Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that Mr./~~Miss~~ Harslik Bhagat

of COMPS Department, Semester VII with

Roll No. 17 has completed a course of the necessary

experiments in the subject ML under my

supervision in the **Thadomal Shahani Engineering College**

Laboratory in the year 2024 - 2025

Teacher In-Charge

Head of the Department

Date 14/10/24

Principal

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1]	Data Pre steps, Input, Normalisation etc	3	11/7/24	
2]	Exp on GDA	8	18/7/24	
3]	Exp on simple LR	13	25/7/24	
4]	Exp on LR	19	8/08/24	
5]	Implement Decision Tree	24	22/8/24	
6]	SVM	31	29/8/24	14/10/2
7]	Ensemble Methods	36	5/9/24	
8]	PCA	42	26/9/24	
9]	Mini Project	46	26/9/24	
10]	Asign 1	56	25/7/24	
11]	Asign 2	68	3/10/24	

THADOMAL SHAHANI

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Experiment No. 1

Aim: Select Appropriate dataset & perform the following preprocessing steps

1) Imputation it is the process of replacing missing values in the dataset there are several techniques to handle missing values

a) Mean/Median/Mode : Imputation replace missing values with the mean, median, mode of the respective column.

b) Forward and Backward fill : Fill missing values using the previous or next observation.

c) Interpolation use linear or polynomial interpolation to estimate missing values.

d) K-nearest : Neighbours Use the KNN algorithm to impute missing values based on the values of their nearest neighbour

2) Anomaly Detection: It involves identifying or abnormal data points in the datasets.

a) Statistical Methods

b) Machine learning models

c) time series methods

3) Standardization :- Transforms data to have a mean of standard deviation of one useful for algorithms that assume normally distributed data

4] Normalization :- Transform data to have a mean of zero & standard deviations of one useful for algorithms that assume normally, which is useful for algorithms that rely on dist measurement.

5] Encoding : used to convert categorical data into numerical format

a) Label encoding
b) One-hot encoding
c) Ordinal encoding
d) Frequency encoding

(A) Status

EXPERIMENT 1

In [19]:
```
import pandas as pd

# Load the dataset using a raw string
df = pd.read_csv(r'C:\Users\hp\OneDrive\Desktop\housing.csv')
print("First 5 rows of the dataset:")
print(df.head())
```

```
First 5 rows of the dataset:
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished
```

IMPUTATION

In [20]:
```
from sklearn.impute import SimpleImputer

# Separate numerical and categorical columns
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns

# Impute missing values for numerical columns with mean
num_imputer = SimpleImputer(strategy='mean')
df[numerical_cols] = num_imputer.fit_transform(df[numerical_cols])

# Impute missing values for categorical columns with mode
cat_imputer = SimpleImputer(strategy='most_frequent')
df[categorical_cols] = cat_imputer.fit_transform(df[categorical_cols])

print("First 5 rows after imputation:")
print(df.head())
```

```
First 5 rows after imputation:
        price    area  bedrooms  bathrooms  stories mainroad guestroom  \
0  13300000.0  7420.0       4.0        2.0      3.0      yes        no
1  12250000.0  8960.0       4.0        4.0      4.0      yes        no
2  12250000.0  9960.0       3.0        2.0      2.0      yes        no
3  12215000.0  7500.0       4.0        2.0      2.0      yes        no
4  11410000.0  7420.0       4.0        1.0      2.0      yes       yes

  basement hotwaterheating airconditioning  parking prefarea furnishingstatus
0       no              no             yes      2.0      yes        furnished
1       no              no             yes      3.0       no        furnished
2      yes              no              no      2.0      yes   semi-furnished
3      yes              no             yes      3.0      yes        furnished
4      yes              no             yes      2.0       no        furnished
```

ANOMALY DETECTION

In [25]:
```
from sklearn.ensemble import IsolationForest

# Verify the column name and use it in anomaly detection
print(df.columns)  # Print column names to identify the correct one

# Choose a numerical column for anomaly detection, e.g., 'LotArea'
# Replace 'LotArea' with the correct column name if it differs
column_to_check = 'LotArea'

# Ensure the column exists in the DataFrame
if column_to_check in df.columns:
    clf = IsolationForest(contamination=0.05)
    df['anomaly'] = clf.fit_predict(df[[column_to_check]])
```

```
    # Display rows marked as anomalies
    anomalies = df[df['anomaly'] == -1]
    print("Anomalous rows:")
    print(anomalies.head())
else:
    print(f"Column '{column_to_check}' not found in DataFrame.")
```

```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
Column 'LotArea' not found in DataFrame.
```

STANDARDIZATION

In [22]:
```
from sklearn.preprocessing import StandardScaler

# Standardize numerical columns
scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

print("First 5 rows after standardization:")
print(df.head())
```

```
First 5 rows after standardization:
      price      area  bedrooms  bathrooms   stories mainroad guestroom  \
0  4.566365  1.046726  1.403419   1.421812  1.378217      yes        no
1  4.004484  1.757010  1.403419   5.405809  2.532024      yes        no
2  4.004484  2.218232  0.047278   1.421812  0.224410      yes        no
3  3.985755  1.083624  1.403419   1.421812  0.224410      yes        no
4  3.554979  1.046726  1.403419  -0.570187  0.224410      yes       yes

  basement hotwaterheating airconditioning   parking prefarea furnishingstatus
0       no              no             yes  1.517692      yes        furnished
1       no              no             yes  2.679409       no        furnished
2      yes              no              no  1.517692      yes   semi-furnished
3      yes              no             yes  2.679409      yes        furnished
4      yes              no             yes  1.517692       no        furnished
```

NORMALIZATION

In [23]:
```
from sklearn.preprocessing import MinMaxScaler

# Normalize numerical columns
normalizer = MinMaxScaler()
df[numerical_cols] = normalizer.fit_transform(df[numerical_cols])

print("First 5 rows after normalization:")
print(df.head())
```

```
First 5 rows after normalization:
      price      area  bedrooms  bathrooms   stories mainroad guestroom  \
0  1.000000  0.396564       0.6   0.333333  0.666667      yes        no
1  0.909091  0.502405       0.6   1.000000  1.000000      yes        no
2  0.909091  0.571134       0.4   0.333333  0.333333      yes        no
3  0.906061  0.402062       0.6   0.333333  0.333333      yes        no
4  0.836364  0.396564       0.6   0.000000  0.333333      yes       yes

  basement hotwaterheating airconditioning   parking prefarea furnishingstatus
0       no              no             yes  0.666667      yes        furnished
1       no              no             yes  1.000000       no        furnished
2      yes              no              no  0.666667      yes   semi-furnished
3      yes              no             yes  1.000000      yes        furnished
4      yes              no             yes  0.666667       no        furnished
```

