Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that Mr./~~Miss~~ Om Anirudha Shete

of Computer Department, Semester VII with

Roll No. 2103163 has completed a course of the necessary

experiments in the subject Machine Learning under my

supervision in the **Thadomal Shahani Engineering College**

Laboratory in the year 2024 - 2025

Teacher In-Charge

Head of the Department

Date ~~15~~ 15/10/2024

Principal

CONTENTS

15/10/24

Om Shale
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE
Group 2

Experiment No: 1

2/03/23

Aim: Select appropriate dataset and perform data preprocessing steps:

i) Imputation
ii) Anomaly detection
iii) Standardization
iv) Normalization
v) Encoding

Theory:

i) **Imputation**

⇒ Imputation is a process of replacing missing or incomplete data with substituted values.

- Missing data can arise from various issues such as data corruption or human error.
- Handling missing data correctly is crucial to avoid biases and inaccuracies in ML model.
- Common imputation techniques involves:

a) **Mean/Median Imputation**

⇒ Replace missing values with mean or median of the column. This is suitable for numeric data.

b) **Forward/Backward Fill**

⇒ Use the previous or next observation to fill missing values, which is useful for time-series

ii) Anomaly detection

⇒ Anomaly detection is the process of identifying unusual patterns that do not conform to expected behavior, often referred to as outliers.
- Anomalies can indicate errors, rare events or important insights.

iii) Standardization

⇒ Standardization involve rescaling the feature so that they have a mean of 0 and a standard deviation of 1.
- This is essential for algorithms that assumes normally distributed data or that are sensitive to the scale of the features.
- The formula of standardization is:-

$$\text{Standardized Value} = \frac{x - \text{mean}(x)}{\text{std}(x)}$$

iv) Normalization

⇒ Normalization scales the data to a fixed range, typically [0, 1] or [-1, 1].
- It is especially useful for algorithms that requires bounded input values.

- The Min-Max Normalization formula is:

$$\text{Normalized Value} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

v) Encoding

⇒ Encoding converts the categorical data into numerical format so that machine learning algorithm can process it.

- Common encoding techniques are :-

a) Label Encoding

⇒ Convert each unique category into a unique integer.

b) Ordinal Encoding

⇒ Used for categorical features that have a natural order, such as "low", "medium", "High".

- Each category is assigned an integer based on its order.

27/9/24

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.ensemble import IsolationForest
```

```
# Load the Titanic dataset
data = pd.read_csv('iris.csv')

# Display the first few rows of the dataset
print(data.head())
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

```
imputer = SimpleImputer(strategy='mean')

# Perform imputation on numeric columns (excluding 'Id' and 'Species')
data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']] = imputer.fit_transform(
    data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
)
data.replace({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}, inplace=True)

# Check for any missing values after imputation
print("Missing values after imputation:")
print(data.isnull().sum())
```

```
Missing values after imputation:
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

```
iso_forest = IsolationForest(contamination=0.1, random_state=42)
data['anomaly'] = iso_forest.fit_predict(data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']])

# Identify anomalies
anomalies = data[data['anomaly'] == -1]
print("Detected anomalies:")
print(anomalies)
```

```
Detected anomalies:
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species  \
13    14            4.3           3.0            1.1           0.1        0
14    15            5.8           4.0            1.2           0.2        0
15    16            5.7           4.4            1.5           0.4        0
22    23            4.6           3.6            1.0           0.2        0
32    33            5.2           4.1            1.5           0.1        0
41    42            4.5           2.3            1.3           0.3        0
60    61            5.0           2.0            3.5           1.0        1
62    63            6.0           2.2            4.0           1.0        1
105  106            7.6           3.0            6.6           2.1        2
109  110            7.2           3.6            6.1           2.5        2
117  118            7.7           3.8            6.7           2.2        2
118  119            7.7           2.6            6.9           2.3        2
122  123            7.7           2.8            6.7           2.0        2
131  132            7.9           3.8            6.4           2.0        2
135  136            7.7           3.0            6.1           2.3        2

     anomaly
13        -1
14        -1
15        -1
22        -1
32        -1
41        -1
60        -1
62        -1
105       -1
109       -1
117       -1
118       -1
122       -1
```

```
131       -1
135       -1
```

```
# Step 4: Standardization
scaler = StandardScaler()

# Standardize the features (excluding 'Id' and the 'anomaly' column)
data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']] = scaler.fit_transform(
    data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
)

# Display the standardized data
print("Standardized data:")
print(data.head())
```

```
Standardized data:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species  \
0   1      -0.900681      1.032057      -1.341272     -1.312977        0
1   2      -1.143017     -0.124958      -1.341272     -1.312977        0
2   3      -1.385353      0.337848      -1.398138     -1.312977        0
3   4      -1.506521      0.106445      -1.284407     -1.312977        0
4   5      -1.021849      1.263460      -1.341272     -1.312977        0

   anomaly
0        1
1        1
2        1
3        1
4        1
```

```
# Step 5: Normalization
min_max_scaler = MinMaxScaler()

# Normalize the features (excluding 'Id' and the 'anomaly' column)
data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']] = min_max_scaler.fit_transform(
    data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
)

# Display the normalized data
print("Normalized data:")
print(data.head())
```

```
Normalized data:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species  \
0   1       0.222222      0.625000       0.067797      0.041667        0
1   2       0.166667      0.416667       0.067797      0.041667        0
2   3       0.111111      0.500000       0.050847      0.041667        0
3   4       0.083333      0.458333       0.084746      0.041667        0
4   5       0.194444      0.666667       0.067797      0.041667        0

   anomaly
0        1
1        1
2        1
3        1
4        1
```

```
# One-Hot Encoding for the 'Species' column
data = pd.get_dummies(data, columns=['Species'], drop_first=True)

# Display the dataset after One-Hot Encoding
print("Encoded dataset using One-Hot Encoding:")
print(data.head())
```

```
Encoded dataset using One-Hot Encoding:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  anomaly  \
0   1       0.222222      0.625000       0.067797      0.041667        1
1   2       0.166667      0.416667       0.067797      0.041667        1
2   3       0.111111      0.500000       0.050847      0.041667        1
3   4       0.083333      0.458333       0.084746      0.041667        1
4   5       0.194444      0.666667       0.067797      0.041667        1

   Species_1  Species_2
0      False      False
1      False      False
2      False      False
3      False      False
4      False      False
```

```
data.to_csv('p_iris.csv', index=False)
```

Experiment NO : 2

Aim : Implement Gradient descent algorithm to minimise given objective function.
For eg., $f(x_1, x_2) = x_1^3 + 6x_2^2$.

Theory :

- A gradient is nothing but derivative that defines the effects on outputs of the function with little bit of variations in Input.
- Gradient descent stands as a cornerstone orchestrating the intricate dance of model optimization.
- As its core, it is numerical optimization algorithm that aims to find the optimal parameters weights and biases of a neural networks by minimizing a defined cost function.
- It works by iteratively adjusting the weights or parameters of the model in the direction of negative gradient of the cost function until the minimum of the cost function is reached.
- The cost function evaluates between the difference between the actual and predicted outputs.
- Gradient descent is a fundamental optimization technique in ML used to minimise the cost or loss function during model

training.

i) It iteratively adjust model parameters by moving in the direction of the steepest decrease in the cost function.

ii) The algorithm calculates gradients, representing the partial derivative of the cost function concerning each parameter

The key steps are

i) Initialize the parameters a and b with some initial values.

ii) Compute the gradient of the function with each parameter.

iii) Update the parameters

$$a = a - \alpha \cdot \frac{\partial f}{\partial a}$$

$$b = b - \alpha \cdot \frac{\partial f}{\partial b}$$

Where,

α = learning rate.

iv) Repeat ~~step~~ (ii) and (iii) until convergence.

- For e.g, $f(x_1, x_2) = -x_1^3 + 6x_2^2$

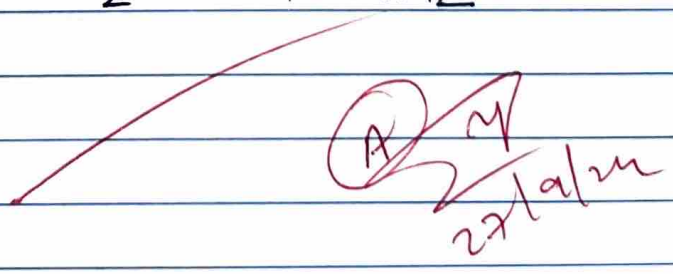
$$\frac{\partial f}{\partial x_1} = \frac{\partial(-x_1^3 + 6x_2^2)}{\partial x_1}$$
$$= -3x_1^2$$

$$\frac{\partial f}{\partial x_2} = \frac{\partial(-x_1^3 + 6x_2^2)}{\partial x_2}$$
$$= 12x_2$$

- Update the parameters:

$$x_1 = x_1 - \alpha(-3x_1^2)$$
$$x_2 = x_2 - \alpha(12x_2)$$

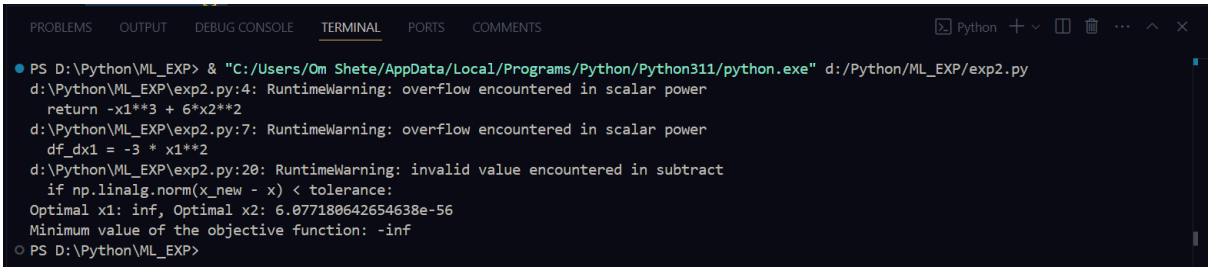
$$\therefore x_1 = x_1 + 3\alpha x_1^2$$
$$x_2 = x_2 - 12\alpha x_2$$



Code:

```
import numpy as np

def objective_function(x1, x2):
    return -x1**3 + 6*x2**2

def gradient(x1, x2):
    df_dx1 = -3 * x1**2
    df_dx2 = 12 * x2
    return np.array([df_dx1, df_dx2])

def gradient_descent(x_init, learning_rate=0.01, tolerance=1e-6,
max_iters=1000):
    x = np.array(x_init, dtype='float64')
    values = []

    for i in range(max_iters):
        grad = gradient(x[0], x[1])
        x_new = x - learning_rate * grad
        values.append(objective_function(x_new[0], x_new[1]))

        if np.linalg.norm(x_new - x) < tolerance:
            print(f'Converged in {i+1} iterations')
            break
        x = x_new

    return x, values

x_init = [2.0, 2.0]

optimal_x, values = gradient_descent(x_init, learning_rate=0.01)

print(f'Optimal x1: {optimal_x[0]}, Optimal x2: {optimal_x[1]}')

print(f'Minimum value of the objective function:
{objective_function(optimal_x[0], optimal_x[1])}')
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
COMMENTS
Python
PS D:\Python\ML_EXP> & "C:/Users/Om Shete/AppData/Local/Programs/Python/Python311/python.exe" d:/Python/ML_EXP/exp2.py
d:\Python\ML_EXP\exp2.py:4: RuntimeWarning: overflow encountered in scalar power
return -x1**3 + 6*x2**2
d:\Python\ML_EXP\exp2.py:7: RuntimeWarning: overflow encountered in scalar power
df_dx1 = -3 * x1**2
d:\Python\ML_EXP\exp2.py:20: RuntimeWarning: invalid value encountered in subtract
if np.linalg.norm(x_new - x) < tolerance:
Optimal x1: inf, Optimal x2: 6.07718064265463 8e-56
Minimum value of the objective function: -inf
PS D:\Python\ML_EXP>

Experiment No: 3

Aim: Implement simple linear regression using analytical and machine learning without using SkLearn.

Theory:

- A simple linear regression is a type of regression algorithms that models the relationship between a dependent variable and one independent variable.
- The relationship shown by simple Linear Regression model is Linear or a sloped straight line, hence it is called Simple Linear Regression.
- The key point in simple linear regression is that dependent variable must be continuous / real value.
- However, independent variable can be measured on continuous or categorical values.
- The regression line is represented using the following equation,

$$y' = aX + b + e$$

- In the above equation y' represents the predicted value, a represents the slope of the line, b shows the Y intercept and e is the random error.

- Here, we assume the mean value of random error is 0, so the equation it becomes,

$$Y' = aX + b$$

- The value of a and b can be calculated as -

$$a = \frac{n \cdot \Sigma XY - \Sigma X \cdot \Sigma Y}{n\Sigma X^2 - (\Sigma X)^2}$$

$$b = \frac{1}{n}(\Sigma Y - a\Sigma X)$$

- For e.g., consider the set of data as $\{(-1,-1), (2,2), (3,2)\}$

	X	Y	XY	X^2
	-1	-1	+1	1
	2	2	4	4
	3	2	6	9
Total	6	3	11	14

- The equation for the regression line is,

$$Y' = aX + b$$

$$\therefore a = \frac{n.\Sigma XY - \Sigma X.\Sigma Y}{n.\Sigma X^2 - (\Sigma X)^2}$$

$$= \frac{3 \times 11 - 4 \times 3}{3 \times 14 - 16}$$

$$\therefore a = 0.807$$

$$\therefore b = \frac{1}{n}(\Sigma Y - a \times \Sigma X)$$

$$= \frac{1}{3}(3 - 0.807 \times 4)$$

$$= -0.076$$

- Now, the equation for the line becomes,

$$Y' = 0.807X - 0.076$$

A
27/4/22

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

```
data = pd.read_csv('p_iris.csv')
```

```
x = data['SepalLengthCm'].values
y = data['SepalWidthCm'].values
```

