(4+7+5+5+4.5)}{5}$$

$$C_2 = 3.9, 5.1$$

Elbow method for optimal k



Code:

```

import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
df = pd.read_csv("/content/iris (6).csv") # Replace with actual file path if needed
# Use only petal_length and petal_width
X = df[['petal_length', 'petal_width']]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Elbow method to find optimal k
wcss = []
k_values = range(1, 11)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
plt.figure(figsize=(8, 5))
plt.plot(k_values, wcss, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('WCSS')
plt.grid(True)
plt.tight_layout()
plt.show()
# Apply K-Means with optimal k = 3
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_scaled)
# Add cluster labels to original dataframe
df['cluster'] = clusters
plt.figure(figsize=(8, 5))
for cluster in range(3):
    cluster_points = X_scaled[clusters == cluster]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {cluster}')
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
                s=200, c='black', marker='X', label='Centroids')
plt.title('K-Means Clustering (k=3)')
plt.xlabel('Petal Length (scaled)')
plt.ylabel('Petal Width (scaled)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Lab - 10

PAGE NO :
DATE :

PCA

Reduce dimension from 2 to 1

feature	eg 1	eg 2	eg 3	eg 4
xc ₁	4	8	13	7
xc ₂	11	4	5	11

Eigen values $\lambda_1 = 30.38$ $\lambda_2 = 6.61$

Eigen Vectors $e_1 = \begin{bmatrix} 0.5594 \\ -0.8303 \end{bmatrix}$ $e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

→ 1) data : $\begin{bmatrix} 4 & 8 & 13 & 7 \\ 11 & 4 & 5 & 11 \end{bmatrix}$

2) mean center of data

mean 1 = $\frac{4+8+13+7}{4} = 8$

mean 2 = $\frac{11+4+5+11}{4} = 6.3$

$x_{\text{cluster}} : \begin{bmatrix} 4-8 & 8-8 & 13-8 & 7-8 \\ 11-8.3 & 4-8.3 & 5-8.3 & 11-8.3 \end{bmatrix} = \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.3 & 3.5 & 5.5 \end{bmatrix}$

3) using projecting data into first principle component $z = e_1^T x_{\text{centered}}$

$z_1 = (0.5594)(-4) + (0.8303)(-1) = 4.305$

$z_2 = 3.7363$

$$z_3 = 5.6986$$

$$z_4 = -5.124$$

$$\therefore Z [-4.305, 3.73, 5.69, -5.12]$$

→ Model accuracy

	Before	Before	After
logistic	0.8833		0.8667

	Before	After
SVM	0.8478	0.8556

	Before	After
Random forest	0.8989	0.8722

~~87.5%
71.5%
22%~~

Code:

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

# Load dataset
df = pd.read_csv("/content/heart (1).csv") # Use actual path if needed

# Label encode binary categorical columns
le = LabelEncoder()
df["Sex"] = le.fit_transform(df["Sex"]) # M/F -> 1/0
df["ExerciseAngina"] = le.fit_transform(df["ExerciseAngina"]) # Y/N -> 1/0

# One-hot encode categorical columns with >2 categories
df = pd.get_dummies(df, columns=["ChestPainType", "RestingECG", "ST_Slope"],
drop_first=True)

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

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

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

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

# Train and evaluate models (before PCA)
print("==== Accuracy Before PCA ====")
for name, model in models.items():
```

```

model.fit(X_train, y_train)
acc = accuracy_score(y_test, model.predict(X_test))
print(f'{name}: {acc:.4f}')

# Apply PCA to reduce dimensions to 2
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Split again for PCA-transformed data
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2,
random_state=42)

# Train and evaluate models (after PCA)
print("\n==== Accuracy After PCA (2 components) ====")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    acc = accuracy_score(y_test, model.predict(X_test_pca))
    print(f'{name}: {acc:.4f}')

```