ENCODING

In [26]:
```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Load the dataset
df = pd.read_csv(r'C:\Users\hp\OneDrive\Desktop\housing.csv')  # Update the path if necessary

# Strip leading and trailing spaces from column names, if any
df.columns = df.columns.str.strip()

# Separate categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
```

```python
# Initialize the OneHotEncoder with sparse_output=False
encoder = OneHotEncoder(sparse_output=False, drop='first')  # Use sparse_output instead of sparse

# Fit and transform the categorical columns
encoded_features = encoder.fit_transform(df[categorical_cols])

# Create DataFrame for encoded features and concatenate with original DataFrame
encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_out(categorical_cols))
df_encoded = pd.concat([df.drop(categorical_cols, axis=1), encoded_df], axis=1)

print("First 5 rows after encoding:")
print(df_encoded.head())
```

```
First 5 rows after encoding:
      price  area  bedrooms  bathrooms  stories  parking  mainroad_yes  \
0  13300000  7420         4          2        3        2           1.0
1  12250000  8960         4          4        4        3           1.0
2  12250000  9960         3          2        2        2           1.0
3  12215000  7500         4          2        2        3           1.0
4  11410000  7420         4          1        2        2           1.0

   guestroom_yes  basement_yes  hotwaterheating_yes  airconditioning_yes  \
0            0.0           0.0                  0.0                  1.0
1            0.0           0.0                  0.0                  1.0
2            0.0           1.0                  0.0                  0.0
3            0.0           1.0                  0.0                  1.0
4            1.0           1.0                  0.0                  1.0

   prefarea_yes  furnishingstatus_semi-furnished  furnishingstatus_unfurnished
0           1.0                              0.0                           0.0
1           0.0                              0.0                           0.0
2           1.0                              1.0                           0.0
3           1.0                              0.0                           0.0
4           0.0                              0.0                           0.0
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

THADOMAL SHAHANI
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment No. 2

Aim: Implement Gradient Descent Algorithm to minimize give obj function

Theory: Gradient Descent is an optimization algorithm used to minimize a given objective function particularly in machine learning optimization problems It is an iterative approach that seeks to find the minimum of a function by updating parameters in the direction

1] Objective function:-
- the Objective function, also know as the cost function or loss function measures how good or bad a set of parenthesis is the goal of decent is to find the set of parameters that minimizes this function

2] Initialization :- You start by initializing the parameters of the model with some initial values

3] Gradient Calculation:- Calculate the gradient of the objective function with some initial values the gradient is a vector that points in a direction of the increases of the function. In the context of optimization you need to move in the opposite direction to reach the minimum.

4) Update parameter : Update the parameters by moving them a small step in the direction of the negative gradient this step in the direction of the negative gradient this step size is determination by a parameter called the leader

5) Iteration : Repeat step 3 & 4 until a stopping criterion is met. The criteria would be maximum number of iterations the change in the objective function very small or some other measure indicating convergence.

A 5/9/24

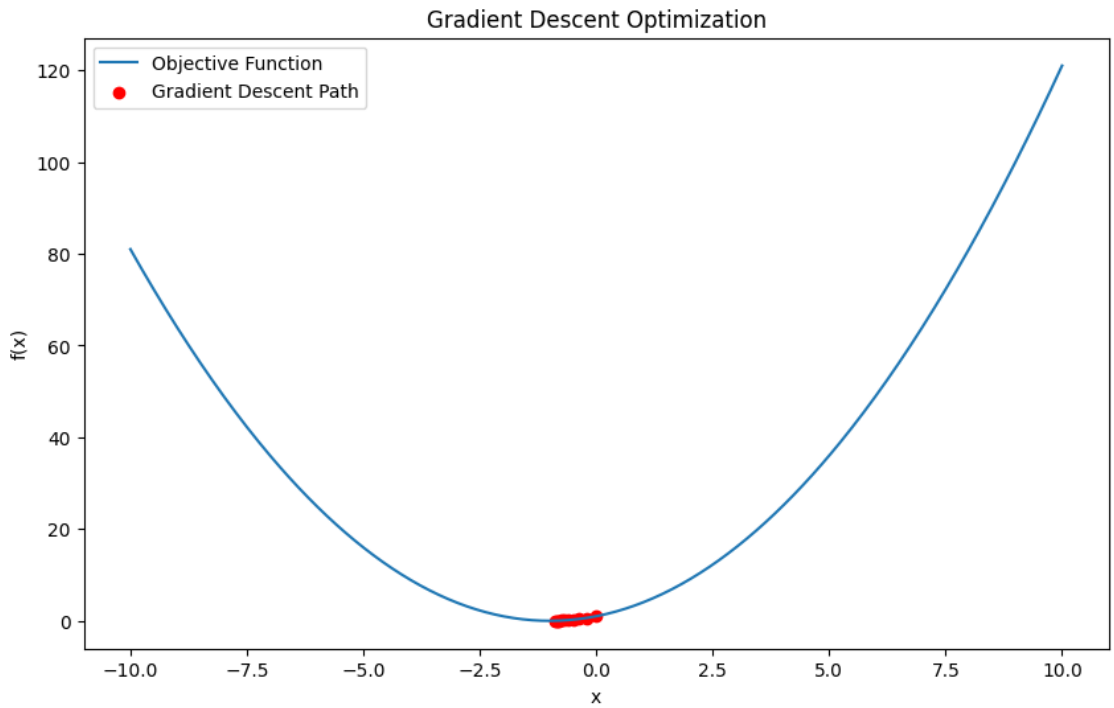
exp2-gradient-descent

[ ]:

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

# Objective function: f(x) = ax^2 + bx + c
def objective_function(x, a=1, b=2, c=1):
    return a * x**2 + b * x + c

# Gradient of the objective function: df/dx = 2ax + b
def gradient(x, a=1, b=2):
    return 2 * a * x + b

# Gradient Descent Algorithm
def gradient_descent(initial_x, learning_rate, num_iterations, a=1, b=2, c=1):
    x = initial_x
    history = [x]

    print(f"{'Iteration':<10}{'x':<10}{'Gradient':<15}{'Objective Function'}")
    print("-" * 45)

    for i in range(num_iterations):
        grad = gradient(x, a, b)
        obj_value = objective_function(x, a, b, c)
        print(f"{i+1:<10}{x:<10.4f}{grad:<15.4f}{obj_value:<20.4f}")

        x = x - learning_rate * grad
        history.append(x)

    return x, history

# Parameters
initial_x = 0  # Starting point
learning_rate = 0.1  # Step size
num_iterations = 10  # Number of iterations for demonstration

# Run Gradient Descent
```

```python
optimal_x, history = gradient_descent(initial_x, learning_rate, num_iterations)

print(f"\nOptimal x: {optimal_x:.4f}")
print(f"Value of the objective function at optimal x:␣
↪{objective_function(optimal_x):.4f}")

# Plot the objective function and the path taken by Gradient Descent
x_values = np.linspace(-10, 10, 400)
y_values = objective_function(x_values)

plt.figure(figsize=(10, 6))
plt.plot(x_values, y_values, label='Objective Function')
plt.scatter(history, [objective_function(x) for x in history], color='red',␣
↪label='Gradient Descent Path')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gradient Descent Optimization')
plt.legend()
plt.show()
```

```
Iteration x         Gradient       Objective Function
----------------------------------------------
1         0.0000    2.0000         1.0000
2         -0.2000   1.6000         0.6400
3         -0.3600   1.2800         0.4096
4         -0.4880   1.0240         0.2621
5         -0.5904   0.8192         0.1678
6         -0.6723   0.6554         0.1074
7         -0.7379   0.5243         0.0687
8         -0.7903   0.4194         0.0440
9         -0.8322   0.3355         0.0281
10        -0.8658   0.2684         0.0180

Optimal x: -0.8926
Value of the objective function at optimal x: 0.0115
```


Gradient Descent Optimization
Objective Function
Gradient Descent Path
f(x)
x
120
100
80
60
40
20
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

THADOMAL SHAHANI
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment No. 3

Aim :- Implement Simple Linear Regression

Theory :-

Linear Regression model studies the relationship between a single dependent variable 'Y' & one or more independent variable 'X' using a best fit straight line function
A line can be charterist by the slope intercept with one of the axis. Hence a given data points goal is to find a line which indicates finding slope & intercept

$$Y = B_0 + B_1 x$$

Where ϵ . random error associated with data point so we assume underlying function from which the data is generated $y_0 = B_0 + B_1 x + \epsilon$. the given data points, goal is to find out B_0 & B_1

$$Y = B_0 + B_1 x_1 + \epsilon$$

where B_0 = population of y - intercept
B_1 = population slope
ϵ = random error

eg

height	weight	$x_i - x$	$y_i - y$	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \ldots$
131	6.3	2.8	-2.3	6.4	7.84
144	81	20.2	20.2	317.14	407.04
138	56	-15.8	-15.8	146.96	249.67
186	91	32.2	32.2	827	1036.8
128	47	-25.8	-25.8	472	656.6
136	57	-17.8	-17.8	147	116
149	76	25.2	25.2	61	84.64
163	72	9.2	9.2	54	8464
152	62	-1.8	-1.8	44	3.24
131	48	-22.8	-27.8	264	3127.6
153	65				