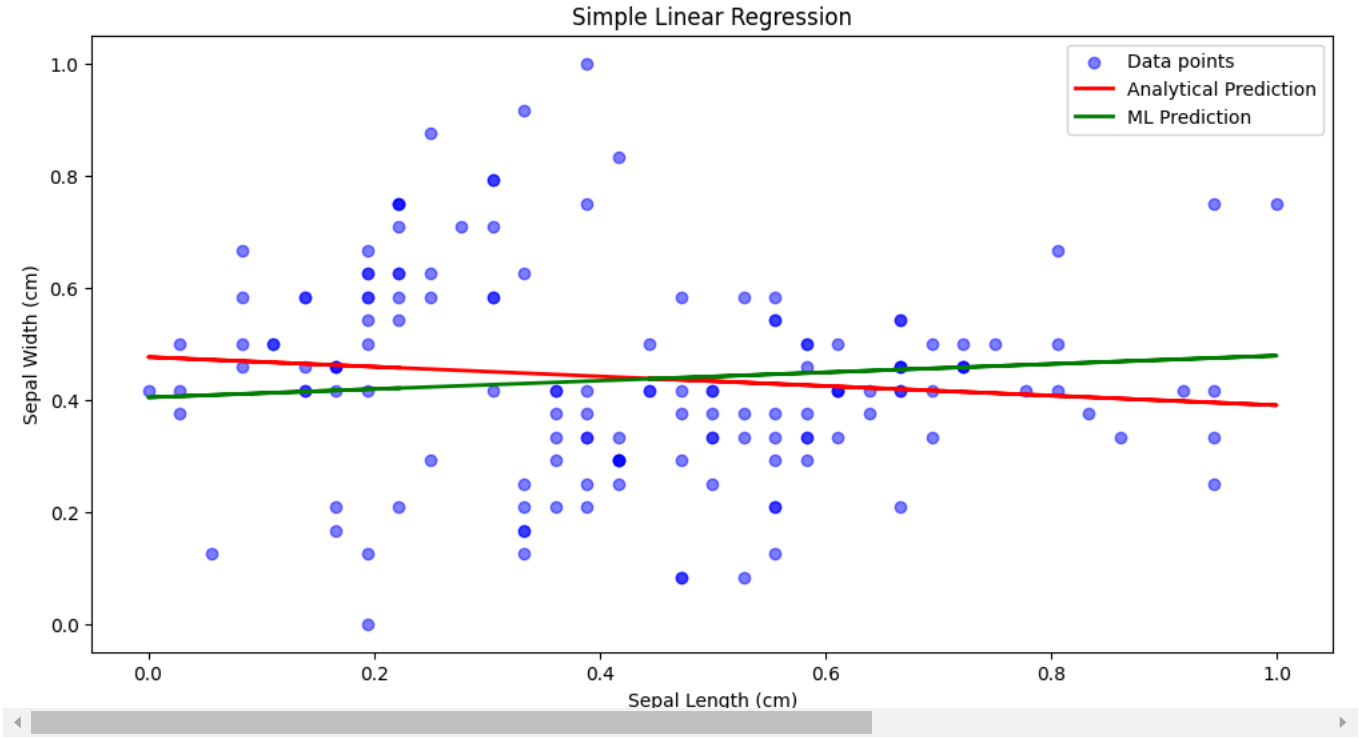
```
# 1. Analytical Method
# Calculate means
N = len(x)
mean_x = np.mean(x)
mean_y = np.mean(y)

# Calculate coefficients
beta_1 = (N * np.sum(x * y) - np.sum(x) * np.sum(y)) / (N * np.sum(x ** 2) - np.sum(x) ** 2)
beta_0 = mean_y - beta_1 * mean_x

print(f"Analytical Method: Intercept (β0) = {beta_0}, Slope (β1) = {beta_1}")

# Make predictions using the derived coefficients
y_pred_analytical = beta_0 + beta_1 * x
```

```
Analytical Method: Intercept (β0) = 0.47599332256678795, Slope (β1) = -0.08590235069574728
```

```
# 2. Machine Learning Method (Gradient Descent)
# Initialize parameters
alpha = 0.01  # Learning rate
iterations = 1000
m = len(x)  # Number of observations

# Initialize weights
theta_0 = 0  # Intercept
theta_1 = 0  # Slope

# Gradient Descent
for _ in range(iterations):
    # Predictions
    y_pred_ml = theta_0 + theta_1 * x
    # Calculate gradients
    error = y_pred_ml - y
    gradient_0 = (1/m) * np.sum(error)
    gradient_1 = (1/m) * np.sum(error * x)

    # Update weights
    theta_0 -= alpha * gradient_0
    theta_1 -= alpha * gradient_1

print(f"Machine Learning Method: Intercept (θ0) = {theta_0}, Slope (θ1) = {theta_1}")

# Make predictions using the ML method
y_pred_ml = theta_0 + theta_1 * x
```

```
Machine Learning Method: Intercept (θ0) = 0.40406666573537464, Slope (θ1) = 0.07447297472687159
```

```
# Plotting the results
plt.figure(figsize=(12, 6))
plt.scatter(x, y, color='blue', label='Data points', alpha=0.5)
plt.plot(x, y_pred_analytical, color='red', label='Analytical Prediction', linewidth=2)
plt.plot(x, y_pred_ml, color='green', label='ML Prediction', linewidth=2)
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```


Simple Linear Regression
Data points
Analytical Prediction
ML Prediction
Sepal Width (cm)
Sepal Length (cm)
0.0
0.2
0.4
0.6
0.8
1.0

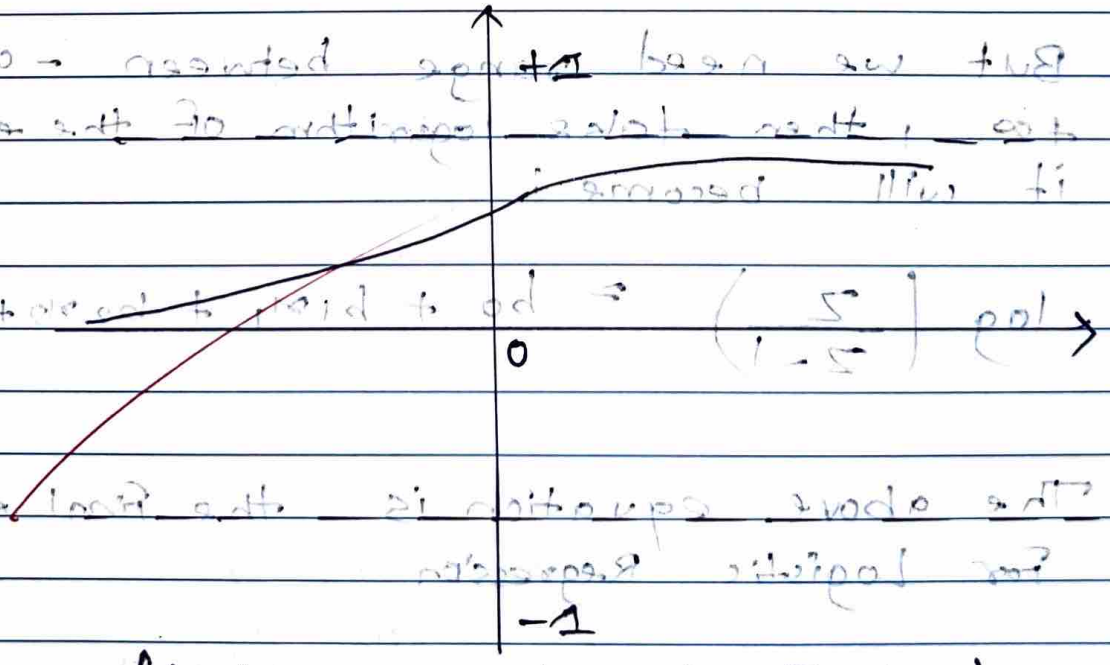
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment NO: 4

Aim: Implement logistic regression using ML method without using SKLearn.

Theory:

- Logistic regression is a supervised learning algorithm used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not.
- Logistic regression is a statistic algorithm which analyze the relationship between the two data factors.
- Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1.



(Logistic or sigmoid Function)

- The net input to the sigmoid function is,

$$Z = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$

- In the net input equation x represents the input data or the features.
- When we use optimized coefficients b, classifier will be successful.
- Optimized b can be calculated using the optimization technique/concepts.
- In logistic regression Z can be between 0 and 1 only, so for this let's divide the above equation by $(Z-1)$.

$\frac{Z}{Z-1}$ is 0 for $Z = 0$ and infinite for $Z = 1$

- But we need range between $-\infty$ to $+\infty$, then take logarithm of the equation it will become:

$$\log\left(\frac{Z}{Z-1}\right) = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$

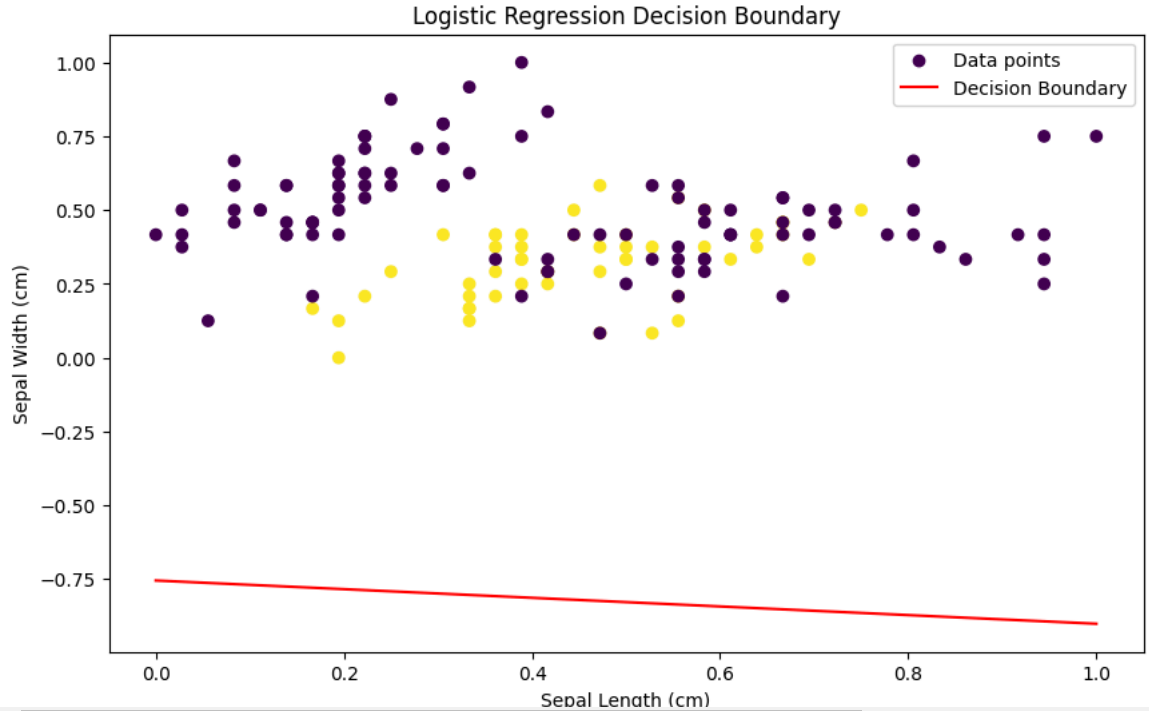
- The above equation is the final equation for Logistic Regression.

27/9/24

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

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

# Load the dataset
data = pd.read_csv('p_iris.csv')

# Extract features and target variable
# Assuming you want to predict Species_1 or Species_2
# Let's use 'Species_1' for binary classification
X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].values
y = data['Species_1'].astype(int).values  # Convert boolean to integer (0 or 1)

# Define the sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Define the cost function (binary cross-entropy)
def cost_function(y, y_pred):
    return -np.mean(y * np.log(y_pred + 1e-10) + (1 - y) * np.log(1 - y_pred + 1e-10))

# Implementing Logistic Regression using Gradient Descent
def logistic_regression(X, y, learning_rate=0.01, iterations=1000):
    m, n = X.shape
    weights = np.zeros(n)  # Initialize weights
    bias = 0  # Initialize bias

    for _ in range(iterations):
        # Linear combination
        linear_model = np.dot(X, weights) + bias
        # Apply sigmoid function
        y_pred = sigmoid(linear_model)

        # Compute gradients
        dw = (1/m) * np.dot(X.T, (y_pred - y))  # Gradient with respect to weights
        db = (1/m) * np.sum(y_pred - y)  # Gradient with respect to bias

        # Update weights and bias
        weights -= learning_rate * dw
        bias -= learning_rate * db

        # Optionally print the cost every 100 iterations
        if _ % 100 == 0:
            cost = cost_function(y, y_pred)
            print(f"Cost after iteration {_}: {cost}")

    return weights, bias

# Train the model
weights, bias = logistic_regression(X, y)

print(f"Trained weights: {weights}")
print(f"Trained bias: {bias}")

# Make predictions
def predict(X, weights, bias):
    linear_model = np.dot(X, weights) + bias
    y_pred = sigmoid(linear_model)
    return [1 if i >= 0.5 else 0 for i in y_pred]

# Get predictions
y_pred = predict(X, weights, bias)

# Plotting the results (only if you are using 2 features)
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', label='Data points')
x_values = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
y_values = -(weights[0] * x_values + bias) / weights[1]  # Decision boundary
plt.plot(x_values, y_values, color='red', label='Decision Boundary')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Logistic Regression Decision Boundary')
plt.legend()
plt.show()
```

```
Cost after iteration 0: 0.6931471803599453
Cost after iteration 100: 0.65922578838218
Cost after iteration 200: 0.6438176806245366
Cost after iteration 300: 0.6357462632668022
Cost after iteration 400: 0.6307088229397281
Cost after iteration 500: 0.627005773448819
Cost after iteration 600: 0.6239441545051513
Cost after iteration 700: 0.6212320977419205
Cost after iteration 800: 0.6187417998431392
Cost after iteration 900: 0.6164140176717633
Trained weights: [-0.08170087 -0.55821476  0.09061479 -0.00896527]
Trained bias: -0.4212070132659844
```



Om Shet

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment No: 5

Aim: Implement Decision tree classification algorithm

Theory:

- Decision trees are very strong and most suitable tools for classification and prediction.
- The attractiveness of decision tree is due to the fact that, in contrast to neural networks, decision tree represent rules.
- Rules are represented using linguistic variables so that user interpretability may be achieved.
- By comparing the records with the rules one can easily find a particular category to which the record belongs to.
- Decision tree method is mainly used for the tasks that possess the following properties:

i) The tasks or the problems in which the records are represented by attribute-value pairs.

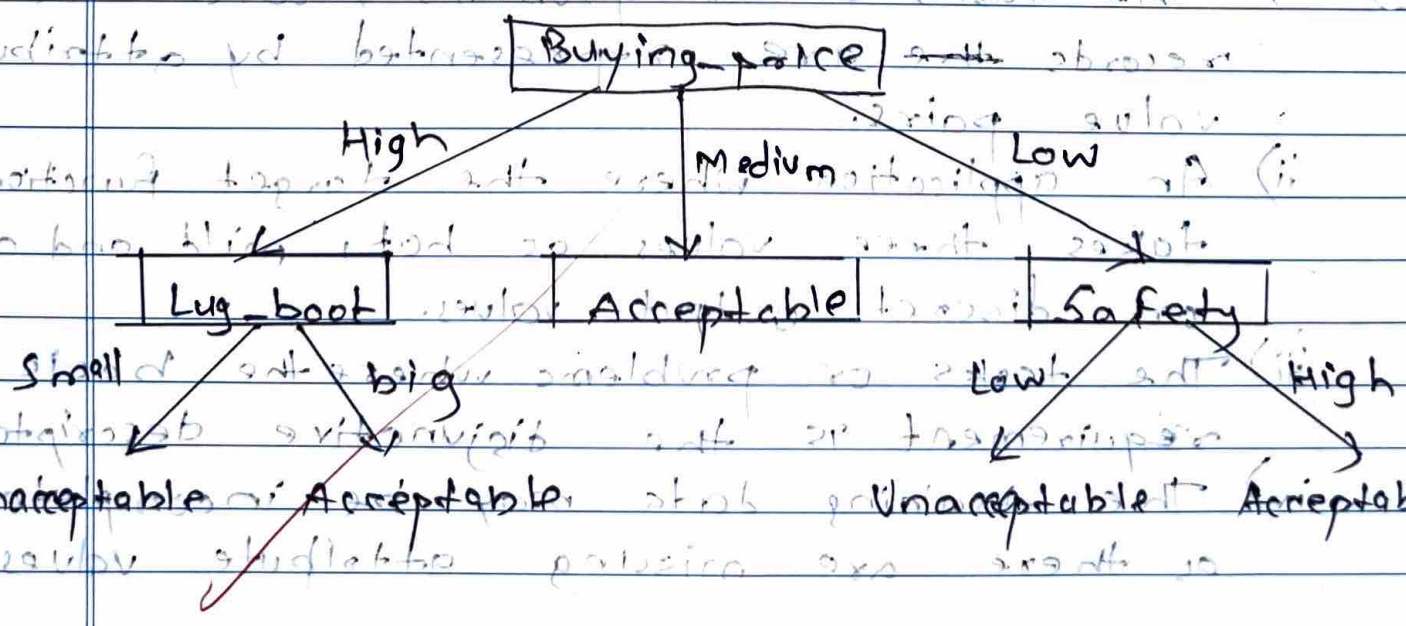
ii) An application where the target function takes three values as hot, mild and cold or discret output values.

iii) The tasks or problems where the basic requirement is the disjunctive descriptor.

iv) The training data may be incomplete as there are missing attribute values

Decision tree representation:

- Decision tree is a classifier which is represented in the form of a tree structure where each node is either a leaf node or decision node. In leaf node represents the value of target or responce attribute of examples.
- Decision node represents some test to be carried out on a single attribute value, with one branch and sub tree for each possible outcome of the test.
- Decision tree generates regression or classification models in the form of a tree structure.
- Decision tree divides a dataset into smaller subsets with increase in depth of tree. The final decision tree is a tree with decision nodes and leaf nodes.



- A decision node (e.g., buying_price) has two or more branches (e.g.; High, Medium & Low).
- Leaf node shows a classification or decision.
- The topmost decision node in a tree which represents the best predictor is called root node.
- Decision trees can be used to represent categorical as well as numerical data.

- **Gini Index:**

- All attributes are assumed to be continuous valued.
- It is assumed that there exist several possible split values for each attribute.
- Gini index method can be modified for categorical attributes.
- Gini Index is used in CART.
- IF a dataset T contains example from n classes, gini index gini(T) is defined as-

$$Gini(T) = 1 - \sum_{j=1}^{n} (P_j)^2$$

- In the above equation P_j represents the relative frequency of class j in T.

27/9/22

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

```
data = pd.read_csv('p_iris.csv')
```

```
X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].values
y = data['Species_1'].astype(int).values
```

```
def gini_impurity(y):
    if len(y) == 0:
        return 0
    prob = np.bincount(y) / len(y)
    return 1 - np.sum(prob ** 2)

# Define function to find the best split
def best_split(X, y):
    m, n = X.shape
    best_gain = 0
    best_split_value = None
    best_left_indices = None
    best_right_indices = None

    for feature_index in range(n):
        thresholds = np.unique(X[:, feature_index])
        for threshold in thresholds:
            left_indices = np.where(X[:, feature_index] <= threshold)[0]
            right_indices = np.where(X[:, feature_index] > threshold)[0]

            if len(left_indices) == 0 or len(right_indices) == 0:
                continue

            # Calculate Gini impurity for left and right
            left_impurity = gini_impurity(y[left_indices])
            right_impurity = gini_impurity(y[right_indices])

            # Calculate weighted average Gini impurity
            gain = gini_impurity(y) - (len(left_indices) / m * left_impurity + len(right_indices) / m * right_impurity)

            if gain > best_gain:
                best_gain = gain
                best_split_value = (feature_index, threshold)
                best_left_indices = left_indices
                best_right_indices = right_indices

    return best_split_value, best_left_indices, best_right_indices

# Define a Decision Tree Classifier
class DecisionTreeClassifier:
    def __init__(self, depth_limit=None):
        self.depth_limit = depth_limit
        self.tree = None

    def _build_tree(self, X, y, depth=0):
        if len(np.unique(y)) == 1 or (self.depth_limit and depth >= self.depth_limit):
            return np.bincount(y).argmax()

        split, left_indices, right_indices = best_split(X, y)
        if split is None:
            return np.bincount(y).argmax()

        left_tree = self._build_tree(X[left_indices], y[left_indices], depth + 1)
        right_tree = self._build_tree(X[right_indices], y[right_indices], depth + 1)

        return (split, left_tree, right_tree)

    def fit(self, X, y):
        self.tree = self._build_tree(X, y)

    def _predict_single(self, x, tree):
        if not isinstance(tree, tuple):
            return tree

        feature_index, threshold = tree[0]
        if x[feature_index] <= threshold:
            return self._predict_single(x, tree[1])
        else:
            return self._predict_single(x, tree[2])
```

```
    def predict(self, X):
        return np.array([self._predict_single(x, self.tree) for x in X])

# Train the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(depth_limit=5)  # Set depth limit for tree
dt_classifier.fit(X, y)

# Make predictions
y_pred = dt_classifier.predict(X)

# Evaluate accuracy
accuracy = np.mean(y_pred == y) * 100
print(f"Accuracy: {accuracy:.2f}")

# Display the predictions
print("Predictions:", y_pred)
```

```
Accuracy: 100.00
Predictions: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```

```
def plot_decision_boundaries(X, y, model):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
    plt.xlabel('Sepal Length (cm)')
    plt.ylabel('Sepal Width (cm)')
    plt.title('Decision Boundary of Decision Tree Classifier')
    plt.show()
```

```
plot_decision_boundaries(X, y, dt_classifier)
```

Om Shete

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment NO: 6

Aim: Implement Support Vector Machines for non-linear classification.

Theory:

- **Support Vectors**

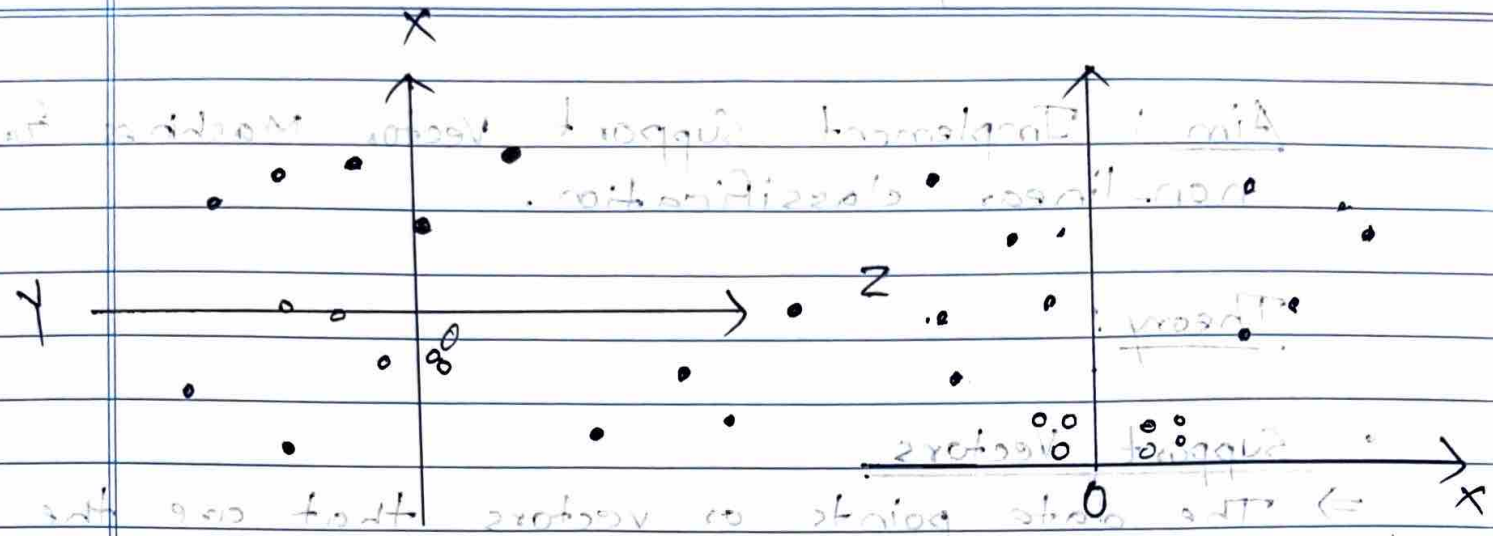
⇒ The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as support vectors.

- Since these vectors support the hyperplane, hence called a support vector.

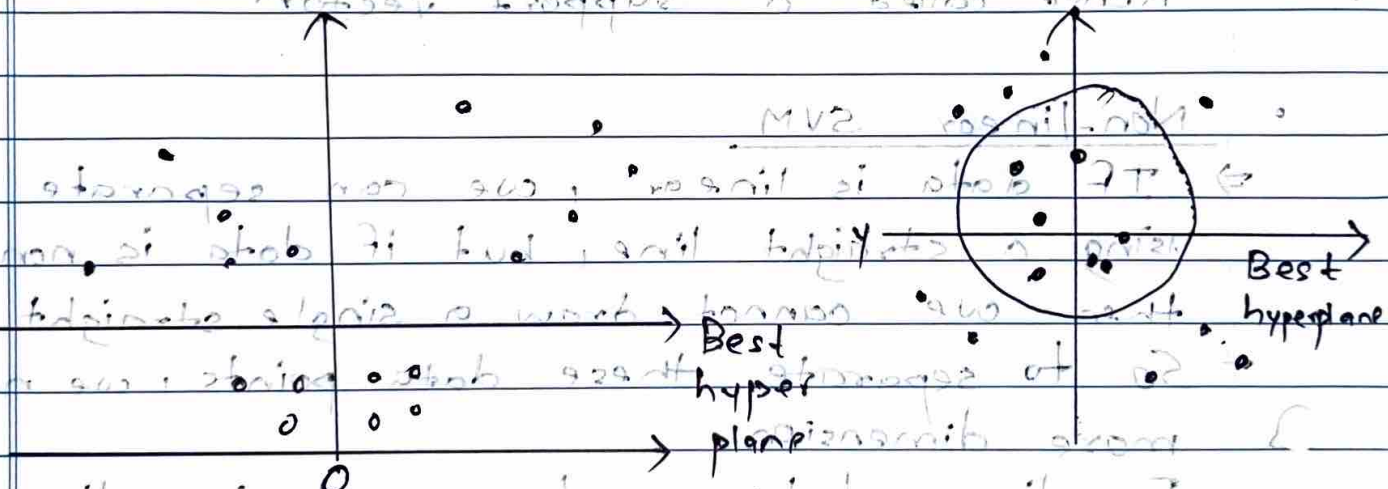
- **Non-linear SVM**

⇒ If data is linear, we can separate it by using a straight line, but if data is non-linear, then we cannot draw a single straight line.

- So to separate these data points, we need one more dimension.
- For linear data, we have used two dimensions x and y, so for non-linear data, we add a third dimension Z.
- It can be calculated as (say), $Z = x^2 + y^2$.
- After adding the third dimensions, the sample space becomes:



- Now, SVM will decide the datapoints into classes in the following way.



- If we put $z_i = 1$, then it will becomes $x_i^2 + y_i^2 = 1$,
- we get circumference of radius 1 as hyperplane surface in case of non-linear data.

A+
27/9/22

```
import pandas as pd
from sklearn.impute import SimpleImputer
from                                    _split
from
from                                    er, MinMaxScaler
from                                    ort, accuracy_score, confusion_matrix
from                                    r
from
impor
impor
impor
```



```
# Loa
data =

data.h
```

	LengthCm	PetalWidthCm	anomaly	Species_1	Species_2
	0.067797	0.041667	1	False	False
	0.067797	0.041667	1	False	False
	0.050847	0.041667	1	False	False
	0.084746	0.041667	1	False	False
	0.067797	0.041667	1	False	False

```
# 2. Split the dataset into features (X) and target (y)
X = data.iloc[:, :2].values
y = data.iloc[:, -1].values
```

```
# Encode target labels (species) to numeric values
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

```
# 3. Split the dataset into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)
```

```
# 4. Initialize the SVM model with a non-linear kernel (RBF)
svm_model = SVC(kernel='rbf', gamma='auto')
```

```
# 5. Train the SVM model
svm_model.fit(X_train, y_train)
```

```
SVC
SVC(gamma='auto')
```

```
y_pred = svm_model.predict(X_test)
```

```
# Accuracy
accuracy = accuracy_score(y_test, y_pred) * 100
print(f"Accuracy: {accuracy:.4f}\n")
```

```
Accuracy: 100.0000
```

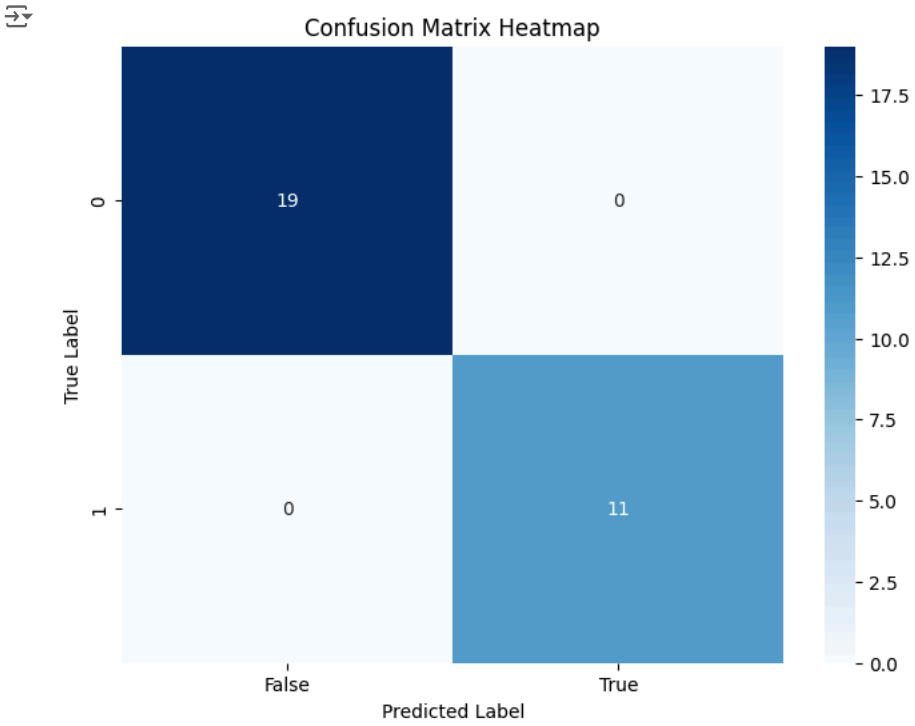
```
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
print()
```

```
Confusion Matrix:
[[19  0]
 [ 0 11]]
```

```
# Classification report (formatted as a DataFrame)
report_dict = classification_report(y_test, y_pred, target_names=label_encoder.classes_, output_dict=True)
report_df = pd.DataFrame(report_dict).transpose()
print("Classification Report:\n", report_df)
```

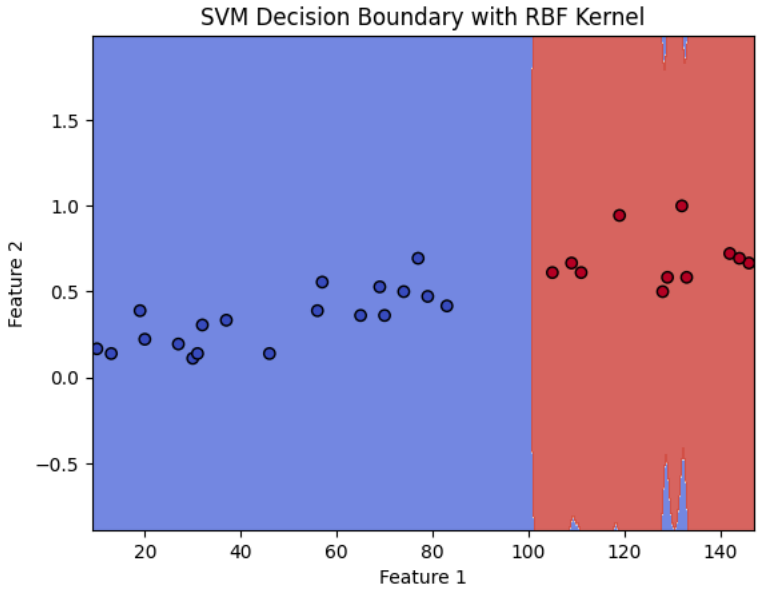
```
Classification Report:
                precision  recall  f1-score  support
False               1.0     1.0       1.0     19.0
True                1.0     1.0       1.0     11.0
accuracy            1.0     1.0       1.0      1.0
macro avg           1.0     1.0       1.0     30.0
weighted avg        1.0     1.0       1.0     30.0
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=1)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix Heatmap')
plt.show()
```




```
def plot_decision_boundary(X, y, model):
    h = 0.02  # step size in the mesh
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    # Predict the class label for each point in the mesh
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Create the contour plot
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap=plt.cm.coolwarm)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('SVM Decision Boundary with RBF Kernel')
    plt.show()
```

```
plot_decision_boundary(X_test, y_test, svm_model)
```


SVM Decision Boundary with RBF Kernel
Feature 2
Feature 1
1.5
1.0
0.5
0.0
−0.5
20
40
60
80
100
120
140

Om Shete

Experiment No: 7

Aim: Implement ensemble methods to combine different models.

Theory:

- Ensemble methods are machine learning techniques that combine the predictions of multiple models to produce a more robust, accurate prediction.
- The idea is that by combining models, the weaknesses of one model can be compensated by the strengths of others.
- Below are the most common ensembles methods used to combine different models:

1) **Bagging** (Bootstrap Aggregating)

- Bagging involves training multiple instances of the same model on different random subsets of the training data.
- Each model is trained independently, and their predictions are combined, typically by averaging or majority voting.
- For eg.,

Random Forest is a bagging method that builds multiple decision trees and combines them to produce more accurate and stable predictions.

2) Boosting

⇒ Boosting sequentially trains models, where each subsequent model tries to correct the errors of the previous one.

- Models are not trained independently, and the errors made by previous models are given more importance in each new model.
- For eg., Gradient Boosting, AdaBoost, and XGBoost are common boosting algorithms that combine weak learners to form a strong prediction model.

3) Stacking

⇒ Stacking involves training multiple different models and then using another model to combine the predictions of these base models.

- The meta-model takes the predictions of the base models as input and learns how to best combine them to make the final prediction.

A
27/9/24

```
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 LabelEncoder
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Load dataset from CSV
data = pd.read_csv('p_iris.csv')
```

```
# Use only the first two features for training and visualization
X = data.iloc[:, :2].values # First two features
y = data.iloc[:, -1].values # Target variable (last column)
```

```
# Encode target labels (species) to numeric values
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.3, random_state=42)
```

```
# 1. SVM Model
svm_rbf = SVC(kernel='rbf', gamma='auto', probability=True)
svm_rbf.fit(X_train, y_train)
y_pred_svm = svm_rbf.predict(X_test)
```

```
# 2. Random Forest Model (Bagging)
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

```
# 3. AdaBoost Model (Boosting)
ada = AdaBoostClassifier(base_estimator=SVC(kernel='linear', probability=True), n_estimators=50, random_state=42)
ada.fit(X_train, y_train)
y_pred_ada = ada.predict(X_test)
```

```
c:\Users\Om Shete\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\ensemble\_base.py:156: FutureWarning: `base_esti
  warnings.warn(
```

```
# Combine predictions using majority voting
y_pred_ensemble = np.array([y_pred_svm, y_pred_rf, y_pred_ada])
y_pred_final = np.array([np.bincount(x).argmax() for x in y_pred_ensemble.T])
```

```
# Accuracy for each model
for model_name, y_pred in zip(['SVM', 'Random Forest', 'AdaBoost', 'Ensemble'], [y_pred_svm, y_pred_rf, y_pred_ada, y_pred_final]):
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{model_name} Accuracy: {accuracy:.4f}\n")
```

```
SVM Accuracy: 0.9778

Random Forest Accuracy: 1.0000

AdaBoost Accuracy: 1.0000

Ensemble Accuracy: 1.0000
```

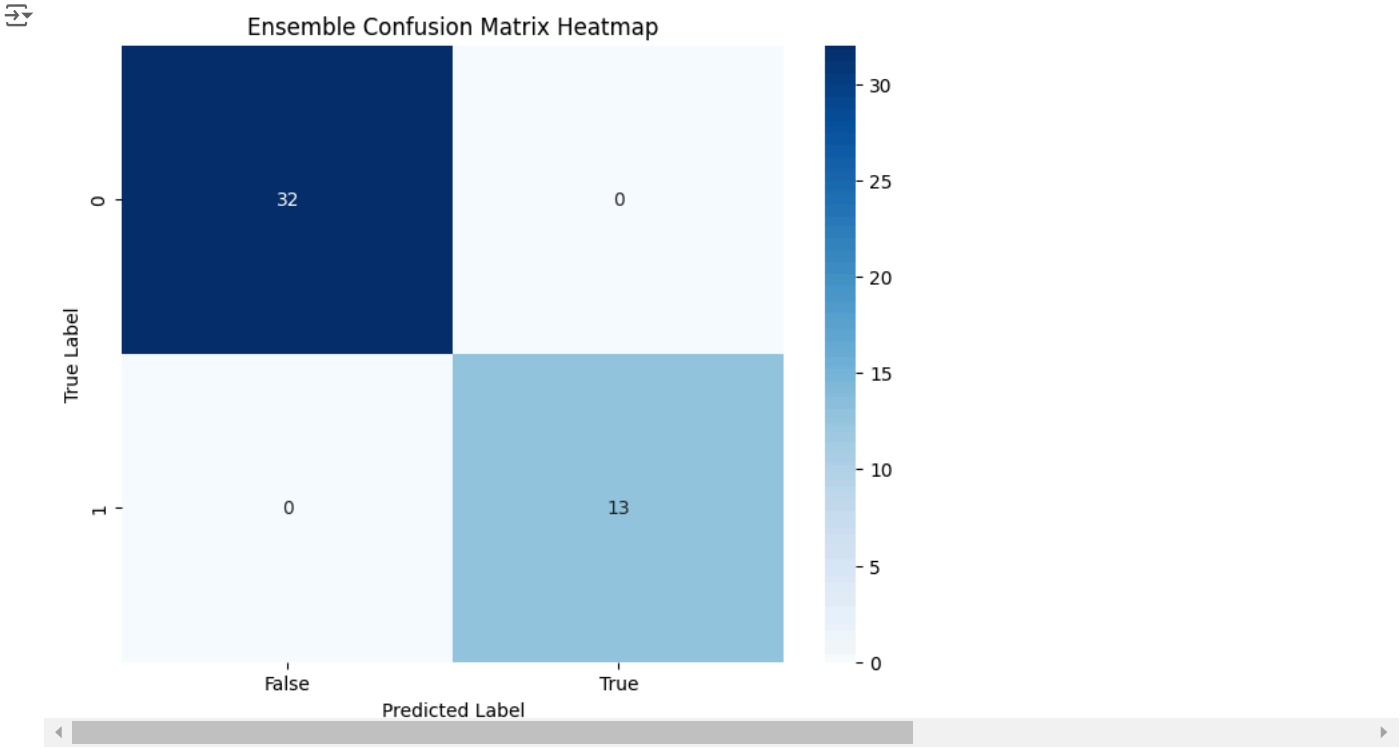
```
# Classification report for the ensemble model
report_dict = classification_report(y_test, y_pred_final, target_names=label_encoder.classes_, output_dict=True)
report_df = pd.DataFrame(report_dict).transpose()
print("Ensemble Classification Report:\n", report_df)
```

```
Ensemble Classification Report:
                precision  recall  f1-score  support
False               1.0     1.0       1.0     32.0
True                1.0     1.0       1.0     13.0
accuracy            1.0     1.0       1.0      1.0
macro avg           1.0     1.0       1.0     45.0
weighted avg        1.0     1.0       1.0     45.0
```

```
# Confusion matrix for the ensemble model
conf_matrix = confusion_matrix(y_test, y_pred_final)
print("\nEnsemble Confusion Matrix:")
print(conf_matrix)
```

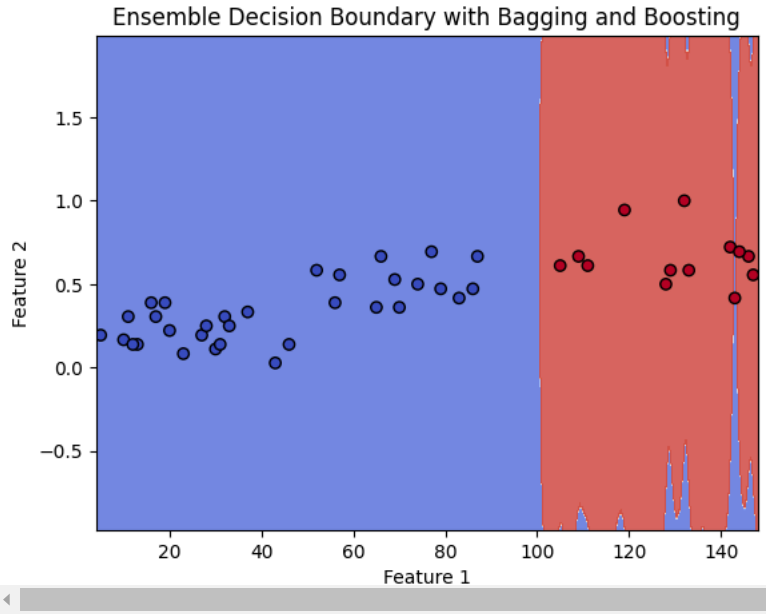
```
Ensemble Confusion Matrix:
[[32  0]
 [ 0 13]]
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=1)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Ensemble Confusion Matrix Heatmap')
plt.show()
```




```
def plot_decision_boundary(X, y, model):
    h = .02 # step size in the mesh
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap=plt.cm.coolwarm)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Ensemble Decision Boundary with Bagging and Boosting')
    plt.show()
```