formula $b_1 = \frac{2649.6}{3927.6}$

$= 0.6746$

(3) $b_0 = \frac{1}{n}(\Sigma y_i - b_1 \Sigma x_i) = \bar{y} - b\bar{x}$

$= 65 - 3(0.6743 \times 153.8)$

$= -38.4551$

$= 0.6742x - 38.4551$

$y = 0.6742x - 38.4551$

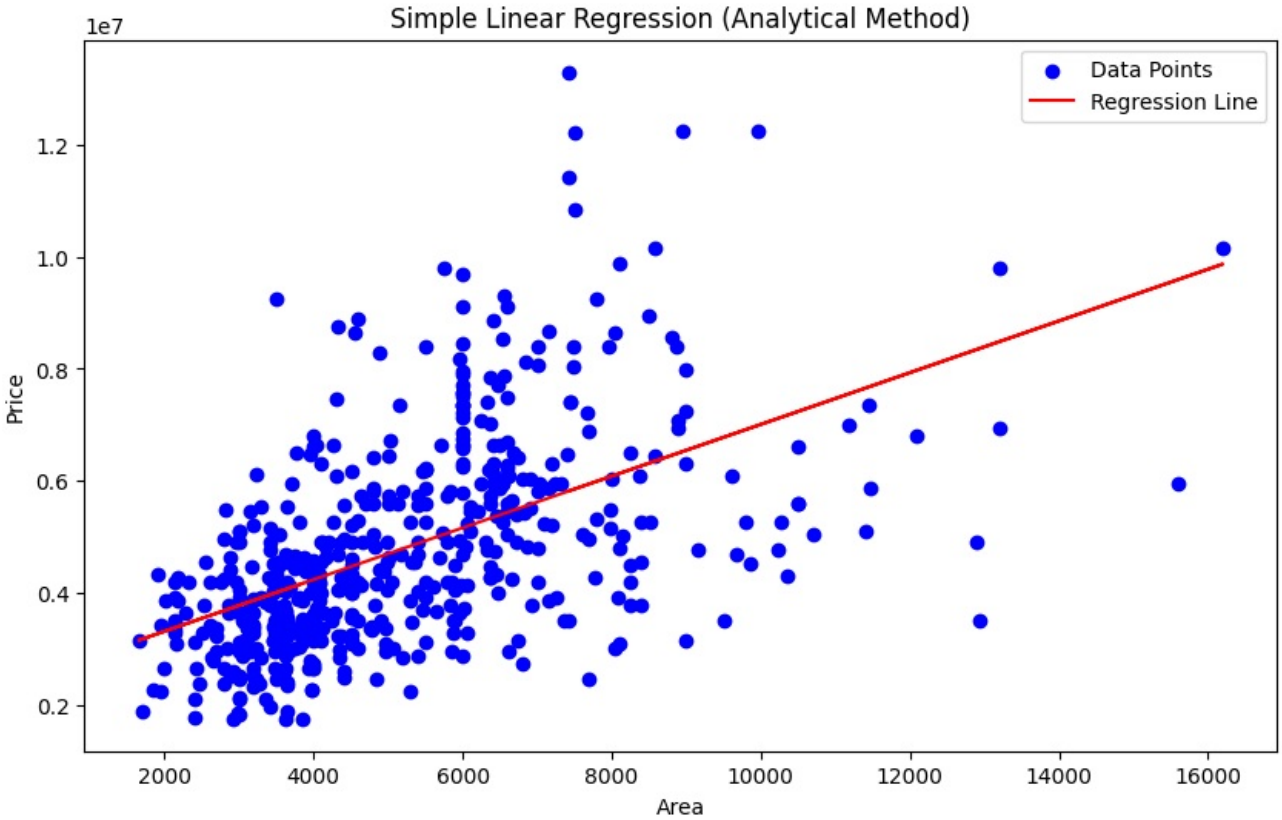
A+ 5/9/24

EXPERIMENT 3

Analytical Method

In [8]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv(r'C:\Users\hp\OneDrive\Desktop\housing.csv')  # Update the path to your dataset

# Select relevant columns (Area as the independent variable and Price as the dependent variable)
X = df['area'].values
y = df['price'].values

# Add a column of ones to include the intercept (bias) in the model
X_b = np.c_[np.ones((len(X), 1)), X]

# Analytical method: theta = (X^T * X)^-1 * X^T * y
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)

print("Optimal parameters (theta):", theta_best)

# Make predictions
y_pred = X_b.dot(theta_best)

# Plotting the data points and the regression line
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.xlabel('Area')
plt.ylabel('Price')
plt.title('Simple Linear Regression (Analytical Method)')
plt.legend()
plt.show()

# Print steps of analytical solution
print("\nSteps of Analytical Solution:")
print("1. Compute X^T * X:")
X_T_X = X_b.T.dot(X_b)
print(X_T_X)
print("\n2. Compute (X^T * X)^-1:")
X_T_X_inv = np.linalg.inv(X_T_X)
print(X_T_X_inv)
print("\n3. Compute X^T * y:")
X_T_y = X_b.T.dot(y)
print(X_T_y)
print("\n4. Compute theta = (X^T * X)^-1 * X^T * y:")
print(theta_best)
```

```
Optimal parameters (theta): [2.38730848e+06 4.61974894e+02]
```




```
Steps of Analytical Solution:
1. Compute X^T * X:
[[5.45000000e+02 2.80704500e+06]
 [2.80704500e+06 1.70197757e+10]]