```
# Since we can't train an ensemble model directly, we just plot the decision boundary using the SVM model
plot_decision_boundary(X_test, y_test, svm_rbf)
```


Ensemble Decision Boundary with Bagging and Boosting
Feature 2
Feature 1
1.5
1.0
0.5
0.0
−0.5
20
40
60
80
100
120
140

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

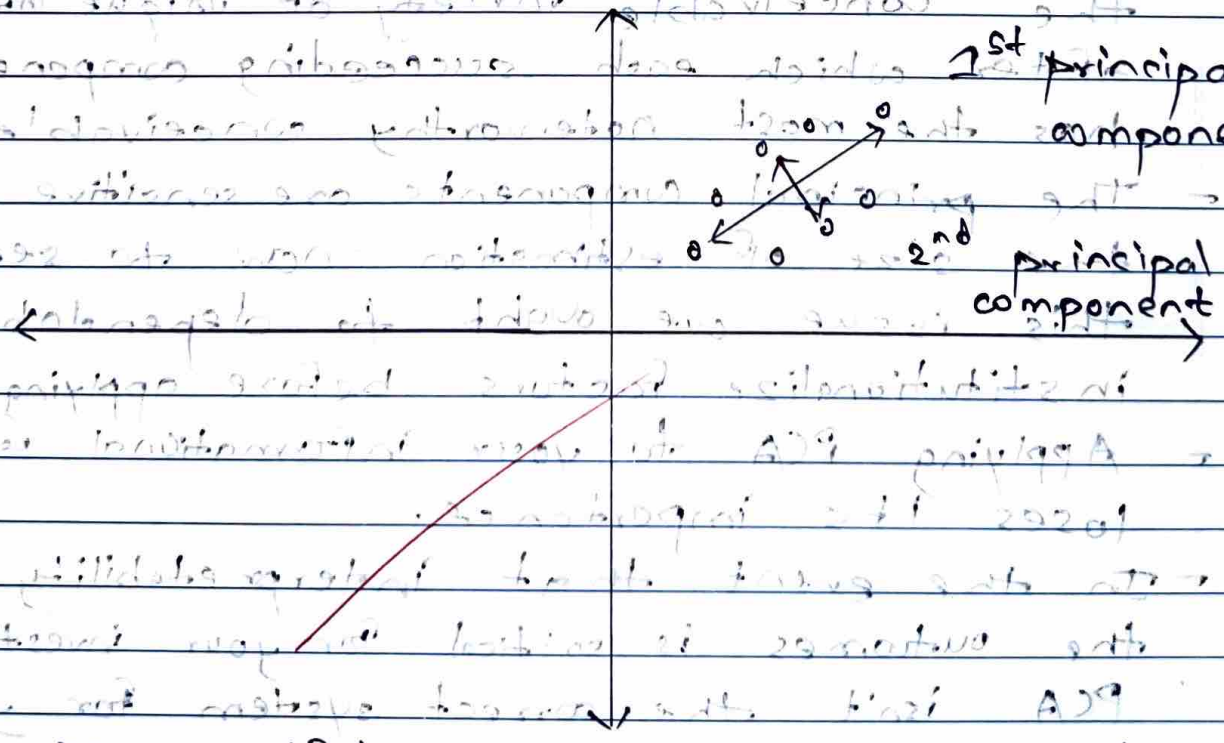
On Shelf

Experiment No: 8

Aim: Implement Principal Component Analysis technique.

Theory:

- Dimensional reduction is regularly the best beginning stage when dealing with high dimensional information.
- It is utilized for an assortment of reasons, from perception to denoising, and in a wide range of uses from signal processing to bioinformatics.
- A standout amongst the most broadly utilized dimensional reduction tools is Principal Component Analysis (PCA).



- PCA verifiably accept that the dataset under thought is typically dispersed, furthermore

chooses the subspace which expands the anticipated difference.

- We consider a centered data set and develop the sample covariance matrix; at that point q-dimensional PCA is identical to anticipating onto the q-dimensional subspace spread over by the q eigenvectors of S with biggest eigenvalues.
- In this system, variables are changed into another arrangement of variables, which are straight blend of unique variables.
- These new arrangement of variables are known as principal components.
- They are calculated so that first principal component is represents a large portion of the conceivable variety of unique information after which each succeeding component has the most noteworthy conceivable variance.
- The principal components are sensitive to the size of estimation, now to settle this issue we ought to dependably institutionalize factors before applying PCA.
- Applying PCA to your informational collection loses its importance.
- In the event that interpretability of the outcomes is critical for your investigation, PCA isn't the correct system for your task.

A

21/9/22

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

```
data = pd.read_csv('p_iris.csv')
```

```
# Assuming the last column is the target label and the rest are features
X = data.iloc[:, :-1].values # Features (all rows, all columns except the last)
y = data.iloc[:, -1].values # Target (all rows, last column)
```

```
# Step 2: Standardize the data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

```
# Step 3: Calculate the covariance matrix
cov_matrix = np.cov(X_std.T)
```

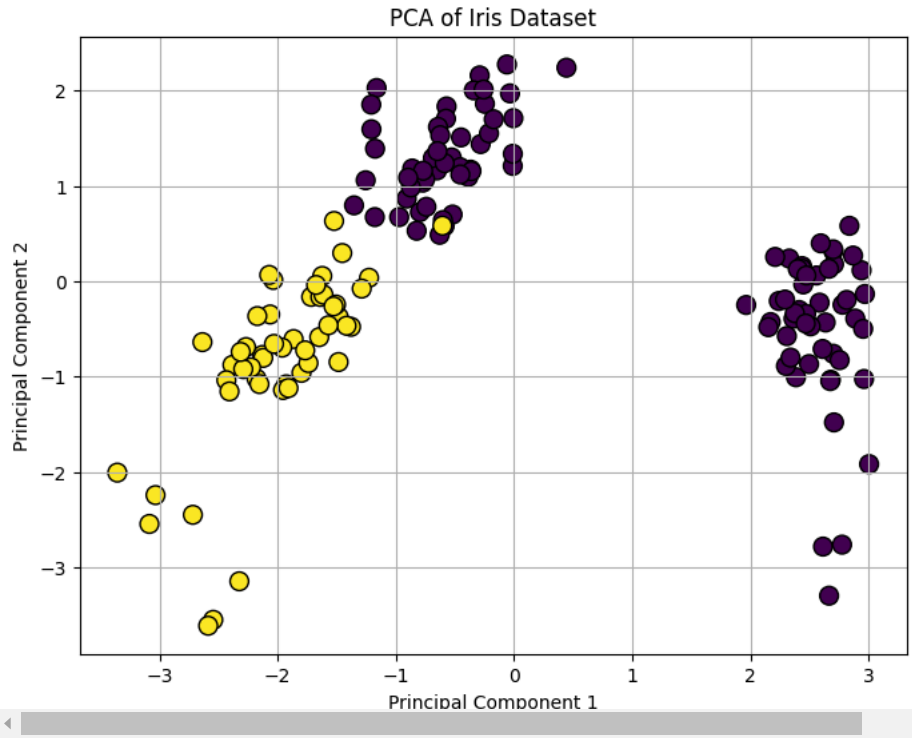
```
eigen_values , eigen_vectors = np.linalg.eig(cov_matrix)
```

```
sorted_indices = np.argsort(eigen_values)[::-1]
eigenvalues_sorted = eigen_values[sorted_indices]
eigenvectors_sorted = eigen_vectors[:, sorted_indices]
```

```
n_components = 2
eigenvectors_subset = eigenvectors_sorted[:, :n_components]
```

```
X_pca = X_std.dot(eigenvectors_subset)
```

```
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, edgecolor='k', s=100)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.grid()
plt.show()
```



Report on Mini Project

Subject: Machine Learning

AY: 2024-25

Air Passenger Data Analysis - Time Series Forecasting - SARIMAX

Parth Puranik: 2103142

Mohib Abbas Sayed: 2103158

Hamza Sayyed: 2103159

Om Shete: 2103163

Guided By

(Tanuja Sarode)

10/10/24

CHAPTER 1: INTRODUCTION

In this project, we aim to forecast the number of air passengers for the next 24 months based on historical data from 1949 to 1960. This type of analysis is crucial in the aviation industry as it aids in capacity planning, demand forecasting, and ensuring smooth operations. The dataset consists of monthly totals of US airline passengers over 12 years. Time series forecasting is essential for capturing the data's underlying patterns such as seasonality, trends, and cycles.

We use the ARIMA (AutoRegressive Integrated Moving Average) model, particularly SARIMAX (Seasonal ARIMA with Exogenous Regressors), which accounts for seasonality and other external factors, to provide accurate forecasts. We aim to implement this model and generate estimates for the upcoming 24 months based on the data trends from 1949 to 1960.

CHAPTER 2: DATA DESCRIPTION AND ANALYSIS

2.1 Dataset Overview

The dataset used in this project contains two key attributes:

- **Month**: The period in which the passengers travelled.
- **Passengers**: The total number of passengers who travelled in each month.

The dataset spans from January 1949 to December 1960, consisting of 144 observations. Each data point represents the total number of passengers for that month.

2.2 Data Preprocessing

Before proceeding with time series modelling, the following preprocessing steps were performed:

- **Parsing Dates**: The 'Month' attribute was converted into a DateTime object for time-based operations.

- **Handling Missing Data**: The dataset did not contain any missing values.
- **Stationarity Check**: The Augmented Dickey-Fuller (ADF) test was applied to check the stationarity of the series.
- **Seasonality and Trends**: Visualizations such as line plots and seasonal decomposition were used to identify trends, seasonality, and cycles in the data.

2.3 Exploratory Data Analysis (EDA)

Using libraries such as matplotlib and seaborn, we generated the following visual insights:

- **Trend Analysis**: A clear upward trend in the number of passengers over the years.

- **Seasonal Patterns**: A yearly recurring seasonal pattern was evident, with peak months typically being mid-year (summer travel).
- **Correlation**: We analyzed autocorrelations and partial autocorrelations to explore the relationship between monthly passenger numbers over time.

CHAPTER 3: DESIGN OF DATA PIPELINE

The data pipeline for this project is designed to ensure a seamless flow from data ingestion to model deployment. The pipeline consists of the following steps:

3.1 Data Ingestion

The dataset was loaded using the pandas library, and date parsing was performed to ensure proper time series handling.

3.2 Data Preprocessing

- **Handling DateTime Format**: The 'Month' column was converted to a proper datetime index to facilitate time series analysis.

- **Differencing**: To achieve stationarity, differencing techniques were applied to remove trends and seasonality from the data.
- **Scaling**: Data normalization was applied to ensure consistency across variables, preparing the dataset for the ARIMA model.

3.3 Model Selection and Training

ARIMA Model: The ARIMA model was selected based on the dataset characteristics. We utilized SARIMAX (Seasonal ARIMA) due to the clear seasonality in the data. The model parameters (p, d, q) and seasonal components were tuned using grid search methods to optimize performance.

The model training process involved splitting the data into training and testing sets, followed by fitting the SARIMAX model on the training data.

3.4 Model Evaluation

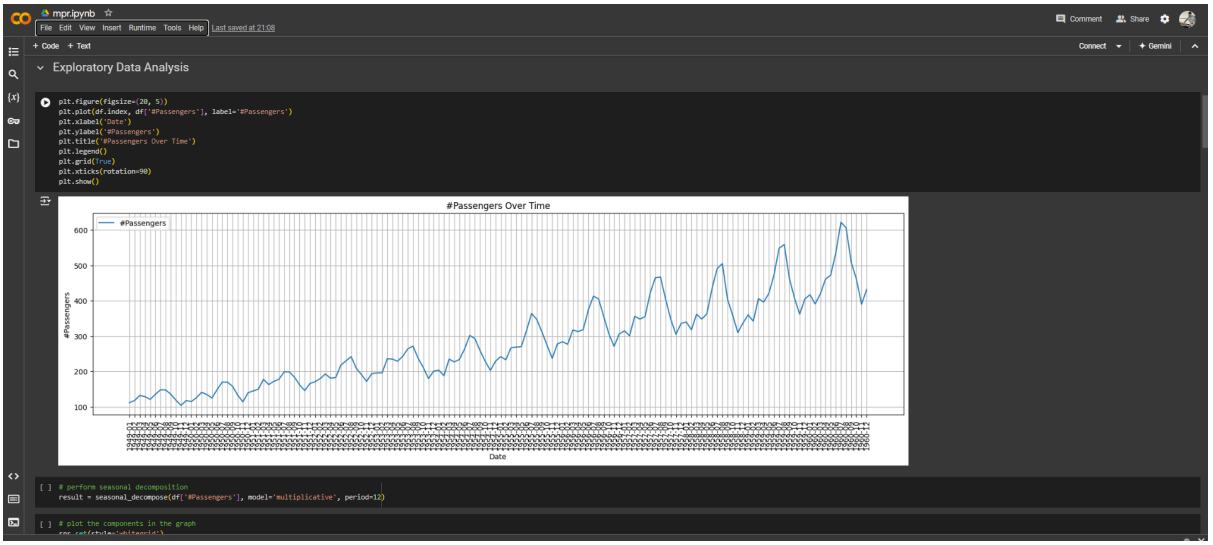
- **Performance Metrics**: Metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and AIC were used to assess model performance.

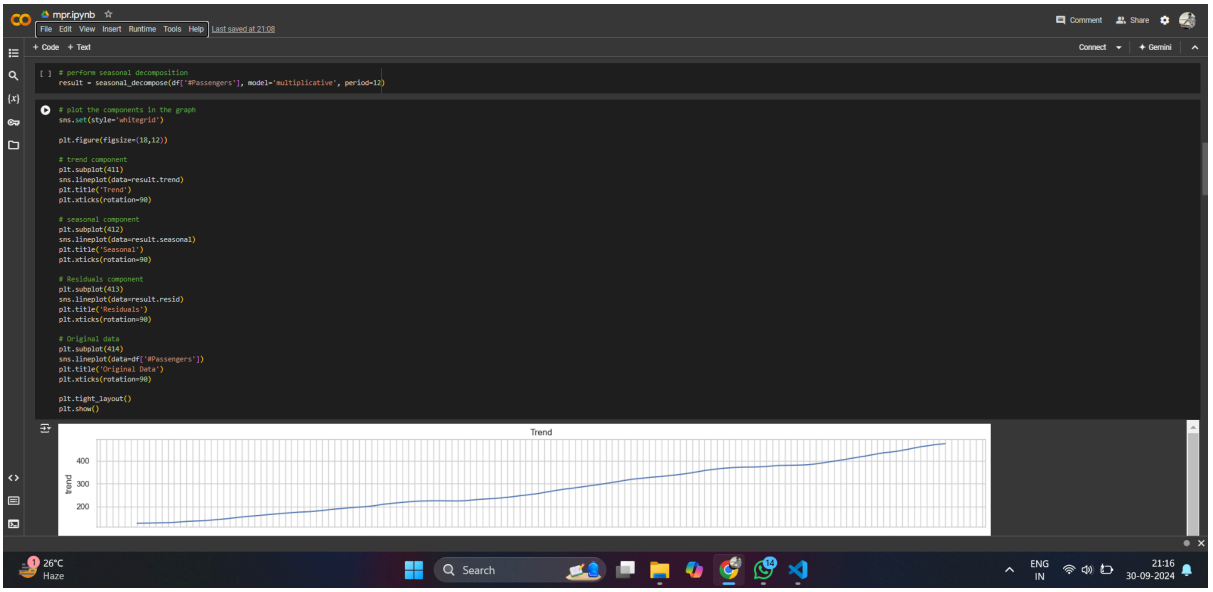
- **Cross-Validation**: The model's robustness was evaluated using time-series cross-validation techniques.

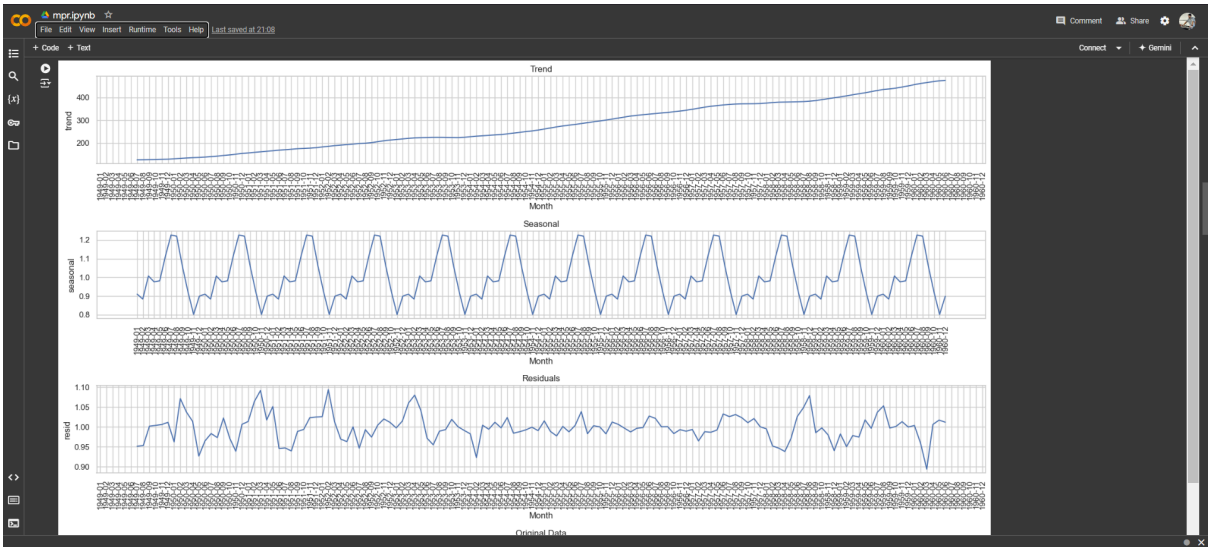
3.5 Forecasting

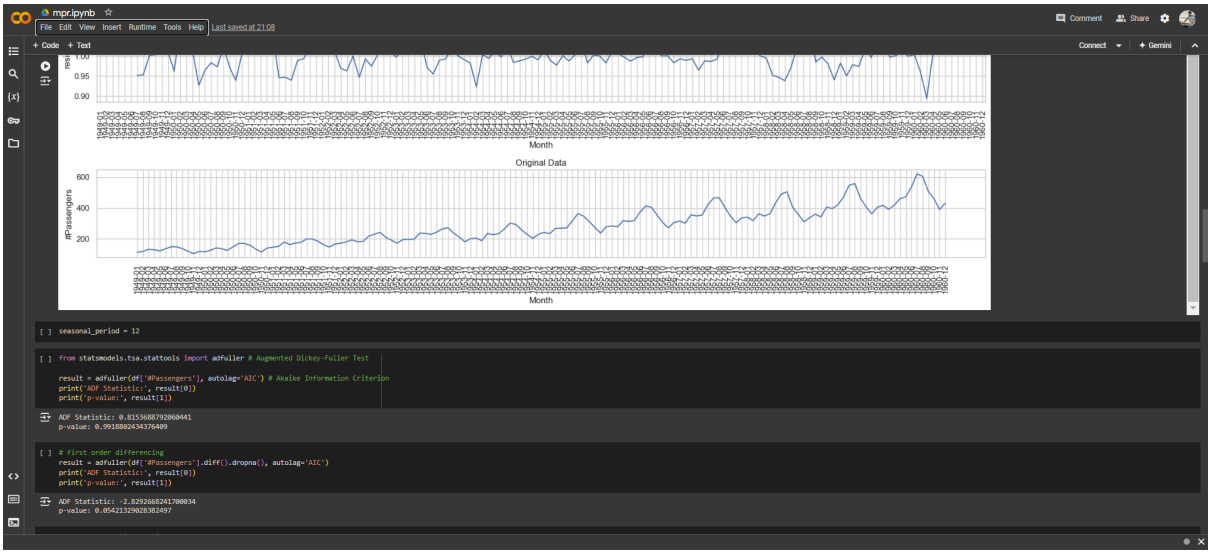
After model training and evaluation, we generated forecasts for the next 24 months. Visualizations were created to compare the forecasts with actual data, providing insights into expected trends and passenger growth.