2. Compute (X^T * X)^-1:
[[ 1.21894053e-02 -2.01037956e-06]
 [-2.01037956e-06  3.90323938e-10]]

3. Compute X^T * y:
[2.59786744e+09 1.45639914e+13]

4. Compute theta = (X^T * X)^-1 * X^T * y:
[2.38730848e+06 4.61974894e+02]
```

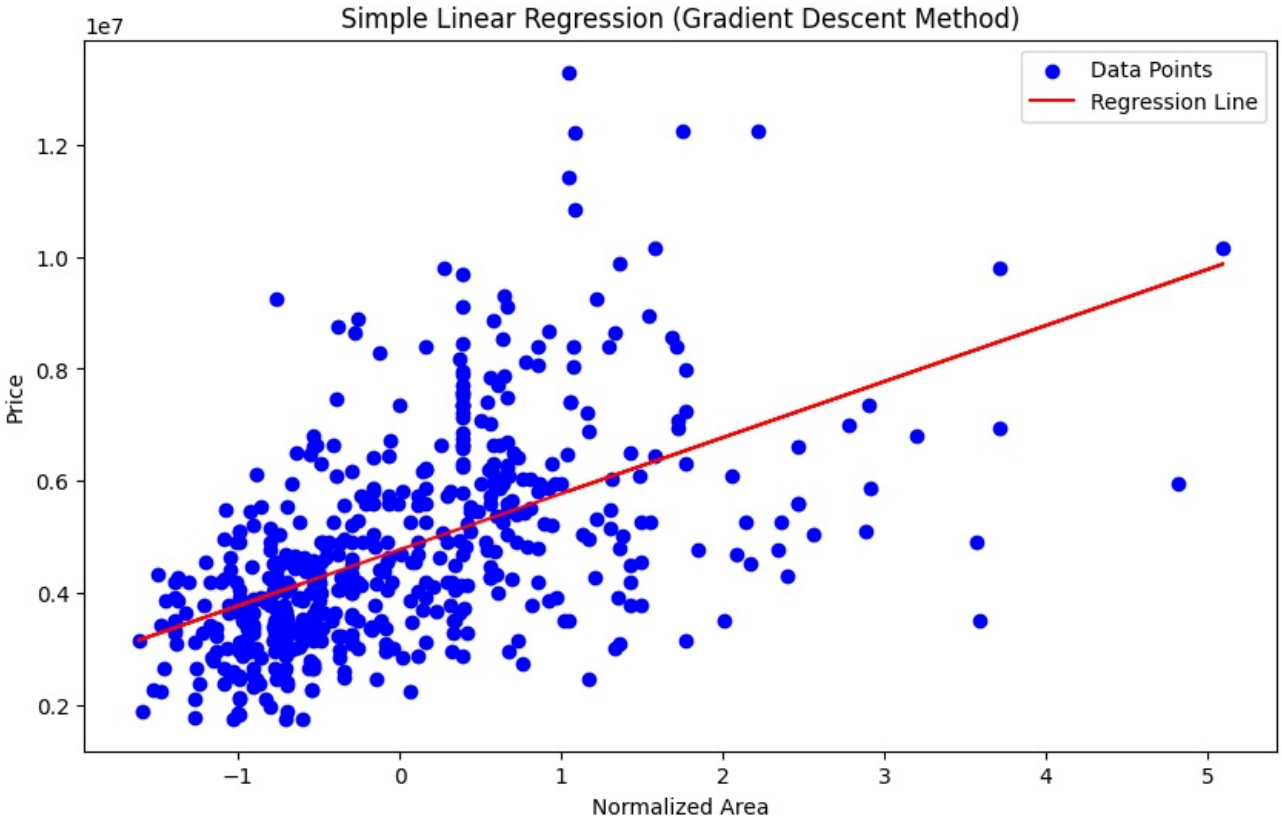
Machine learning Method

In [9]:

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

# Load the dataset
df = pd.read_csv(r'C:\Users\hp\OneDrive\Desktop\housing.csv')  # Update the path to your dataset
```

```python
# Select relevant columns (Area as the independent variable and Price as the dependent variable)
X = df['area'].values
y = df['price'].values

# Normalize the features for better performance
X = (X - X.mean()) / X.std()

# Add a column of ones to include the intercept (bias) in the model
X_b = np.c_[np.ones((len(X), 1)), X]

# Gradient Descent parameters
learning_rate = 0.01
n_iterations = 1000
m = len(X)

# Initialize theta (weights)
theta = np.random.randn(2)

# Gradient Descent Loop
print(f"{'Iteration':<10}{'Theta0':<15}{'Theta1':<15}{'Cost Function'}")
print("-" * 45)

for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - learning_rate * gradients
    cost = np.sum((X_b.dot(theta) - y) ** 2) / (2 * m)

    if iteration % 100 == 0:  # Print every 100 iterations
        print(f"{iteration:<10}{theta[0]:<15.4f}{theta[1]:<15.4f}{cost:<20.4f}")

print("\nOptimal parameters (theta):", theta)

# Make predictions
y_pred = X_b.dot(theta)

# Plotting the data points and the regression line
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.xlabel('Normalized Area')
plt.ylabel('Price')
plt.title('Simple Linear Regression (Gradient Descent Method)')
plt.legend()
plt.show()
```

```
Iteration Theta0         Theta1         Cost Function
---------------------------------------------
0         95334.3807     20031.7703     12637163710153.3242
100       4147210.9350   871451.2915    1444805478982.1462
200       4684569.0042   984366.1703    1247954880020.1990
300       4755833.1927   999340.8913    1244492682196.3521
400       4765284.2177   1001326.8322   1244431789245.8879
500       4766537.6085   1001590.2068   1244430718263.9263
600       4766703.8326   1001625.1354   1244430699427.5530
700       4766725.8772   1001629.7676   1244430699096.2595
800       4766728.8007   1001630.3819   1244430699090.4329
900       4766729.1884   1001630.4634   1244430699090.3306

Optimal parameters (theta): [4766729.23968417 1001630.47418752]
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Experiment No. 4

Aim :- Implement Logistic Regression using machine learning method without using sklear

Theory :- Logistic Regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a prob the dependent variable is bounded between 0 & 1.

1] Binomial - In this there can be 3 or more possible unordered type of the dependent variables such as 0 or 1.

2] Multinomial : In this there can be 3 or more possible unordered type of the dependent variables such as cat, dogs, sheeps, etc.

3] Ordinal: In ordinal logistic regression there can be 3 or more possible ordered types of dependent variables such as low, medium or higher assumption

1] Independent Observations : Each observations is depend of the others meaning there is no correlation b/w any input variables.

2] Binary dependent variable It takes the assumption that the dependent variables must be binary or dichotomous

3) Large sample size :- the sample size is large.

4) Linearity relationship b/w independent variables & log odds :

Let the independent input mat