CHAPTER 4: RESULT ANALYSIS




mpr.ipynb
File Edit View Insert Runtime Tools Help Last saved at 21:08
Comment Share
+ Code + Text
Connect + Gemini
[ ] # perform seasonal decomposition
result = seasonal_decompose(df['#Passengers'], model='multiplicative', period=12)
# plot the components in the graph
sns.set(style='whitegrid')
plt.figure(figsize=(18,12))
# trend component
plt.subplot(411)
sns.lineplot(data=result.trend)
plt.title('Trend')
plt.xticks(rotation=90)
# seasonal component
plt.subplot(412)
sns.lineplot(data=result.seasonal)
plt.title('Seasonal')
plt.xticks(rotation=90)
# Residuals component
plt.subplot(413)
sns.lineplot(data=result.resid)
plt.title('Residuals')
plt.xticks(rotation=90)
# Original data
plt.subplot(414)
sns.lineplot(data=df['#Passengers'])
plt.title('Original Data')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
Trend
trend
400
300
200
26°C
Haze
Search
ENG
IN
21:16
30-09-2024


mpr.ipynb
File Edit View Insert Runtime Tools Help Last saved at 21:08
Comment Share
+ Code + Text
Connect + Gemini
Trend
trend
400
300
200
Month
Seasonal
seasonal
1.2
1.1
1.0
0.9
0.8
Month
Residuals
resid
1.10
1.05
1.00
0.95
0.90
Month
Original Data


mpr.ipynb
File Edit View Insert Runtime Tools Help Last saved at 21:08
Comment Share
+ Code + Text
Connect + Gemini
Month
Original Data
#Passengers
600
400
200
Month
[ ] seasonal_period = 12
[ ] from statsmodels.tsa.stattools import adfuller # Augmented Dickey-Fuller Test
result = adfuller(df['#Passengers'], autolag='AIC') # Akaike Information Criterion
print('ADF Statistic:', result[0])
print('p-value:', result[1])
ADF Statistic: 0.8153688793060441
p-value: 0.991880243437641
[ ] # first order differencing
result = adfuller(df['#Passengers'].diff().dropna(), autolag='AIC')
print('ADF Statistic:', result[0])
print('p-value:', result[1])
ADF Statistic: -2.8202668241700834
p-value: 0.05421329028382497

```
seasonal_period = 12
```

```
from statsmodels.tsa.stattools import adfuller # Augmented Dickey-Fuller Test

result = adfuller(df['#Passengers'], autolag='AIC') # Akaike Information Criterion
print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

```
ADF Statistic: 0.8153688792060441
p-value: 0.9918802434376409
```

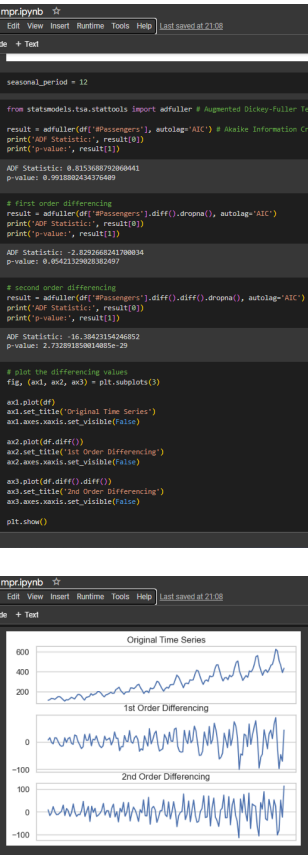
```
# first order differencing
result = adfuller(df['#Passengers'].diff().dropna(), autolag='AIC')
print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

```
ADF Statistic: -2.8292668241700034
p-value: 0.05421329028382497
```

```
# second order differencing
result = adfuller(df['#Passengers'].diff().diff().dropna(), autolag='AIC')
print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

```
ADF Statistic: -16.384231542468529
p-value: 2.732891850014085e-29
```

```
# plot the differencing values
fig, (ax1, ax2, ax3) = plt.subplots(3)

ax1.plot(df)
ax1.set_title("Original Time Series")
ax1.axes.xaxis.set_visible(False)

ax2.plot(df.diff())
ax2.set_title('1st Order Differencing')
ax2.axes.xaxis.set_visible(False)

ax3.plot(df.diff().diff())
ax3.set_title('2nd Order Differencing')
ax3.axes.xaxis.set_visible(False)

plt.show()
```

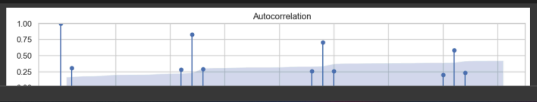



```
# the time series becomes stationary after first order differencing
```

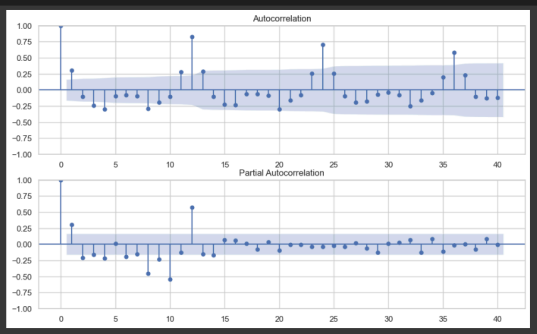
Define Parameters for ARIMA

```
# p = 0 # MA - Moving Average - PACF
# d = 1 # order of differencing - I
# q = 0 # AR - Auto Regressive - ACF
```

```
fig, ax = plt.subplots(2, 1, figsize=(12, 7))
sm.graphics.tsa.plot_acf(df.diff().dropna(), lags=40, ax=ax[0])
sm.graphics.tsa.plot_pacf(df.diff().dropna(), lags=40, ax=ax[1])
plt.show()
```




```
fig, ax = plt.subplots(2, 1, figsize=(12, 7))
sm.graphics.tsa.plot_acf(df.diff().dropna(), lags=40, ax=ax[0])
sm.graphics.tsa.plot_pacf(df.diff().dropna(), lags=40, ax=ax[1])
plt.show()
```




```
p = 2 # pacf
d = 1 # 1st order difference
q = 1 # acf
```

```
P = 1
D = 0
Q = 3
```

mpr.ipynb

File Edit View Insert Runtime Tools Help Last saved at 21:08

Comment Share

+ Code + Text

Connect + Gemini

Model Training

```
# define the arima model
from statsmodels.tsa.statespace.sarimax import SARIMAX

model = SARIMAX(df['#Passengers'], order=(p, d, q), seasonal_order=(P, D, Q, seasonal_period))
fitted_model = model.fit()
print(fitted_model.summary())
```

```
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                          #Passengers   No. Observations:                  144
Model:             SARIMAX(2, 1, 1)x(1, 0, [1, 2, 3], 12)   Log Likelihood        -563.224
Date:                                Mon, 30 Sep 2024   AIC                      1142.448
Time:                                        18:26:03   BIC                      1166.151
Sample:                                  01-01-1949   HQIC                     1152.080
                                       - 12-01-1960
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.6244      0.101      6.168      0.000       0.426       0.823
ar.L2          0.1947      0.100      1.951      0.051      -0.001       0.390
ma.L1         -0.9675      0.039    -24.632      0.000      -1.045      -0.891
ar.S.L12       0.9638      0.036     26.615      0.000       0.893       1.033
ma.S.L12      -0.1127      0.126     -0.898      0.369      -0.359       0.133
ma.S.L24       0.1355      0.129      1.053      0.292      -0.117       0.388
ma.S.L36       0.0049      0.147      0.033      0.973      -0.284       0.294
sigma2       124.2089     14.750      8.421      0.000      95.299     153.119
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):                16.14
Prob(Q):                              0.93   Prob(JB):                         0.00
Heteroskedasticity (H):               3.99   Skew:                             0.18
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.61
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Forecasting

```
[ ] # forecast for next 2 years
forecast_steps = 24
forecast = fitted_model.get_forecast(steps=forecast_steps)
```

mpr.ipynb

File Edit View Insert Runtime Tools Help Last saved at 21:08

Comment Share

+ Code + Text

Connect + Gemini

Forecasting

+ Code + Text

```
[ ] # forecast for next 2 years
forecast_steps = 24
forecast = fitted_model.get_forecast(steps=forecast_steps)

# create the date range for the forecasted values
forecast_index = pd.date_range(start=df.index[-1], periods=forecast_steps+1, freq='M')[1:].strftime('%Y-%m') # remove start date
```

```
[ ] # create a forecast dataframe
forecast_df = pd.DataFrame({
    "Forecast": list(forecast.predicted_mean),
    "Lower CI": list(forecast.conf_int().iloc[:, 0]),
    "Upper CI": list(forecast.conf_int().iloc[:, 1])
}, index=forecast_index)

forecast_df.head()
```

	Forecast	Lower CI	Upper CI
1961-01	446.711482	424.867844	468.555119
1961-02	423.325409	397.191175	449.459823
1961-03	456.418435	426.807576	486.029294
1961-04	491.562749	459.538919	523.586579
1961-05	505.131996	471.260404	538.003589

```
[ ] # plot the forecast values

plt.figure(figsize=(12, 6))
plt.plot(df['#Passengers'], label='Historical Data')
plt.plot(forecast_df['Forecast'], label='Forecast Data')
plt.fill_between(forecast_df.index, forecast_df['Lower CI'], forecast_df['Upper CI'], color='k', alpha=0.1)
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.title('Air Passengers Forecast')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```

Air Passengers Forecast

mpr.ipynb

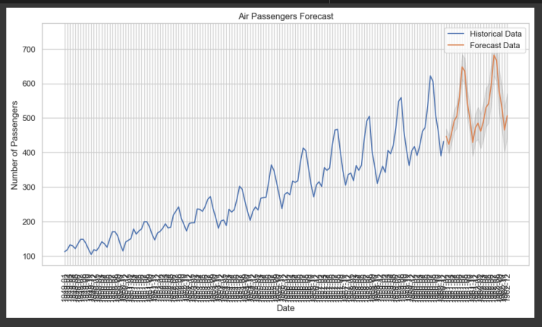
File Edit View Insert Runtime Tools Help Last saved at 21:08

Comment Share

+ Code + Text

Connect + Gemini

```
# plot the forecast values

plt.figure(figsize=(12, 6))
plt.plot(df['#Passengers'], label='Historical Data')
plt.plot(forecast_df['Forecast'], label='Forecast Data')
plt.fill_between(forecast_df.index, forecast_df['Lower CI'], forecast_df['Upper CI'], color='k', alpha=0.1)
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.title('Air Passengers Forecast')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



CHAPTER 5: CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

In this project, we successfully implemented time series forecasting on the Air Passenger dataset using the SARIMAX (Seasonal ARIMA with Exogenous Regressors) model. Through exploratory data analysis, we identified clear trends and seasonality in the dataset, which were crucial for accurate forecasting.

Our analysis revealed that air passenger traffic has been steadily increasing over time, with consistent seasonal peaks during the summer months. By applying the SARIMAX model, we were able to capture these patterns effectively and generate reliable forecasts for the next 24 months.

The accuracy of the model was evaluated using key performance metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The results showed that the SARIMAX model was able to make reasonably accurate predictions, though there were some challenges with periods of higher variability in passenger numbers.

This project demonstrates the effectiveness of time series analysis and forecasting techniques in the airline industry, where accurate demand predictions can lead to better resource management, optimized flight scheduling, and improved customer service.

5.2 Future Scope

While the results of this analysis are promising, there are several areas for future improvement and exploration:

- **Model Refinement**: Future work can focus on further tuning the SARIMAX model or experimenting with more advanced models like Prophet or deep learning-based models (e.g., LSTM or GRU) to improve accuracy and better capture fluctuations in data.

- **Incorporating External Factors**: The current model only uses historical passenger data for predictions. Incorporating external factors such as fuel prices, economic conditions, or weather data could provide a more comprehensive forecast.
- **Real-Time Forecasting**: The current model is built on historical data up to 1960. A real-time data integration approach could allow for continuous updates to forecasts, enabling more dynamic and responsive models in a real-world setting.
- **Longer-Term Forecasting**: While this project focused on a 24-month forecast horizon, future studies could explore longer-term predictions, considering multi-year trends and global shifts in air travel demand.
- **Data Augmentation**: Exploring additional data sources, such as data from multiple airlines or other regions, could enhance model performance and provide broader insights into air passenger traffic patterns.
- **Forecast Validation in Modern Context**: While this dataset is historic, testing the forecasting model on more recent datasets would provide insights into how well these traditional models perform in today's airline industry context, which may have new patterns due to modern technologies and global factors like pandemics.

By expanding the scope of the model and incorporating more advanced techniques, this study can be extended to provide even more robust and actionable insights for the airline industry.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Om Shete
Group 7

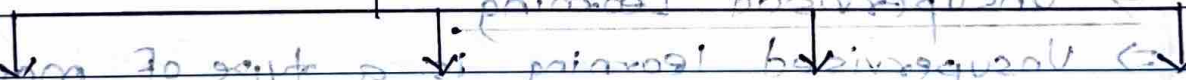
Assignment 1

Q.1 What is learning? Explain different types of learning with examples.

⇒

- Machine learning is a subset of AI, which enables the machine to automatically learn from data, improve performance from past experiences and make predictions.
- Learning contains a set of algorithms that works on a huge amount of data.
- Data is fed to these algorithms to train them, and on the basis of training, they build the model and perform a specific task.

- Types of Learning :-



1) Supervised Learning

⇒ Supervised learning is defined as when a model gets trained on a "Labelled Dataset".

- Labelled Dataset have both inputs and outputs parameters.
- In supervised learning algorithms learn to map points between inputs and correct outputs.
- It has both training and validation datasets labelled.

- For e.g., consider the scenario where you have to build an image classifier to differentiate between cats and dogs.
- If you feed the datasets of dogs and cats labelled images to the algorithm, the machine will learn to classify between a dog or a cat from these labeled images.
- When we input new dog or cat images that it has never seen before, it will use the learned algorithms and predict whether it is a dog or cat.
- This is how supervised learning works and this is particularly an image classification.

2) Unsupervised Learning

⇒ Unsupervised learning is a type of machine learning technique in which an algorithm discovers patterns and relationships using unlabeled data.

- Unlike supervised learning, unsupervised learning doesn't involve providing the algorithm with labeled target outputs.
- The primary goal of Unsupervised learning often involve providing the is often to discover hidden patterns, similarities or clusters within the data, which can be then used for various purposes, such as data exploration, visualization, dimensionality reduction and more.

3) Semi-supervised Learning

⇒ Semi-supervised is a machine learning algorithm that works between supervised and unsupervised learning so it uses both labelled and unlabelled data.

- It's particularly useful when obtaining labeled data is costly, time-consuming, or resource-intensive.
- This approach is useful when the dataset is expensive and time-consuming.
- Semi-supervised learning is chosen when labeled data require skills and relevant resources in order to train or learn from it.

4) Reinforcement learning

⇒ Reinforcement learning algorithm is a learning method that interacts with the environment by producing actions and discovering errors.

- Trial, error and delay are the most relevant characterstics of reinforcement learning.
- For eg, consider that you are training an AI agent to play a game like chess.
- The agent explores different moves and receives positive or negative feedback based on the outcome.
- Reinforcement learning also finds applications in the which they learn to perform tasks by interacting with their surroundings.

Q.2 Define overfitting and underfitting. How to evaluate a ML model for overfitting or underfitting explain using diagram. What measures need to be taken in case of overfitting and underfitting?

⇒

- **Overfitting**

⇒ A statistical model is said to be overfitted when the model does not make accurate predictions on testing data. When a model gets trained with so much data it starts learning from the noise and inaccurate data entries in our data set.

- And when testing with test data results in High variance. Then the model does not categorize the data correctly, because of too many details and noise.

- **Underfitting**

⇒ A statistical model or machine learning algorithm is said to have underfitting when a model is too simple to capture data complexities.

- It represents the inability of the model to learn the training data effectively result in poor performance both on the training and testing data.

- In simple terms, an underfit model's are inaccurate, especially when applied to new, unseen examples.
- Techniques to evaluate Overfitting & Underfitting.

1) Cross-validation

⇒ Use k-fold cross-validation to assess the model's performance. The dataset is divided into k subsets, and the model is trained and validated k times, each time using a different subset as the validation set and the remaining as a training set.

- Average the performance metrics across the k trials.

2) Learning Curves

⇒ Plot learning curves for both training and test errors. Learning curves shows the model's performance on the training set and validation set as function of the number of training examples or training epochs.

- Measures to Address Overfitting:

1) Increase Training Data

⇒ Gather more training data to help the model generalize better.

- Use data augumentation techniques to artificially

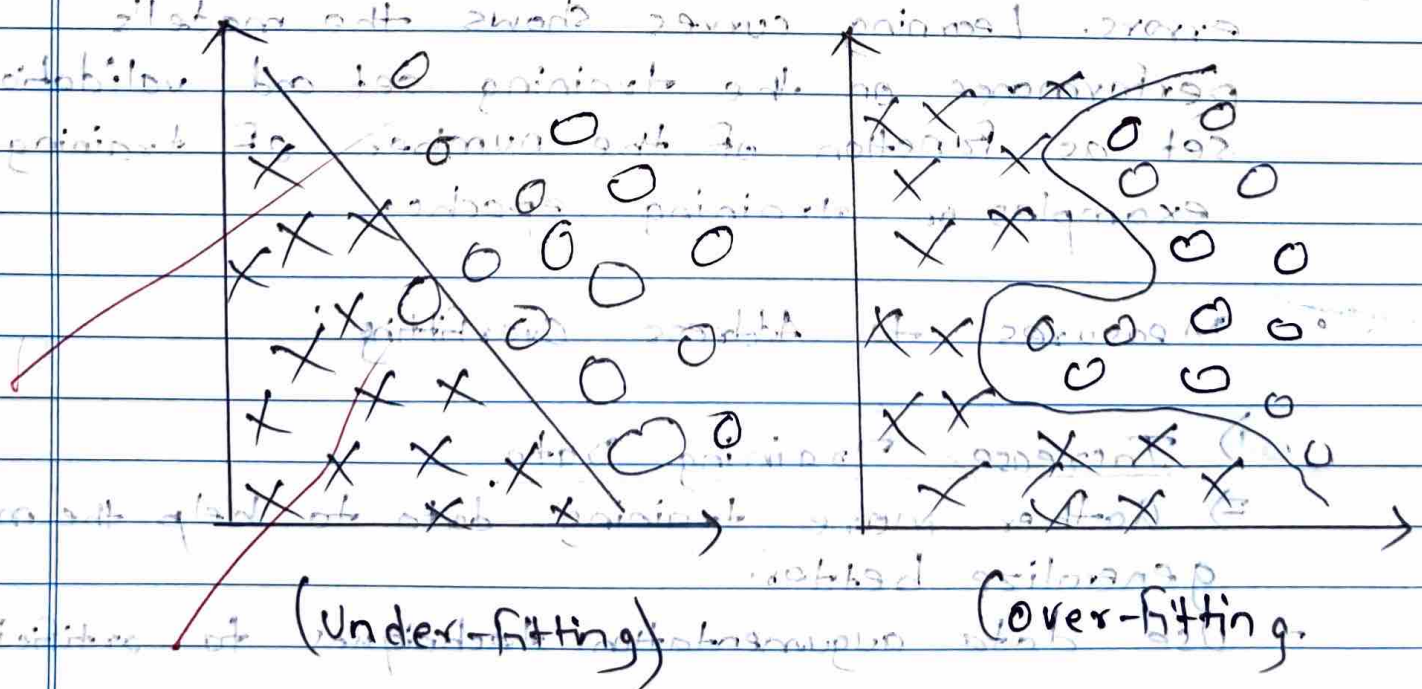
increase the size of the training dataset.