$$x = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$$

& the dependent variable having only binary values

$y = 0$ if class 1

$= 1$ if class 2

Probability $P(y=1) = \sigma(z)$

$P(y=0) = 1 - \sigma(z)$

$\therefore$ Final logistic regression

$$P(x, b, w) = \frac{1}{1+e^{wx+b}}$$

A

5/9/24

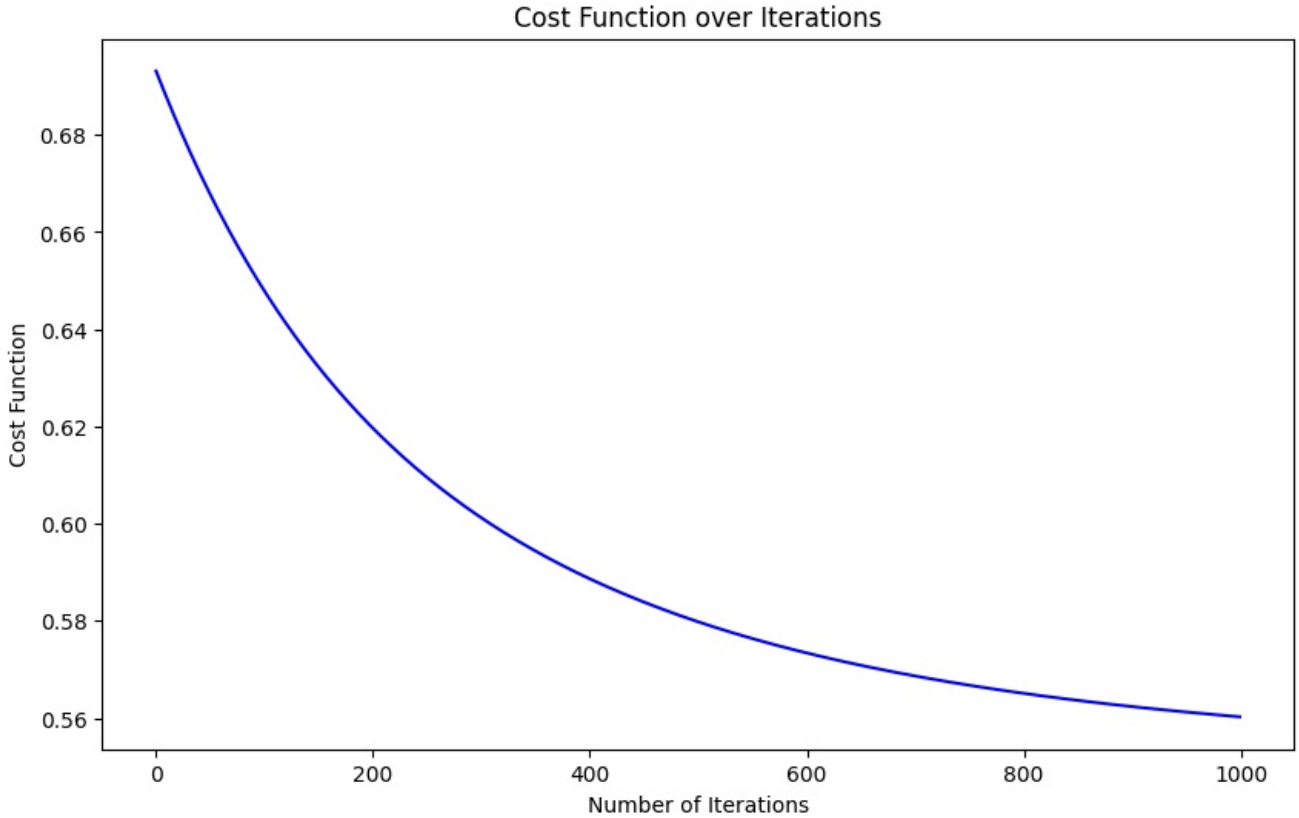
Experiment 4

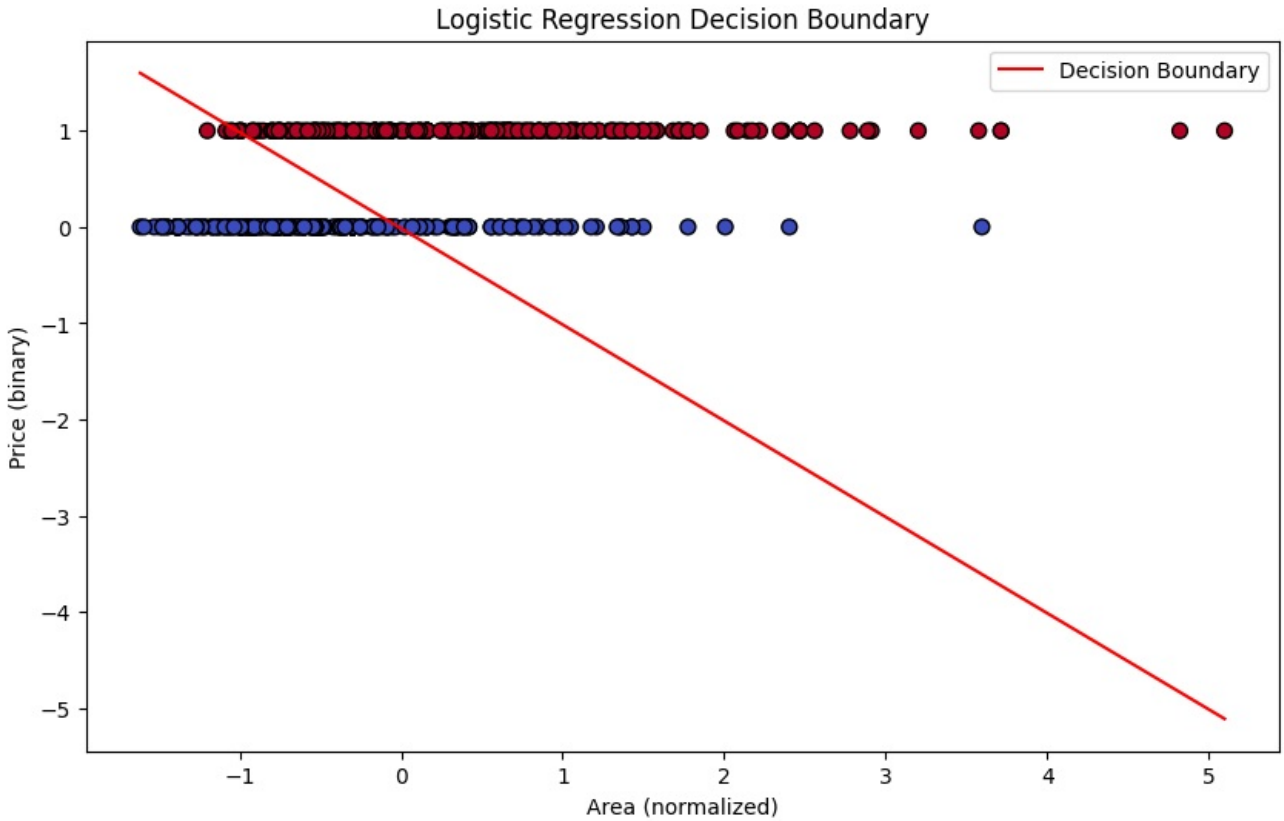
In [3]:

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

# Load the dataset
df = pd.read_csv(r'C:\Users\hp\OneDrive\Desktop\housing.csv')  # Update with your file path

# Feature and target selection
X = df[['area']].values  # Using 'area' as the feature
y = (df['price'] > df['price'].median()).astype(int).values  # Binary target based on median price

# Normalize feature
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Add a column of ones to include the intercept (bias) in the model
X_b = np.c_[np.ones((X.shape[0], 1)), X]

# Sigmoid function with clipping
def sigmoid(z):
    return 1 / (1 + np.exp(-np.clip(z, -250, 250)))  # Clipping to avoid overflow

# Cost function with clipping
def cost_function(y, y_pred):
    y_pred = np.clip(y_pred, 1e-10, 1 - 1e-10)  # Clipping to avoid log(0)
    m = len(y)
    return (-1/m) * np.sum(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred))

# Gradient Descent
def gradient_descent(X, y, learning_rate, num_iterations):
    m, n = X.shape
    theta = np.zeros(n)
    costs = []

    print(f"{'Iteration':<10}{'Theta0':<15}{'Theta1':<15}{'Cost Function'}")
    print("-" * 45)

    for i in range(num_iterations):
        predictions = sigmoid(X.dot(theta))
        errors = predictions - y
        gradients = (1/m) * X.T.dot(errors)
        theta -= learning_rate * gradients
        cost = cost_function(y, predictions)
        costs.append(cost)

        if i % 100 == 0:  # Print every 100 iterations
            print(f"{i:<10}{theta[0]:<15.4f}{theta[1]:<15.4f}{cost:<20.4f}")

    return theta, costs

# Parameters
learning_rate = 0.01
num_iterations = 1000

# Run Gradient Descent
theta, costs = gradient_descent(X_b, y, learning_rate, num_iterations)

print(f"\nOptimal parameters (theta): {theta}")

# Plotting cost function over iterations
plt.figure(figsize=(10, 6))
plt.plot(range(num_iterations), costs, color='blue')
plt.xlabel('Number of Iterations')
plt.ylabel('Cost Function')
plt.title('Cost Function over Iterations')
plt.show()

# Plotting decision boundary
plt.figure(figsize=(10, 6))
plt.scatter(X, y, c=y, cmap='coolwarm', edgecolors='k', s=50)
x_values = np.linspace(X.min(), X.max(), 100)
y_values = (-1 / theta[1]) * (theta[0] + theta[1] * x_values)
plt.plot(x_values, y_values, label='Decision Boundary', color='red')
plt.xlabel('Area (normalized)')
plt.ylabel('Price (binary)')
plt.title('Logistic Regression Decision Boundary')
```

```
plt.legend()
plt.show()
```

```
Iteration Theta0         Theta1         Cost Function
----------------------------------------------
0         -0.0000        0.0024         0.6931
100       -0.0040        0.2142         0.6479
200       -0.0066        0.3816         0.6196
300       -0.0075        0.5170         0.6012
400       -0.0068        0.6287         0.5886
500       -0.0048        0.7225         0.5798
600       -0.0017        0.8023         0.5734
700       0.0020         0.8709         0.5686
800       0.0063         0.9304         0.5651
900       0.0108         0.9823         0.5623

Optimal parameters (theta): [0.01547946 1.02751984]
```