2) Runing
⇒ Remove the parts of tree that provide little power to the predict target variables to reduce the model complexity.

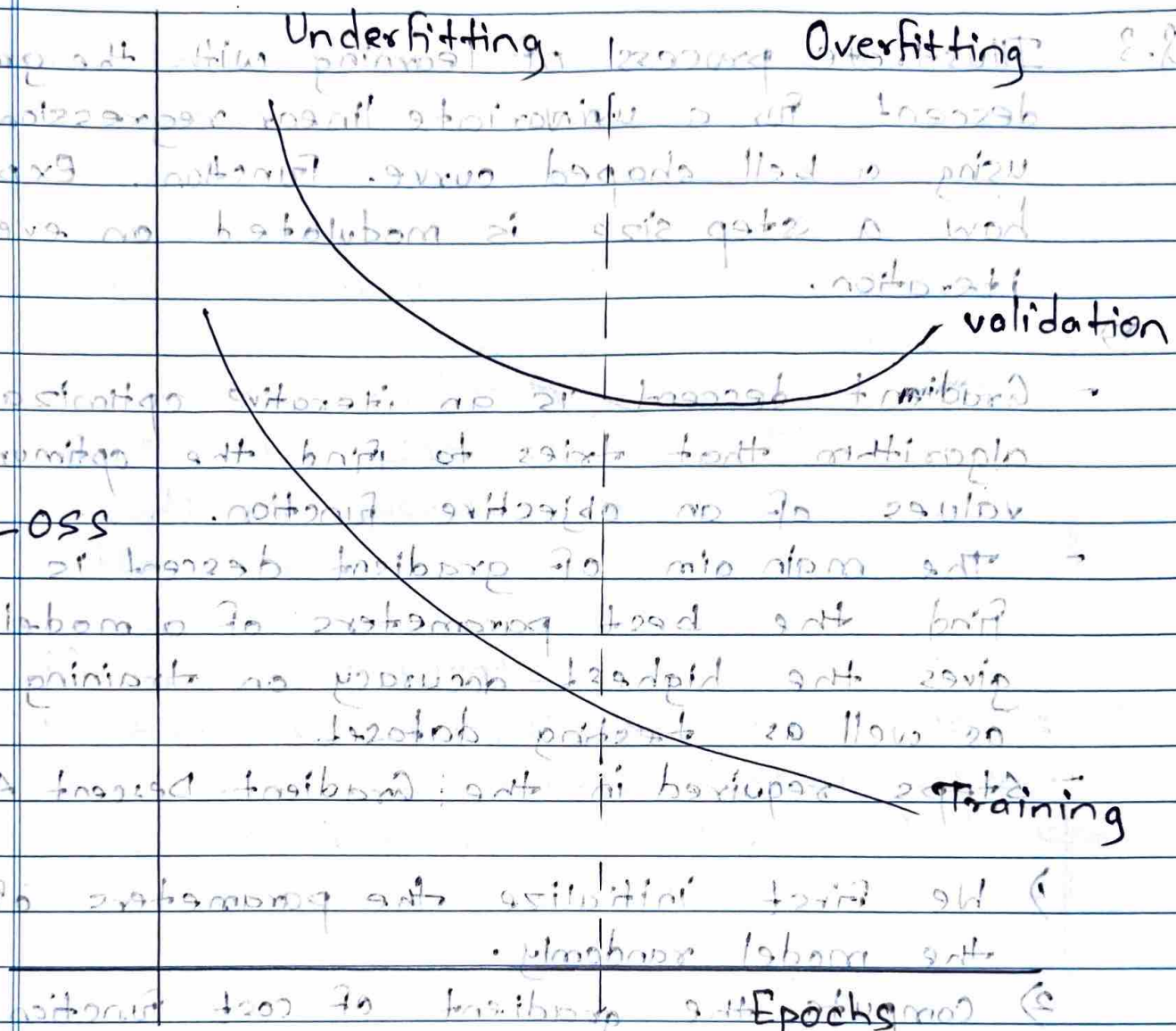
- Measures to address Underfitting:

1) Increase Model complexity
⇒ Use more complex models or algorithms that can capture more intricate patterns in the data.
Increase the number of features or parameters.

2) Train Longer
⇒ Train the model for more epochs to allow it to learn more from the data.



(Under-fitting) (Over-fitting)


Underfitting.
Overfitting
validation
Loss
training
Epochs
Early
stopping

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Q.3 Illustrate process of learning with the gradient descent for a univariate linear regression, using a bell shaped curve. Function. Explain how a step size is modulated on every iteration.

⇒

- Gradient descent is an iterative optimization algorithm that tries to find the optimum values of an objective function.
- The main aim of gradient descent is to find the best parameters of a model which gives the highest accuracy on training data as well as testing dataset.
- Steps required in the Gradient Descent Algorithm:-

1) We first initialize the parameters of the model randomly.
2) Compute the gradient of cost function with respect to each parameter.
3) Update the parameter of the models by taking step in the opposite direction of the model.
4) Repeat step 2 and 3 iteratively to get the best parameter for defined model.

- To apply gradient-descent using programming language we make use of four functions:-

i) gradient_descent
⇒ We make predictions on dataset and compute the difference between predicted and actual target value and accordingly update the parameter to return it.

ii) compute_prediction
⇒ In this function, we will compute the prediction using the parameters at each iterations.

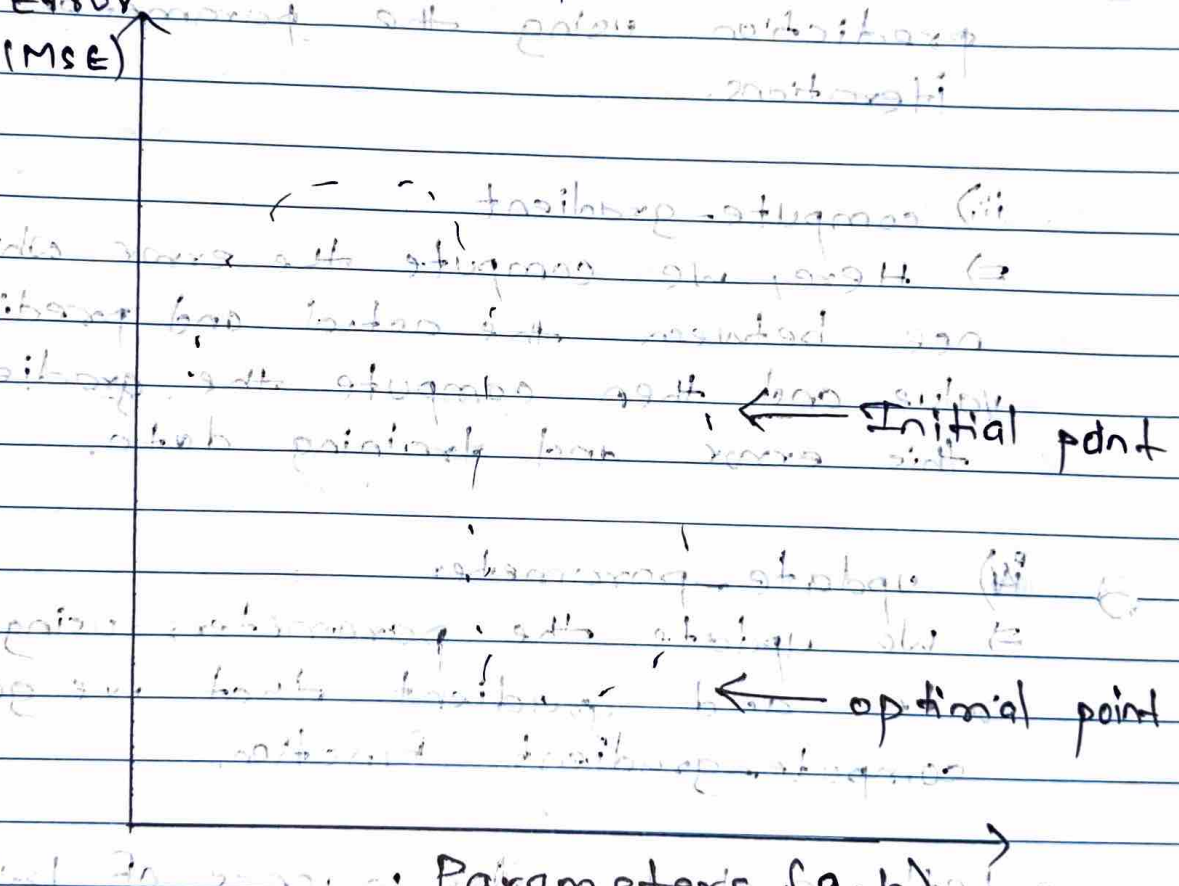
iii) compute_gradient
⇒ Here, we compute the error which is difference between the actual and predicted target value and then compute the gradient using this error and training data.

iv) update_parameter
⇒ We update the parameters using learning rate and gradient that we got from compute_gradient function.

- Let's evaluate the process of bell-shaped error curve to visualize the optimization.

- ⇒ Imagine plotting the curve error (MSE) as a function of the model parameter a and b

- The error surface might look like a bell-shaped curve.
- The minimum point on this curve represents the optimal values of a and b, where the error is minimised.
- Gradient descent moves the parameters the iteratively downhill on this surface until the minimum point is reached.



Q.4 Explain the following performance evaluation parameters with the help of confusion matrix. Illustrate using appropriate example:

a) Accuracy

⇒

- A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known.
- It consists of four outcomes :-

i) True Positive (TP) : The model correctly predicts the positive value/class.

ii) True Negative (TN) : The model correctly predicts the negative class.

iii) False Positive (FP) : The model incorrectly predicts the positive class.

iv) False Negative (FN) : The model incorrectly predicts the negative class.

- Accuracy is the ratio of correctly predicted observations to the total observations.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

b) Precision

⇒

- Precision measures the proportion of true positive predictions in the total predicted positives.
- It indicates how many of the positively classified instances were actually positive.

$$\text{Precision} = \frac{TP}{TP + FP}$$

c) Recall

⇒

- Recall measures the proportion of actual positives that were correctly identified by the model.
- It is also known as Sensitivity or True positive rate.

$$\text{Recall} = \frac{TP}{TP + FN}$$

d) F1-Score

⇒

- The F1-score is the harmonic mean of Precision and recall.
- The F1-score takes both false positive and false negative into account.

- $$F1\text{-}Score = 2 \times \frac{Precision \times recall}{Precision + Recall}$$

e) Specificity

⇒

- Specificity measure the proportion of actual negatives that were correctly identified by the model.
- It is also known as the True Negative Rate.

$$Specificity = \frac{TN}{TN + FP}$$

f) ROC-AUC Curve

⇒

- The ROC curve is a graphical representation of model's performance across all classification thresholds.
- It plots the True Positive Recall (Rate) against the False positive Rate at various threshold settings.
- The AUC provides an aggegrate measure of performance across all possible classification threshold.

- True positive Rate = $\frac{TP}{TP+FN}$

- False positive Rate = $\frac{FP}{FP+TN}$

- The higher the AUC, the better the model's ability to distinguish between positive and negative classes.

Om Shete

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Assignment No: 2

Q.1 Discuss different ensembles techniques.

⇒
- A ensemble is machine learning model that combines the predictions from two or more models.
- The models that contribute for the ensembles are called as ensemble members.
- They may be of the same type or of different types.
- Different Ensemble techniques:

i) Bagging

⇒ Bagging involves training multiple instances of the same learning algorithm on different subsets of the data and then averaging the predictions or taking a majority vote.
- Random forest is a widely used bagging technique where multiple decision trees are trained, and the final prediction is based on the majority vote or average.
- Advantages:
  - i) Reduces variance
  - ii) Helps prevent overfitting
- Disadvantage:
  - i) Computationally expensive as it requires training multiple models.

2) Boosting

⇒ Boosting works by training models sequentially, with each new models focusing on the errors made by the previous models.

- The predictions are combined in a weighted manner where the models that perform better are given higher importance.
- For eg:, AdaBoost focuses on misclassified instances and adjusts weights accordingly for the next learner.
- Advantages :

i) Reduces both bias and variance.

ii) Effective for improving the performance of weak models.

- Disadvantages :

i) Prone to overfitting

ii) Computationally intensive

3) Voting

⇒ In voting ensemble, multiple models are trained, and their predictions are combined by taking vote.

- There are two types of voting :-

a) Hard voting : Majority vote is taken for classification.

b) Soft voting : Probabilities or confidences scores are averaged to make the final

prediction.

- For eg.:
A simple voting classifiers might combine logistic regression, decision trees and SVMs and take the majority vote of their predictions

- Disadvantages:
i) Requires the base models to be resonably good.

- Advantages:
i) Easy to implement and interpret.
ii) Reduces the risk of choosing the wrong model.

Q.2 Explain following terms :

1) Weak learners and Strong learners

⇒

- **Weak learners**

⇒ A weak learner is a machine learning model that performs slightly better than random guessing, meaning its accuracy is marginally above 50% for binary classification problems.

- Individually, weak learners have low predictive accuracy.
- Weak learners often underfit the data and fail to capture the complexity of the underlying patterns.
- For e.g., A decision stump is often considered a weak learner.

- **Strong learners**

⇒ A strong learner is a model that has high predictive accuracy and is capable of making highly accurate position on its own.

- Strong learners have low bias and low error rates.
- Strong learners are often more complex models like deep decision trees, neural networks or gradient-boosted models.
- Strong learners are capable of fitting the training data well while also generalizing

to new, unseen data.
- For eg., A fully grown decision tree or a well-tuned deep learning model can be considered strong learners.

2) Meta learning
⇒
- Meta learning, often referred to as "learning to learn", involves algorithms that improve their learning processes over time by using past experiences or learning patterns.
- It focuses on learning at a higher level than conventional machine learning by adjusting the learning process itself based on previous outcomes.
- Characteristics
i) The system can adapt quickly to new tasks by leveraging prior knowledge or patterns learned from other tasks.
ii) Meta-learning models can achieve good performance with fewer data points or less computational effort by learning how to learn effectively.
- Types of Meta learning
1) Model-based Meta learning: Involve learning an internal model that can quickly adapt

to new tasks with limited fine-tuning.

ii) Metric-based Meta-learning: The algorithm learns a similarity metric that allows it to generalize well to new tasks with limited fine-tuning.

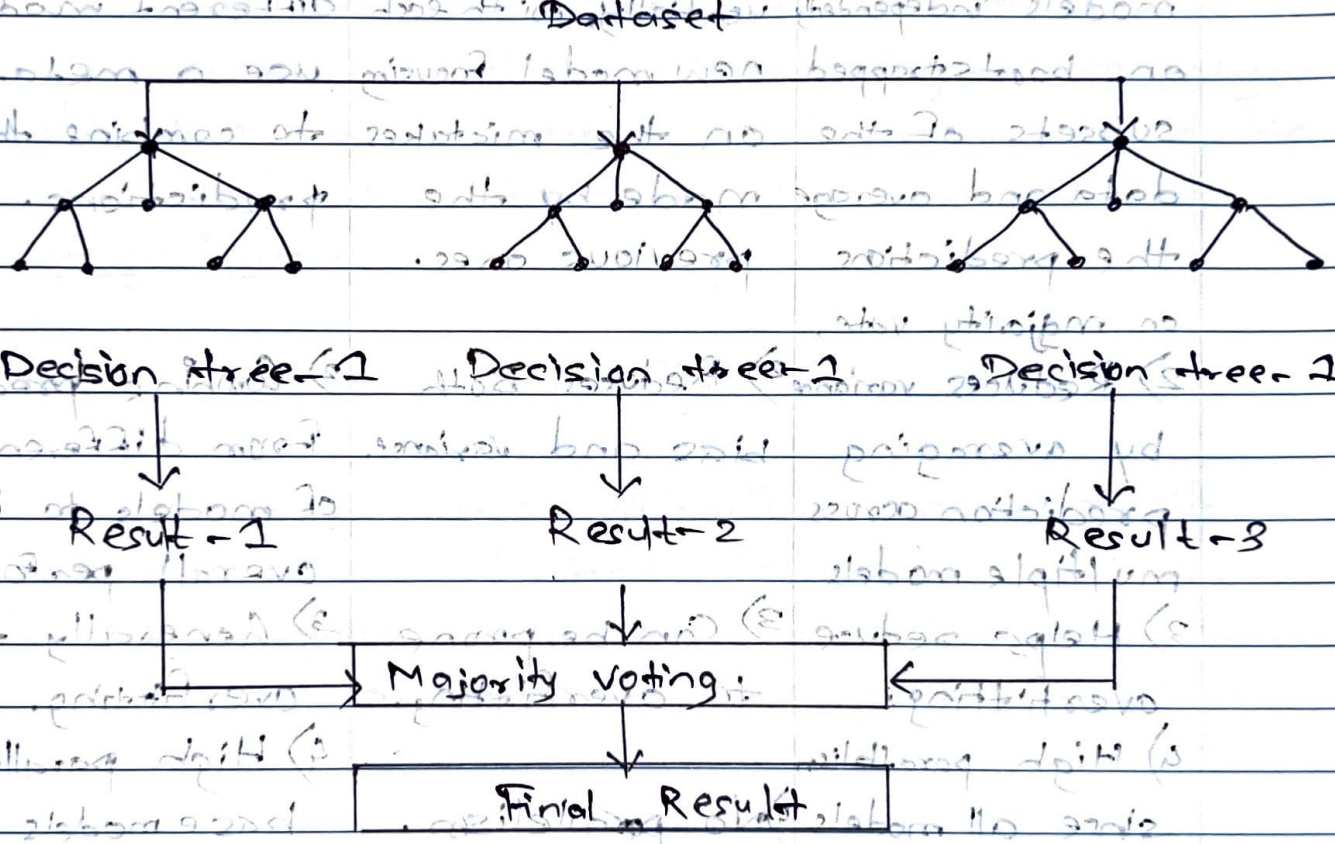
iii) Optimization-based Meta-learning: Focuses on improving the optimization process itself, such as by learning better update rules for gradient descent.