In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Experiment No. 5

Aim:- Implement Decision Tree classification Algo

Theory: A decision tree is a tree like structure that represents a set of decisions & their possible consequences. Each node in the tree represents a decision. The leaves of the tree represent the final decisions. The leaves of the tree represent the final decisions.

Consider the dataset

Age	Competition	Types	Profit
old	yes	software	down
old	No	software	down
old	No	Hardware	down
Mid	yes	software	down
Mid	yes	Hardware	down
Mid	No	Hardware	up
Mid	No	software	up
New	Yes	software	up
New	No	Hardware	up
New	No	software	up

$$Gini(T) = \left[1 - \left[\left(\frac{5}{10}\right)^2 + \left(\frac{5}{10}\right)^2\right]\right] = 0.5$$

Age :

split $= \frac{3}{10}$ gini old $+ \frac{4}{10}$ gini (mid) $+ \frac{3}{10}$ gini (new)

$$= \frac{3}{10}\left[1 - \left[\left(\frac{0}{3}\right)^2 + \left(\frac{2}{3}\right)^2\right]\right] + \frac{4}{10}\left[1 - \left(\frac{2}{4}\right)^2 + \left(\frac{2}{4}\right)^2\right] + \frac{3}{10}\left[1 - \left(\frac{2}{3}\right)\right]$$

$= \underline{0.2}$

* Competition

split $= \frac{4}{10}$ gini yes $+ \frac{6}{10}$ gini no

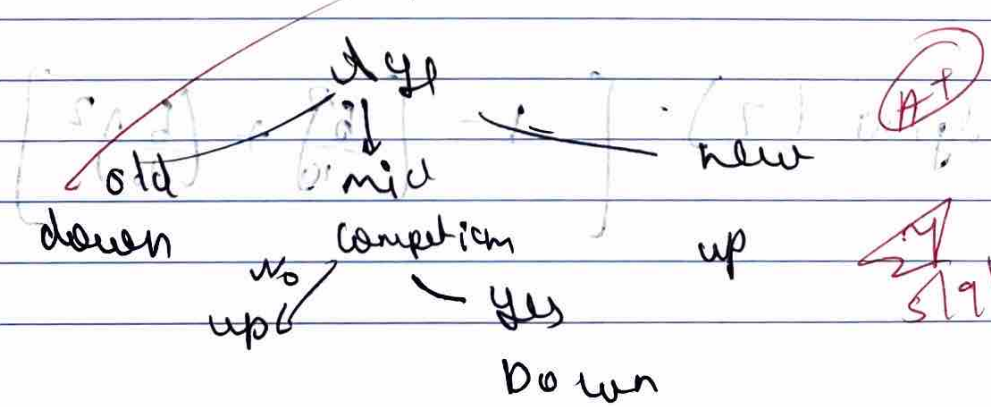
$= 0.42$

- type

split $= \frac{6}{10}$ gini (software) $+ \frac{4}{10}$ gini (hardware)

$= \underline{0.5}$

∴ split values of age is the smallest age = root node



5/9/22

```python
import pandas as pd
import pydotplus


from graphviz import Source
from IPython.display import Image, display

def unique_values(rows, col):
    return set([row[col] for row in rows])

def class_counts(rows):
    counts = {}
    for row in rows:
        label = row[-1]
        if label not in counts:
            counts[label] = 0
        counts[label] += 1
    return counts

def split_dataset(rows, col, value):
    left, right = [], []
    for row in rows:
        if row[col] == value:
            left.append(row)
        else:
            right.append(row)
    return left, right

def gini(rows):
    counts = class_counts(rows)
    impurity = 1
    total = float(len(rows))
    for label in counts:
        prob_of_label = counts[label] / total
        impurity -= prob_of_label ** 2
    return impurity

def information_gain(left, right, current_uncertainty):
    p = float(len(left)) / (len(left) + len(right))
    return current_uncertainty - p * gini(left) - (1 - p) *
gini(right)

def find_best_split(rows):
    best_gain = 0
    best_split = None
    current_uncertainty = gini(rows)
    num_features = len(rows[0]) - 1

    for col in range(num_features):
        values = unique_values(rows, col)
```

```
        for val in values:
            left, right = split_dataset(rows, col, val)
            if len(left) == 0 or len(right) == 0:
                continue
            gain = information_gain(left, right, current_uncertainty)
            if gain > best_gain:
                best_gain, best_split = gain, (col, val)

    return best_gain, best_split

class DecisionNode:
    def __init__(self, feature, value, true_branch, false_branch,
is_leaf=False, prediction=None):
        self.feature = feature
        self.value = value
        self.true_branch = true_branch
        self.false_branch = false_branch
        self.is_leaf = is_leaf
        self.prediction = prediction

def build_tree(rows):
    gain, split = find_best_split(rows)

    if gain == 0:
        return DecisionNode(None, None, None, None, is_leaf=True,
prediction=class_counts(rows))

    col, val = split
    left, right = split_dataset(rows, col, val)

    true_branch = build_tree(left)
    false_branch = build_tree(right)

    return DecisionNode((col, val), None, true_branch, false_branch)

def print_tree(node, indent=""):
    if node.is_leaf:
        print(indent + f"Predict: {node.prediction}")
    else:
        col, val = node.feature
        print(indent + f"{header[col]} == {val}?")
        print(indent + '-> True:')
        print_tree(node.true_branch, indent + "   ")
        print(indent + '-> False:')
        print_tree(node.false_branch, indent + "   ")

def visualize_tree(node, dot=None):
    if dot is None:
        dot = pydotplus.Dot(graph_type="digraph")
```

```
    if node.is_leaf:
        label = f"Predict: {node.prediction}"
        leaf = pydotplus.Node(str(id(node)), label=label, shape="box",
style="filled", fillcolor="lightgray")
        dot.add_node(leaf)
    else:
        col, val = node.feature
        label = f"{header[col]} == {val}?"
        split_node = pydotplus.Node(str(id(node)), label=label)
        dot.add_node(split_node)

        true_branch_dot = visualize_tree(node.true_branch, dot)
        false_branch_dot = visualize_tree(node.false_branch, dot)

        dot.add_edge(pydotplus.Edge(str(id(node)),
str(id(node.true_branch))))
        dot.add_edge(pydotplus.Edge(str(id(node)),
str(id(node.false_branch))))

    return dot

def main():
    # Define the dataset as a dictionary and convert it to a pandas
DataFrame
    df = pd.DataFrame({
        'outlook': ['sunny', 'sunny', 'overcast', 'rain', 'rain',
'rain', 'overcast', 'sunny', 'sunny', 'rain', 'sunny', 'overcast',
'overcast'],
        'temp': ['hot', 'hot', 'hot', 'mild', 'cool', 'cool', 'cool',
'mild', 'cool', 'mild', 'mild', 'mild', 'hot'],
        'humidity': ['high', 'high', 'high', 'high', 'normal',
'normal', 'normal', 'normal', 'high', 'normal', 'normal', 'high',
'normal'],
        'wind': ['weak', 'strong', 'weak', 'weak', 'weak', 'strong',
'strong', 'weak', 'strong', 'strong', 'weak', 'strong', 'weak'],
        'answer': ['no', 'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no',
'yes', 'yes', 'yes', 'yes', 'no']
    })

    # Convert DataFrame to list of lists
    data = df.values.tolist()

    # Define header globally
    global header
    header = df.columns.tolist()[:-1]  # Exclude the last column
(target) from header

    # Build the decision tree
    decision_tree = build_tree(data)
```

```python
    # Print the decision tree in text format
    print("Decision Tree (Text Format):")
    print_tree(decision_tree)

    # Visualize and display the decision tree
    dot_data = visualize_tree(decision_tree)
    display(Source(dot_data.to_string()))

if __name__ == "__main__":
    main()
```

```
Decision Tree (Text Format):
temp == hot?
-> True:
  outlook == overcast?
  -> True:
    humidity == high?
    -> True:
      Predict: {'yes': 1}
    -> False:
      Predict: {'no': 1}
  -> False:
    Predict: {'no': 2}
-> False:
  humidity == normal?
  -> True:
    outlook == overcast?
    -> True:
      Predict: {'yes': 1}
    -> False:
      outlook == rain?
      -> True:
        temp == cool?
        -> True:
          wind == strong?
          -> True:
            Predict: {'no': 1}
          -> False:
            Predict: {'yes': 1}
        -> False:
          Predict: {'yes': 1}
      -> False:
        Predict: {'no': 1, 'yes': 1}
  -> False:
    Predict: {'yes': 3}
```

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Experiment 6

Aim: Implement SVM for Classification

Theory: SVM is a powerful machine learning algorithm used for linear or non linear classification regression and even outlier detection tasks. SVM can be used for a variety of task, such as text, hand classification image classification, spam detection, handwriting identification gene extension analysis, face detection tasks. SVMs are adaptive & efficient in a variety efficient in a variety of application because they can manage high-dimensional data & non linear relationships. SVM algorithms are very effective as we try to find the maximum seperating hyperplane between the different classes accessible in the target feature

SVM terminology :-

Hyperplane :- Hyperplane is the decision boundary that is used to seperate the data points of different classes in a feature space.

Support Vectors :- Support vectors are the closest data points to the hyperplane, which makes a critical role in deciding the hyperplane & margin

Margin :- Margin is the distance b/w the support vectors & hyperplane. Hard margin : the hard margin hyperplane is a hyperplane that properly seperates the data points of

of different categories without any mis calculation.
Soft margin: When the data is not perfectly separable or contains outliers, SVM permits a soft margin technique. Each data point has a slack variable introduced by the soft margin SVM formulation, which softens the strict margin requirement & permits certain misclassification.

Conclusion:

We successfully implemented SVM for classification

A

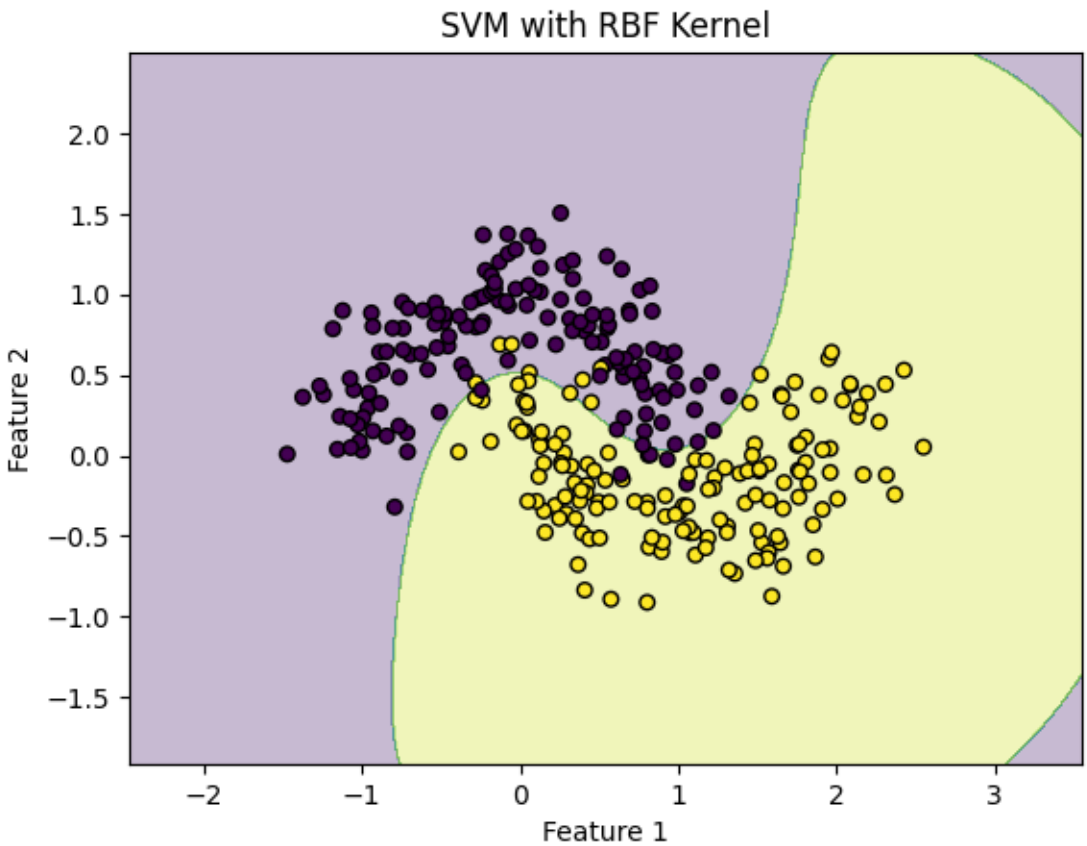
14/10/24

exp6-support-vector-machines

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

# Generate a non-linearly separable dataset (moons dataset)
X, y = make_moons(n_samples=300, noise=0.2, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

# Initialize the SVM classifier with an RBF kernel
svm_classifier = SVC(kernel='rbf', C=1, gamma='scale')

# Train the SVM classifier
svm_classifier.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = svm_classifier.predict(X_test)

# Evaluate the classifier
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Plot the decision boundary
def plot_decision_boundary(clf, X, y):
    # Create a grid to plot the decision boundary
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
```

```python
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    # Predict the classification for each grid point
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot the decision boundary
    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=30, edgecolor='k')
    plt.title("SVM with RBF Kernel")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()

# Visualize the decision boundary
plot_decision_boundary(svm_classifier, X, y)
```

```
Confusion Matrix:
[[37  1]
 [ 2 50]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96        38
           1       0.98      0.96      0.97        52

    accuracy                           0.97        90
   macro avg       0.96      0.97      0.97        90
weighted avg       0.97      0.97      0.97        90
```


SVM with RBF Kernel
Feature 2
Feature 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2
3

Experiment 7

Aim :- Implement ensemble methods to combine different model.

Theory :

Ensemble methods are powerful techniques in ML that combine the predictions of multiple individual models to improve overall prediction performance.

Advantages :

(1) Reduces Overfitting
(2) Increases Model Robustness
(3) Enhance prediction accuracy

(1) Voting ensembles :

Hard voting :- Multiple base models make predictions & final decision is the one with majority vote

Soft voting : Similar to hard voting, but instead of selecting the class with most votes, it calculates weighted avg of class probabilities predicted by the base models

(2) Bagging :- Bagging aims to reduce the variance of an individual base model by training multiple base models on random subsets of the training data with replacement. The final prediction is often made by averaging or majority voting

(3) Boosting :-

they focus on improving the accuracy of weak base models sequentially. Each base model is trained to correct the mistakes made by the previous model

(4) Stacking :-
It combines multiple base models by training a meta model on their predictions. the base models outputs serve as features for the meta model
It is known for its high predictive performance

(5) Ensemble pruning :-

these techniques aim to reduce the complexity of an ensemble by selecting a subset of base models that contribute the most to the ensembles performance. Reduces computational overhead

Other methods : Stacking with cross validation
weighted average ensembles
adaptive ensembles

Conclusion :
Ensemble method implemented

14/10/24

EXPERIMENT 7

BAGGING

In [2]:
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load data
data = load_iris()
X = data.data
y = data.target

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

# Bagging implementation
class Bagging:
    def __init__(self, base_model, n_models=10):
        self.base_model = base_model
        self.n_models = n_models
        self.models = []

    def fit(self, X, y):
        for i in range(self.n_models):
            # Bootstrap sampling
            indices = np.random.choice(len(X), len(X), replace=True)
            X_sample, y_sample = X[indices], y[indices]
            model = self.base_model()
            model.fit(X_sample, y_sample)
            self.models.append(model)
            print(f"Model {i+1} trained with accuracy: {accuracy_score(y_sample, model.predict(X_sample))}")

    def predict(self, X):
        # Aggregate predictions from each model
        predictions = np.zeros((X.shape[0], len(self.models)))
        for i, model in enumerate(self.models):
            predictions[:, i] = model.predict(X)
        # Majority vote
        return np.round(np.mean(predictions, axis=1))

# Initialize and train the bagging model
bagging_model = Bagging(base_model=DecisionTreeClassifier, n_models=10)
bagging_model.fit(X_train, y_train)

# Make predictions and evaluate
y_pred = bagging_model.predict(X_test)
print(f"Bagging Model Accuracy: {accuracy_score(y_test, y_pred)}")
```

```
Model 1 trained with accuracy: 1.0
Model 2 trained with accuracy: 1.0
Model 3 trained with accuracy: 1.0
Model 4 trained with accuracy: 1.0
Model 5 trained with accuracy: 1.0
Model 6 trained with accuracy: 1.0
Model 7 trained with accuracy: 1.0
Model 8 trained with accuracy: 1.0
Model 9 trained with accuracy: 1.0
Model 10 trained with accuracy: 1.0
Bagging Model Accuracy: 1.0
```

BOOSTING

In [5]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.utils import shuffle

class AdaBoost:
    def __init__(self, base_model, n_models=50):
        self.base_model = base_model
        self.n_models = n_models
        self.models = []
        self.alphas = []
```