- For eg., Meta-learning is commonly used in few-shot learning scenarios, where a model is trained to adapt to new tasks with only a few training examples.

3) Random Forest

1) Random Forest is a supervised machine learning algorithm that is used widely in classification and Regression problems.
- It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

- Random Forest algorithm can handle the dataset containing continuous variables in case of regression and categorical variables in case of classification.
- It performs better result for classification problem.



- Advantages :-

1) It is robust to noise and works well with large datasets and high number of features.
2) Random forest can be used to assess the relative importance of features for the prediction.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Q.3. Compare Bagging, Boosting and stacking techniques.

A)

Bagging	Boosting	Stacking
1) Train multiple models independently on bootstrapped subsets of the data and average the predictions on majority vote.	1) Train models sequentially, with each new model focusing on the mistakes made by the previous ones.	1) Train multiple different models and use a meta-learner to combine their predictions.
2) Reduces variance by averaging prediction across multiple models	2) Reduces both bias and variance.	2) Combine prediction from different types of models to improve overall performance.
3) Helps reduce overfitting.	3) Can be prone to overfitting.	3) Generally reduces overfitting.
4) High parallelism since all models are trained independently.	4) No parallelism.	4) High parallelism, base models are trained independently
5) Models are relatively strong individually	5) Models are weak individually.	5) Base models can be both strong or weak.
6) For e.g., Random Forest, Bagged decision tree	6) For e.g. AdaBoost, XGBoost.	6) For e.g., Stacking with different base learners.

Q.4 Explain AdaBoost Algorithm

=)

- AdaBoost (Adaptive Boosting) algorithm is one of the most popular Boosting techniques that combines multiple weak learners to create strong learners.
- It focuses on improving the performance of weak classifier by giving more weight to the data points that are misclassified in each iteration.
- Algorithm:

1) Weight Initialization:

- At every start, schooling instances is assigned an identical weights.
- These weights determine the importance of every example in the getting to-know method.

2) Model Training:

- A weak learner is skilled at the dataset, with the aim of minimising class errors.
- A weak learner is usually easy model, which includes a selection stump or a small neural network.

3) Weighted Error Calculation:

- After the vulnerable learner is skilled, its miles are used to make predictions at the education dataset.
- The weighted mistakes are then calculated by means of summing up the weights of the misclassified times.

4) Model weight calculation:

- The weight of susceptible learner is calculated primarily based on their performance in classifying the training data.
- Models that perform properly are assigned higher weights, indicating that they're more reliable.

5) Update instance weight:

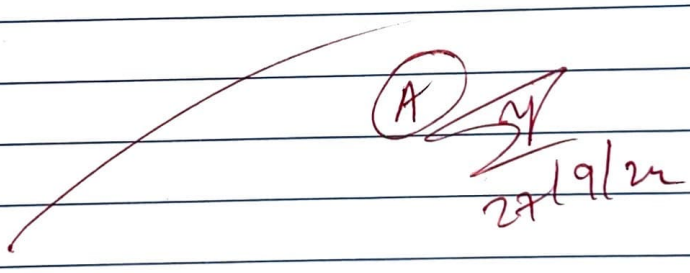
- The example weights are updated to offer more weight to the misclassified samples from the previous step.
- The adjustment focuses on the studying method at the times that the present day model struggles with.

6) Repeat :

- Steps 2 through 5 are repeated for a predefined variety of iterations or till a distinctive overall performance threshold is met.

7) Final Model Creation :

- The very last study model is created by means of combining the weighted outputs of all weak newcomers.



Om Shete

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Assignment 3

Q.1 Compare decision tree classification with K-NN classification.

⇒

K-NN Classification	Decison tree
1) Does not require specific training.	1) Require a proper training phase.
2) Zero learning, that is why called lazy algorithm	2) Builds a model based on training.
3) Data must always be available for making predictions.	3) Once training is completed training data is not needed.
4) Most interpretable algorithm.	4) It is also interpretable since we can see decision making in tree format.
5) Takes much time in decision making, since it have to traverse the whole dataset to calculate the distance with each point.	5) Once training is completed prediction time is less.
6) Since no training is required hence it zero(0)	6) Require initial training time to create decison node and branches.
7) Suitable for small size databases	7) Used for small as well as large size dataset.

Q.2 Compare decision tree classification with logistic regression classification.

Logistic Regression	Decision tree Regression
1) Logistic regression is a type of supervised learning algorithm used for classification tasks where the goal is to model predict probability that an instance belongs to given class or not.	1) A decision tree is a type of supervised learning that is commonly used in ML to model and predict outcomes based on input data.
2) Coefficients can be interpreted to understand the influence of each feature on the outcomes.	2) Highly interpreted, as the tree structure clearly shows the decision made at each node.
3) Less prone to over-fitting compared to decision tree.	3) Prone to overfitting.
4) It is majorly affected by noise.	4) It is majorly affected by noise.
5) Requires a large enough training dataset.	5) Can be trained on small training set.

Q.3 List down the attribute selection measures used by ID3 algorithm to construct a decision tree.

⇒

Attribute Selection Measures :

1) Gini Index
2) Information Gain
3) Entropy
4) Gain Ratio

1) Gini Index

⇒ All attributes are assumed to be continous valued. It is assumed that there exist several possible split values for each attribute.

- Gini index method can be modified for categorical attributes.
- Gini index is used in Classification and Regression Tree (CART).
- If a dataset T contains example from n classes, gini index, gini(T) is defined as -

$$Gini(T) = 1 - \sum_{j=1}^{n} (P_j)^2$$

- In the above equation P_j represents the relative frequency of class j in T.

- After splitting T into two subsets T_1 and T_2 with sizes N_1 and N_2, gini index of split data is,

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2).$$

2) Information Gain (IG)

⇒ In this method all attributes are assumed to be categorical.

- The amount of information required to decide if a random record in S belongs to P or N is defined as,

$$I(p,n) = -\left(\frac{p}{p+n}\right) \log_2\left(\frac{p}{p+n}\right) - \left(\frac{n}{p+n}\right) \log_2\left(\frac{n}{p+n}\right)$$

- The higher the IG, the better the attribute is at classifying the example.

3) Entropy

⇒ Entropy measures the impurity or disorder in the dataset. It is a measure of the uncertainity in the dataset's classification.

$$Entropy(S) = -\sum_{i=1}^{c} P_i \log_2 P_i$$

- To quantify how mixed or impure a set of data is.
- The higher the entropy, the more uncertain or impure the data is.

c) Gain Ration

⇒ Gain ratio is an extension of IG that takes into account the intrinsic information of a split, penalizing attribute that result into a large number of branches with small sets of data.

$$\text{Gain Ratio}(A) = \frac{\text{Information Gain}(S, A)}{\text{Split Information}(A)}$$

Q.4 Explain the properties of Gini Index

⇒

- The Gini index is a measure used in decision trees to evaluate quality of the split.
- It quantifies the degree of impurity or disorder in a dataset.
- The Gini is used in Classification and Regression Tree (CART).
- If a dataset T contains example from n classes, gini index, gini(T) is defined as-

$$\text{Gini}(T) = 1 - \sum_{j=1}^{n} (p_j)^2$$

- In the above equation, P_j represents the relative frequency of class j in T.
- After splitting T in two subsets T_1 and T_2 with sizes N_1 and N_2, gini index of split data is :

$$gini_{(split)} = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute with smallest $gini_{(split)}(T)$ is selected to split the node.
- The Gini index ranges from 0 to 0.5.
- 0 indicates perfect purity, meaning all the instances belong to a single class.
- 0.5 represents maximum impurity, indicating that the instances are uniformly distributed across all classes.

Q.3 Discuss in brief pruning in decision tree.

⇒ - Pruning in a decision tree is a technique used to reduce the size of the decision tree by removing sections of tree that provide little to no additional predictive power.

- The main goal of pruning is to prevent overfitting and improve models ability to generalize to unseen data.
- Decision trees can grow very large and complex, capturing noise and outliers in the training data, which leads to overfitting.
- Pruning helps this mitigate by simplifying the tree.
- By removing unnecessary branches, pruning reduces the complexity of the model, leading to better performance on new, unseen data.
- There are two main types of pruning in decision tree :-

1) Pre-pruning

⇒ This involves halting the growth of the tree, before it becomes too complex.

- It sets a limit on tree depth, maximum samples per leaf, or minimum information gain required for a split.

2) Post-pruning

⇒ This involves first allowing the tree to grow fully and then pruning back unnecessary branches.

The pruning techniques are:

1) Cost-complexity pruning

⇒ This method assigns a score to each subtree primarily based on its accuracy and complexity, then selects the subtree with the lowest score.

2) Reduced Error Pruning

⇒ Removes the branches that do not significantly affect the overall accuracy.

27/9/24

Assignment 4

Q.1 Explain the intuition behind Logistic Regression in detail. Is the decision boundry Linear or Non-linear in the case of a Logistic Regression model ? Also explain the impact of outliers on logistic regression.

⇒

- **Intuition behind Logistic Regression :**

- Logistic regression is supervised learning algorithm used primarily for binary classification problems.
- Despite its name, logistic regression is actually a classification algorithm, not a regression one
- The regression term is used because the model estimates probabilities, and probabilities are continuous values between 0 and 1.
- The core idea behind logistic regression is that we want to model the probability that a given input belongs to a particular class.
- This probability is modelled as sigmoid (logistic) function applied to a linear combination of input features.

- **Is the Decision Boundry Linear or Non-Linear in Logistic Regression?**

- In linear regression logistic regression, the decision boundry is linear.
- This is because the model is fundamentally

based on a linear combination of input features.

- The sigmoid function transforms the linear combination into probabilities, but the decision bounding remains linear in the feature space.
- For eg, for two feature x_1 and x_2, the decision bounding can be expressed as a line:

$$w_1 x_1 + w_2 x_2 + b = 0$$

- If you want non-linear decision boundaries, you can extend logistic regression by including polynomial feature or other types of feature transformation.
- In that case, the decision bounding can become non-linear in the original feature space.
- However, the core logistic regression model is always linear in terms of its input feature.

- **Impact of outliers on Logistic Regression:**

- Outliers are the data point's that are significantly different from the majority of the data.
- In logistic regression, outliers can have a significant impact because logistic regression is a parametric model.

Q.2 What are the assumptions made in logistic regression? Can we solve the multiclass classification problem using logistic regression? If yes then how?

=)

• Assumption Made in Logistic Regression :

1) The observations are independent

⇒ Logistic regression assumes that the observations in the dataset are independent of each other. That is, the observations should not come from repeated measurements of the same individual or be related to each other in any way.

2) There is no multicollinearity among explanatory variables.

⇒ Multicollinearity occurs when two or more explanatory variables are highly correlated to each other, such that they do not provide unique or independent information in regression model.

3) The sample size is sufficiently large

⇒ Logistic regression assumes that the sample size of the dataset is large enough to draw valid conclusions from the fitted logistic regression model.

Solving Multiclass classification problem:

- Yes, logistic regression can be extended to solve multiclass classification problems.
- This extension is done using multinomial logistic regression or by using one-vs-rest (OvR) and one-vs-one (OvO) strategies.
- Multinomial logistic regression is the direct extension of logistic regression for multiclass classification problems where the outcome's variable can have three or more categories.
- It uses softmax function to generalize logistic regression for multiple classes.
- The softmax function computes probabilities for each class label, and the label with the highest probability is chosen as the predicted class.
- In the One-vs-Rest approach, we fit one binary logistic regression classifier per class.
- For each class, we treat that class as the positive class (1) and all the other classes as the negative class (0).
- This results in multiple binary classifier.
- When predicting, we compute the probability that the sample belongs to each class and choose the class with the highest probability.

- In the one-vs-one approach, we build a binary classifier for every pair of classes.
- For each pair of classes, we train a classifier to distinguish between those two classes only.
- For a problem with K classes, we create $\frac{K(K-1)}{2}$ classifier.
- For prediction, we apply each classifier and choose the class that wins the most pairwise comparisons.

Q.3 Why is logistic regression termed as Regression and not as classification?

⇒
- Logistic regression is called "regression" even though it is used for classification because of its origins and nature of the model.
- In linear regression, we predict a continous outcome using a linear combination of the input feature:

$$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

- Logistic regression also computes a linear combination of input features, but it applies the logistic function to transform the output into a probability between 0 and 1.

$$P(y=1|X) = \frac{1}{1+e^{-(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b)}}$$

- The logistic function converts the continuous output of the linear combination into a probability, allowing it to be used for classification.
- The term "classification" refers to the final task of assigning labels (0 and 1) but logistic regression describe the method of calculating the probability of these models.
- Logistic regression doesn't directly classify; instead, it regresses on the features to estimate probabilities, and classification is the "step" that occurs based on the estimated probabilities.
- Although logistic regression is used for classification, it fits a continous decision boundary
- The outcome of the linear combination is continuous, which is then mapped to a range of 0 to 1 using the sigmoid function.
- This continuous output is a hallmark of regression models.

Q.4 Can we use Mean square Error as a cost function for logistic regression? Justify your answer.

⇒

- No, Mean Squared Error (MSE) is not appropriate as a cost function for logistic regression.

1) MSE assumes a linear relationship

⇒ MSE is primarily used in linear regression, where the relationship between the dependent and independent variables is assumed to be linear.

- It works well in this context because the predicted value and true values are both continuous, and MSE measures the squared difference between them.

2) Non-linearity of the Sigmoid Function

⇒ Logistic regression uses the sigmoid function to map the linear combination of the input features to a probability value.

$$P(y = 1 | x) = \frac{1}{1 + e^{-(w^T x + b)}}$$

- The sigmoid function is non-linear and bounded between 0 and 1, where MSE assumes the output can take on any real value.

3) MSE leads to Non-Convex cost function

⇒ When using the sigmoid function in logistic regression, the shape of the cost function derived from MSE often become non-convex.

- A non-convex cost function can make it difficult for optimization algorithms like gradient descent to find the global minimum efficiently.

4) Inappropriate Gradient with MSE

⇒ The derivate of MSE is linear which may not be suitable for updating the weights in the logistic regression since the output is not linear due to the sigmoid transformation.

- With log-loss, the gradient of the cost function is better aligned with the logistic function, resulting in more effective weight updates during optimization.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Q.5 Compare Naive Bayesian with logistic regression classifier.

=>

Naive Bayes	logistic Regression
1) Model type is generative.	1) Model type is discriminative.
2) Assumes features are independent of each other.	2) No assumption about independence.
3) Model complexity is relatively simple, computationally efficient	3) More complex, allowing for modeling intricate relationships.
4) well-suited for categorical data, effective for discrete features and text classification.	4) Versatile, can handle both numerical and categorical features.
5) Highly interpretable due to simplicity and the assumption of feature independence.	5) Good interpretability through coefficients, indicating the strengths and direction of feature relationships.
6) Performs well with small dataset.	6) May require a larger dataset to avoid overfitting.
7) Suitable for text classification, spam filtering, etc.	7) Suitable for customer churn prediction, credit scoring.

27/9/22

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE
Om Shet
C1 07
2103163

Assignment 5

Q.1 Discuss the need of SVM? Explain why they are called as optimal binary classifier?

⇒

- Support Vector Machine (SVM) is a type of supervised learning that can be used for classification or regression.
- Even if the data points are unseen, SVM classify the data properly.
- The need of SVM :-

1) Effective in high dimensional spaces

⇒ SVMs perform well in case of where the number of features is large compared to the number of samples.
- This is particularly useful in areas like text classification, image recognition, and bioinformatics, where data often has many features.

2) Robust to overfitting

⇒ SVMs aims to find the optimal separating bounding between classes with a maximum margin, making them less prone to overfitting, especially in high-dimensional spaces.

3) Handles Outliers

⇒ SVMs are relatively robust to outliers due to use of a margin and support vectors, focusing only on the most important data points near the bounday.

4) Well-suited for binary classification

⇒ SVMs are particularly effective for binary classification problems, as they aim to maximise the margin between two classes, resulting in a clear and strong decision boundry.

- Why SVMs are called Optimal Binary classification

⇒

- SVMs are termed as optimal binary classifier because they focus on finding the most effective way to separate two classes by constructing an optimal hyperplane.
- This hyperplane is positioned in such a way that it maximizes the margin between the two classes - this margin being the distance between the hyperplane and the closest data points from each class, known as support vectors.
- By maximizing this margin, SVM ensures that the model is not only accurate but also generalizes well to new data, minimizing the risk of overfitting.
- Thus, SVM are known for their optimality in achieving both high accuracy and good generalization, especially in binary classification problem.

Q.2 Explain the following terminologies with the help of appropriate illustrations:

i) Optimal Decision Boundary

⇒

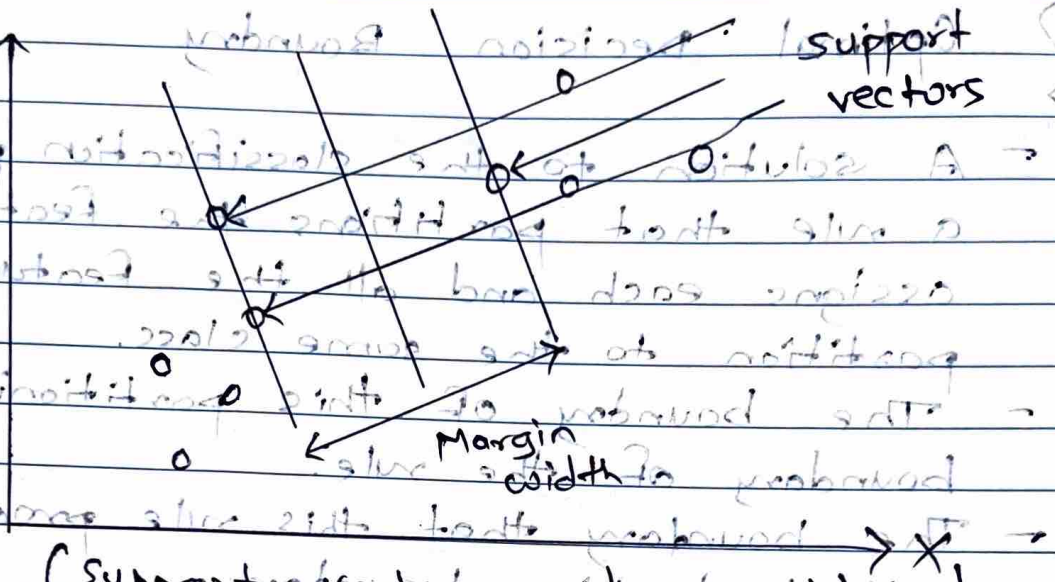
- A solution to the classification problem is a rule that partitions the features and assigns each and all the features and partition to the same class.
- The boundary of this partitioning is decision boundary of the rule.
- The boundary that this rule produces is the optimal decision boundary.
- A decision boundary is a region of a problem space in which the output label of a classifier is ambiguous.
- If the decision surface is a hyperplane, then classification is linear, and the classes are linearly separable.

ii) Support Vectors

⇒

- Support vectors are the datapoints that lie closest to the decision boundary or hyperplane
- These points are critical because they determine the position of the optimal hyperplane.
- The decision boundary is determined based only on these support vectors, while other points do not influence its position.

- Support vectors are essential in SVM because they help maximize the margin between two classes.



(Support vectors and Margin)

ii) Margins

⇒

- Margin refers to the distance between the decision boundary and the nearest data points of each class.
- SVM aims to maximize margin, which leads to a robust decision boundary.
- A larger margin implies a more confident and generalizable classifier, reducing the risk of overfitting.
- The margin is measured from the support vectors to the decision boundary, and maximizing it ensures the model's optimal performance.

Q.3 What do you understand by linear classifiers and non-linear classifiers? Can you use SVM as non-linear classifier? If yes, explain how SVM can be used as non-linear classifier.

⇒

- **Linear classifier**

⇒ A linear classifier is a classification algorithm that makes its predictions based on linear combination of input features.

- The decision boundary it creates to separate different classes is a straight line, a plane, or a hyperplane.

- **Non-linear classifier**

⇒ A non-linear classifier is one that can draw a more complex decision boundary to separate different classes, which isn't just a straight line or hyperplane.

- These classifiers are used when the data is not linearly separable.

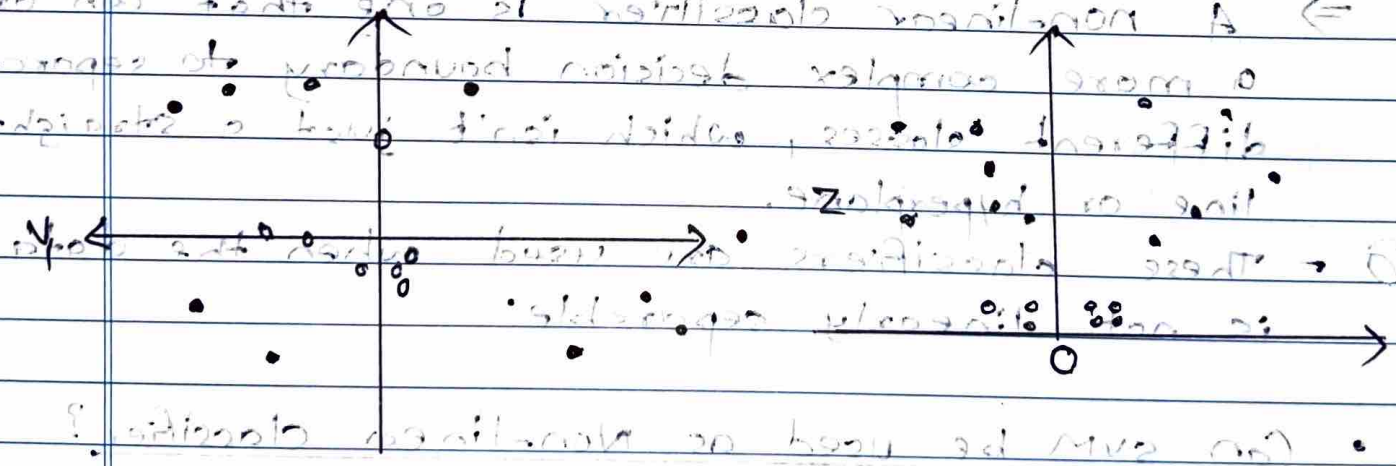
- **Can SVM be used as Non-linear classifier?**

⇒ Yes, SVM can be used as non-linear classifier

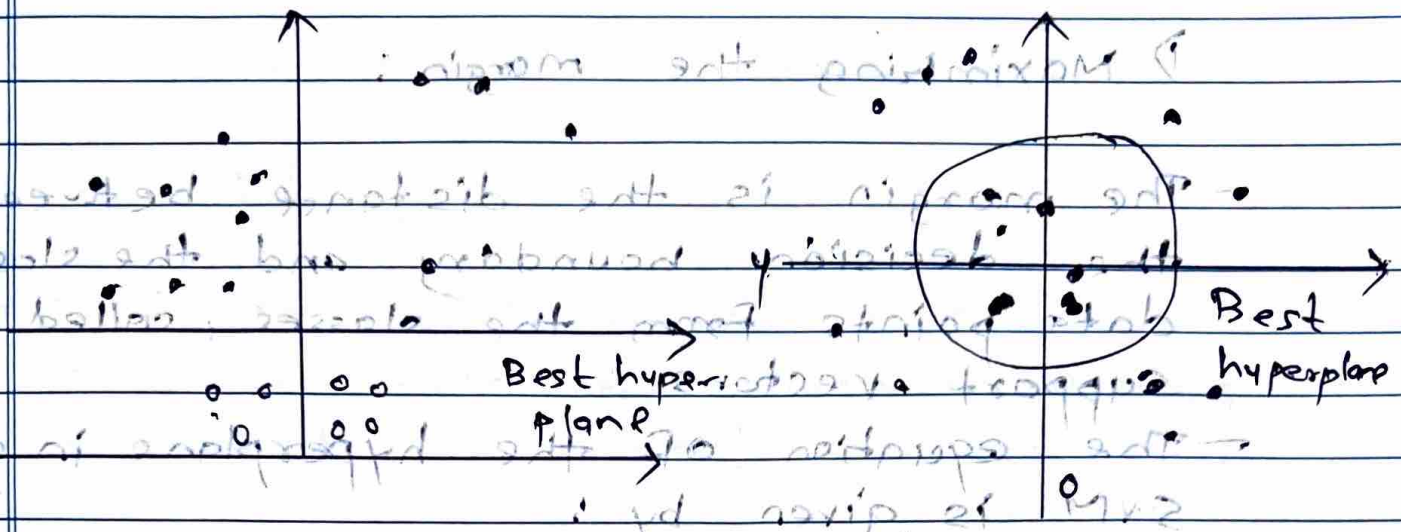
- Although the basic SVM algorithm is linear classifier, it can be extended to handle non-linear classification tasks through a method called the kernel trick.

Non-linear SVM

- If a data is linear, we can separate it by using a straight line, but if data is non-linear, then we cannot draw a single straight line.
- So to separate these data points, we need one more dimension.
- For linear data, we have used two dimensions x and y so for non-linear data, we add a third dimension z.
- It can be calculated as $z = x^2 + y^2$
- After adding the third dimension, the sample space becomes:



- Now, SVM will divide the datasets into classes in the following way:



- If we put $z=1$, then it will become as above,
  $x^2 + y^2 = 1$.
- We get a circumference of radius 1 as hyperplane surface in case of non-linear data.

Q.4 Express SVM as a constrained optimization problem. Discuss how predictions can be done by using SVM.

⇒
- SVMs aims to find the optimal separating hyperplane that maximizes the margin between two classes in dataset.
- To achieve this, SVMs are formulated as a constrained optimization problem.
- constrained optimization is a set of method to identify efficiently and systematically the best solution to a problem.
- The problem is characterized by a number of potential solution in the presence of identified constrained.

1) Maximizing the margin:

- The margin is the distance between the decision boundary and the closest data points from the classes, called support vectors.
- The equation of the hyperplane in an SVM is given by:

$$w \cdot x + b = 0$$

w = weight vector
b = bias (offset)
x = input vector.

- The decision boundary separates two classes such that:

$$w \cdot x_i + b \geq 1, \text{ for } y_i = +1$$
$$w \cdot x_i + b \leq -1, \text{ for } y_i = -1$$

2) Constrained Optimization problem:

- Thus, SVM optimization problem can be written as:

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

- Subject to the constraints :

$$y_i (w \cdot x_i + b) \geq 1 \quad \forall i$$

- This is a convex optimization problem because both the objective function and constraints are convex.

3) Soft Margin SVM

⇒

- If the data is not perfectly separable, we introduce slack variables ξ_i to allow for some misclassifications.
- The optimization problem becomes :

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i$$

- Subject to :

$$y_i (w \cdot x_i + b) \geq 1 - \xi_i \quad \forall i$$

$$\xi_i \geq 0 \quad \forall i$$

Q.5 What are kernel functions? List some kernal functions. What is their use in SVM?

⇒

- Kernel functions is a method used to take data as input and transform it into the required form of processing data.
- "Kernel" is used due to set of mathematical functions used in SVM providing the window to manipulate the data.
- So, Kernel function generally transforms the training set of data so that a non-linear decision surface is able to transform to a linear equation in a higher number of dimension spaces.
- Basically, it returns the inner product between two points in a standard feature dimension.
- Some common kernel functions :-

1) Linear Kernel

⇒ This kernel is used when the data is linearly separable in the original space.

- It's formula is:

$$K(x_i, x_j) = x_i \times x_j$$

2) Polynomial Kernel

⇒ The polynomial kernel allows SVM to create non-linear decision boundaries by using polynomial functions of the input data.

→ $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$

3) Gaussian kernel

⇒ The guassian kernel maps the data into an infinite-dimensional space, allowing for highly complex decision boundaries.

→ $K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$

4) Sigmoid kernel

⇒ This kernel function resembles the activation function of a neural network.

→ $K(x_i, x_j) = \tanh(\alpha(x_i \cdot x_j) + r)$

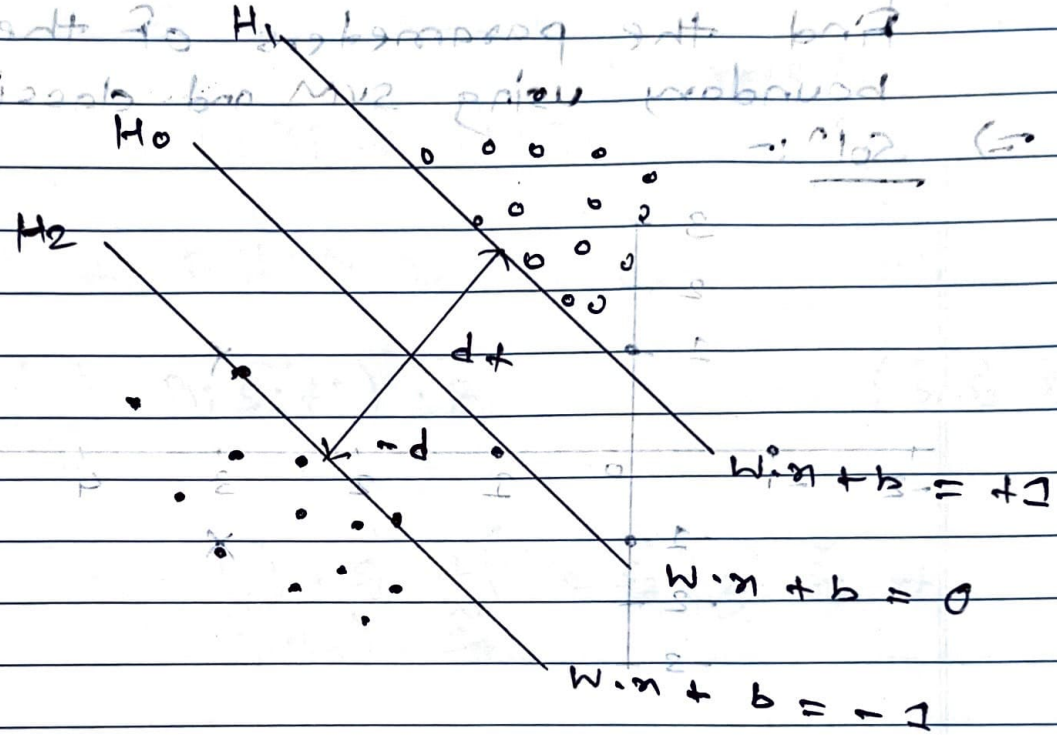
→ When the relationship between features is similar to the neural network structure or in case where sigmoid-like non-linearities are expected.

Q. 6 Explain the concept of kernel trick? Discuss with example, how kernel tricks can speed up the computations.

⇒ - A kernel trick is a simple method where a Non-linear data is projected onto a higher dimensional space so as to make it easier to classify the data. Then it is linearly divided by a plane.

- This is mathematically achieved by Lagragian formula using Lagrangian multipliers.
- Our basic idea of kernel trick is to find the plane which can separate, classify or split the data with maximum margin is also called street width.
- The distance from the point (x, y) to a line $Ax + By + C = 0$ is $\frac{|Ax + By + C|}{\sqrt{A^2 + B^2}}$
- In order to maximize the margin, the distance between H_0 and H_1 is then $\frac{|w \cdot x + b|}{\|w\|} = \frac{1}{\|w\|}$, so the total distance between H_1 and H_2 is $\frac{2}{\|w\|}$
- In order to maximize the margin, we need to minimise $\|w\|$.
- Thus we are trying to optimize the margin or street width by maximising the distance by maximising distance

between the support vectors.



7. How Kernel Trick Speeds up Computation :-

1) Avoid Explicit transformation

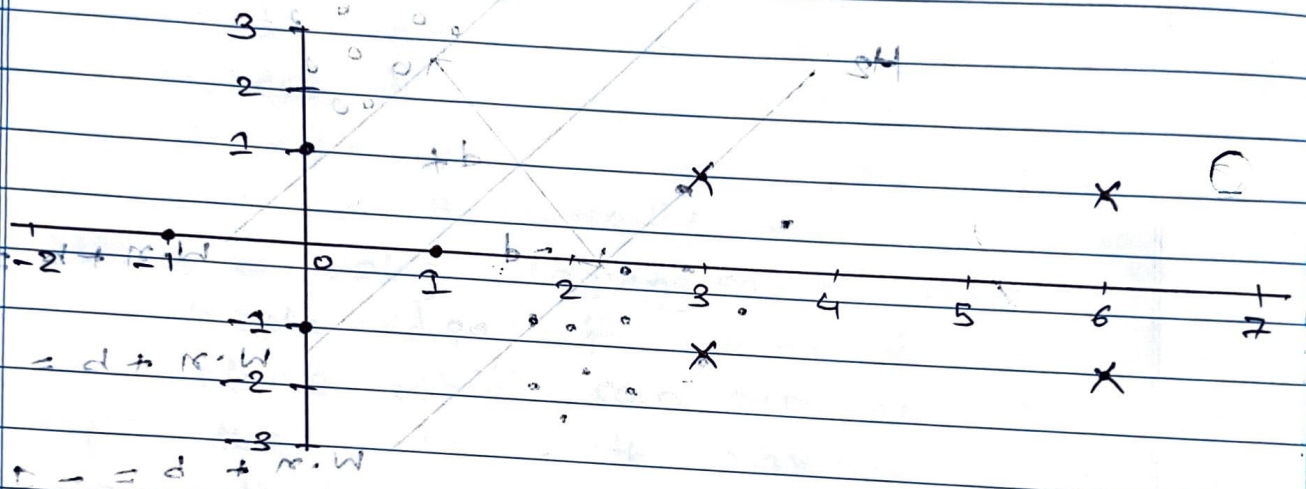
⇒ In cases where the transformation $\phi(x)$ maps the data into the high-dimensional or even infinite-dimensional space, explicitly calculating the transformed data is impractical.

2) Efficient Computation

⇒ The kernel trick allows SVM to compute the dot product between data points in the transformed space without the need to explicitly calculate the coordinates of the data in that space.

Q.7 Give '+'ly labelled data points as {(3,1), (3,-1), (6,1), (6,-1)} and '-'ly labelled data points as {(1,0), (0,1), (0,-1), (-1,0)} Find the parameters of the decision boundary using SVM and classify point (1,3).

⇒ Sol^n :-



- The circles are negatively labelled data points and cross are positively labelled data points.
- There are three support vectors -

$S_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $S_2 = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$, $S_3 = \begin{pmatrix} 3 \\ -1 \end{pmatrix}$

- After adding bias input to S_1, S_2, S_3 we get,

$\tilde{S}_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$, $\tilde{S}_2 = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}$, $\tilde{S}_3 = \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix}$

- Now,

$$D(z) = w^T z + b$$

$$= (\Sigma \lambda_i \tilde{s}_i t_i) \cdot z + b$$

$$= \Sigma \quad \text{(ignore b)}$$

$$\therefore (D(z)) = (\Sigma \lambda_i \tilde{s}_i t_i) \cdot z \quad \text{— (1)}$$

- We need to find $\lambda_1, \lambda_2, \lambda_3$ using eq (1).

$$y_i = \left(\sum_{i=1}^{3} \lambda_i \tilde{s}_i t_i\right) \cdot z \quad , \quad z \in (\tilde{s}_1, \tilde{s}_2, \tilde{s}_3)$$

$$\therefore y_1 = (\lambda_1 \tilde{s}_1 t_1) \cdot \tilde{s}_1 + (\lambda_2 \tilde{s}_2 t_2) \cdot \tilde{s}_2 + (\lambda_3 \tilde{s}_3 t_3) \cdot \tilde{s}_3$$

- Now, $y_1 = 1$

$$\lambda_1 (\tilde{s}_1 \cdot \tilde{s}_1) t_1 + \lambda_2 (\tilde{s}_2 \cdot \tilde{s}_1) t_2 + \lambda_3 (\tilde{s}_3 \cdot \tilde{s}_1) t_3 = -1$$
$$\lambda_1 (\tilde{s}_1 \cdot \tilde{s}_2) t_1 + \lambda_2 (\tilde{s}_2 \cdot \tilde{s}_2) t_2 + \lambda_3 (\tilde{s}_3 \cdot \tilde{s}_2) t_3 = 1$$
$$\lambda_1 (\tilde{s}_1 \cdot \tilde{s}_3) t_1 + \lambda_2 (\tilde{s}_2 \cdot \tilde{s}_3) t_2 + \lambda_3 (\tilde{s}_3 \cdot \tilde{s}_3) t_3 = 1$$

- Putting the values in above equations,

$$\lambda_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} t_1 + \lambda_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} t_2 + \lambda_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} t_3$$

$$= -1$$

$$2\lambda_1 + 4\lambda_2 + 4\lambda_3 = -1 \quad \text{— (2)}$$

- Similarly solving for rest two equations, we get -

$$4\lambda_1 + 11\lambda_2 + 9\lambda_3 = 1 \quad -(3)$$
$$4\lambda_1 + 9\lambda_2 + 11\lambda_3 = 1 \quad -(4)$$

Now, solving eq (2), (3), (4) we get,

$$\lambda_1 = -3.5$$
$$\lambda_2 = 0.75$$
$$\lambda_3 = 0.75$$

$$\therefore \tilde{w} = \sum_{i=1}^{3} \lambda_i \cdot s_i \cdot t_i$$

$$\therefore \tilde{w} = -3.5 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} \begin{matrix} \rightarrow w \\ \rightarrow b \end{matrix}$$

- Since the support vectors were augmented with bias, ~~entries~~, entries in $\tilde{w}$ ~~comp~~ correspond to hyperplane with offset b,

where in, $y = w^T x + b$,

$W = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $b = -2$.

- The point (1, 3) is less than 2, since the decision boundary is $x_1 = 2$, this point lies on the negative side of the decision boundary.

11/10/22