```python
    def fit(self, X, y):
        n_samples = X.shape[0]
        weights = np.ones(n_samples) / n_samples
        for i in range(self.n_models):
            model = self.base_model()
            model.fit(X, y, sample_weight=weights)
            y_pred = model.predict(X)
            # Calculate error
            error = np.sum(weights * (y_pred != y)) / np.sum(weights)

            # Handle zero error
            if error == 0:
                alpha = 1.0
            else:
                alpha = 0.5 * np.log((1 - error) / error)

            # Update weights
            weights *= np.exp(alpha * (y_pred != y))
            weights /= np.sum(weights)

            # Check for NaN in weights
            if np.any(np.isnan(weights)):
                raise ValueError("Weights contain NaN values")

            self.models.append(model)
            self.alphas.append(alpha)
            print(f"Model {i+1}: Error = {error}, Alpha = {alpha}")

    def predict(self, X):
        predictions = np.zeros(X.shape[0])
        for model, alpha in zip(self.models, self.alphas):
            predictions += alpha * model.predict(X)
        return np.sign(predictions)

# Initialize and train the AdaBoost model
adaboost_model = AdaBoost(base_model=DecisionTreeClassifier, n_models=50)
adaboost_model.fit(X_train, y_train)

# Make predictions and evaluate
y_pred = adaboost_model.predict(X_test)
print(f"AdaBoost Model Accuracy: {accuracy_score(y_test, y_pred)}")
```

```
Model 1: Error = 0.0, Alpha = 1.0
Model 2: Error = 0.0, Alpha = 1.0
Model 3: Error = 0.0, Alpha = 1.0
Model 4: Error = 0.0, Alpha = 1.0
Model 5: Error = 0.0, Alpha = 1.0
Model 6: Error = 0.0, Alpha = 1.0
Model 7: Error = 0.0, Alpha = 1.0
Model 8: Error = 0.0, Alpha = 1.0
Model 9: Error = 0.0, Alpha = 1.0
Model 10: Error = 0.0, Alpha = 1.0
Model 11: Error = 0.0, Alpha = 1.0
Model 12: Error = 0.0, Alpha = 1.0
Model 13: Error = 0.0, Alpha = 1.0
Model 14: Error = 0.0, Alpha = 1.0
Model 15: Error = 0.0, Alpha = 1.0
Model 16: Error = 0.0, Alpha = 1.0
Model 17: Error = 0.0, Alpha = 1.0
Model 18: Error = 0.0, Alpha = 1.0
Model 19: Error = 0.0, Alpha = 1.0
Model 20: Error = 0.0, Alpha = 1.0
Model 21: Error = 0.0, Alpha = 1.0
Model 22: Error = 0.0, Alpha = 1.0
Model 23: Error = 0.0, Alpha = 1.0
Model 24: Error = 0.0, Alpha = 1.0
Model 25: Error = 0.0, Alpha = 1.0
Model 26: Error = 0.0, Alpha = 1.0
Model 27: Error = 0.0, Alpha = 1.0
Model 28: Error = 0.0, Alpha = 1.0
Model 29: Error = 0.0, Alpha = 1.0
Model 30: Error = 0.0, Alpha = 1.0
Model 31: Error = 0.0, Alpha = 1.0
Model 32: Error = 0.0, Alpha = 1.0
Model 33: Error = 0.0, Alpha = 1.0
Model 34: Error = 0.0, Alpha = 1.0
Model 35: Error = 0.0, Alpha = 1.0
Model 36: Error = 0.0, Alpha = 1.0
Model 37: Error = 0.0, Alpha = 1.0
Model 38: Error = 0.0, Alpha = 1.0
Model 39: Error = 0.0, Alpha = 1.0
Model 40: Error = 0.0, Alpha = 1.0
Model 41: Error = 0.0, Alpha = 1.0
Model 42: Error = 0.0, Alpha = 1.0
Model 43: Error = 0.0, Alpha = 1.0
Model 44: Error = 0.0, Alpha = 1.0
Model 45: Error = 0.0, Alpha = 1.0
Model 46: Error = 0.0, Alpha = 1.0
Model 47: Error = 0.0, Alpha = 1.0
Model 48: Error = 0.0, Alpha = 1.0
Model 49: Error = 0.0, Alpha = 1.0
Model 50: Error = 0.0, Alpha = 1.0
AdaBoost Model Accuracy: 0.7111111111111111
```

STACKING

In [4]:

```python
from sklearn.linear_model import LogisticRegression

class Stacking:
    def __init__(self, base_models, meta_model):
        self.base_models = base_models
        self.meta_model = meta_model
        self.meta_X_train = None

    def fit(self, X, y):
        meta_X = np.zeros((X.shape[0], len(self.base_models)))
        for i, model in enumerate(self.base_models):
            model.fit(X, y)
            meta_X[:, i] = model.predict(X)
            print(f"Base Model {i+1} predictions: {meta_X[:, i]}")
        self.meta_model.fit(meta_X, y)

    def predict(self, X):
        meta_X = np.zeros((X.shape[0], len(self.base_models)))
        for i, model in enumerate(self.base_models):
            meta_X[:, i] = model.predict(X)
        return self.meta_model.predict(meta_X)

# Initialize base models and meta-model
base_models = [DecisionTreeClassifier(), DecisionTreeClassifier(max_depth=3)]
meta_model = LogisticRegression()

# Initialize and train the stacking model
```

```python
stacking_model = Stacking(base_models=base_models, meta_model=meta_model)
stacking_model.fit(X_train, y_train)

# Make predictions and evaluate
y_pred = stacking_model.predict(X_test)
print(f"Stacking Model Accuracy: {accuracy_score(y_test, y_pred)}")
```

```
Base Model 1 predictions: [1. 2. 2. 1. 2. 1. 2. 1. 0. 2. 1. 0. 0. 0. 1. 2. 0. 0. 0. 1. 0. 1. 2. 0.
 1. 2. 0. 2. 2. 1. 1. 2. 1. 0. 1. 2. 0. 0. 1. 1. 0. 2. 0. 0. 1. 1. 2. 1.
 2. 2. 1. 0. 0. 2. 2. 0. 0. 0. 1. 2. 0. 2. 2. 0. 1. 1. 2. 1. 2. 0. 2. 1.
 2. 1. 1. 1. 0. 1. 1. 0. 1. 2. 2. 0. 1. 2. 2. 0. 2. 0. 1. 2. 2. 1. 2. 1.
 1. 2. 2. 0. 1. 2. 0. 1. 2.]
Base Model 2 predictions: [1. 1. 2. 1. 2. 1. 2. 1. 0. 2. 1. 0. 0. 0. 1. 2. 0. 0. 0. 1. 0. 1. 2. 0.
 1. 2. 0. 2. 2. 1. 1. 2. 1. 0. 1. 2. 0. 0. 1. 1. 0. 2. 0. 0. 2. 1. 2. 1.
 1. 2. 1. 0. 0. 1. 2. 0. 0. 0. 1. 2. 0. 2. 2. 0. 1. 1. 2. 1. 2. 0. 2. 1.
 2. 1. 1. 1. 0. 1. 1. 0. 1. 2. 2. 0. 1. 2. 1. 0. 2. 0. 1. 2. 2. 1. 2. 1.
 1. 2. 2. 0. 1. 2. 0. 1. 2.]
Stacking Model Accuracy: 1.0
```

In [ ]:

```python

```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment 8

Aim :- Implement Principal Component Analysis technique

Theory :-

PCA is a dimensionality reduction technique commonly used to transform high-dimensional data into a lower dimensional representation while preserving the most important information

PCA : this by finding orthogonal axes along which the data exhibits the maximum variance

These principal components are linear combination of the original features, allowing you to reduce the dimensionality of the data

The steps & concepts of PCA are :-

(1) Centering the data : By subtracting the means of each feature from the corresponding data point

(2) Covariance matrix :

(3) Eigen value decomposition

(4) Eigen value & Eigen vector

(5) Selecting principal components

(6) Projection :

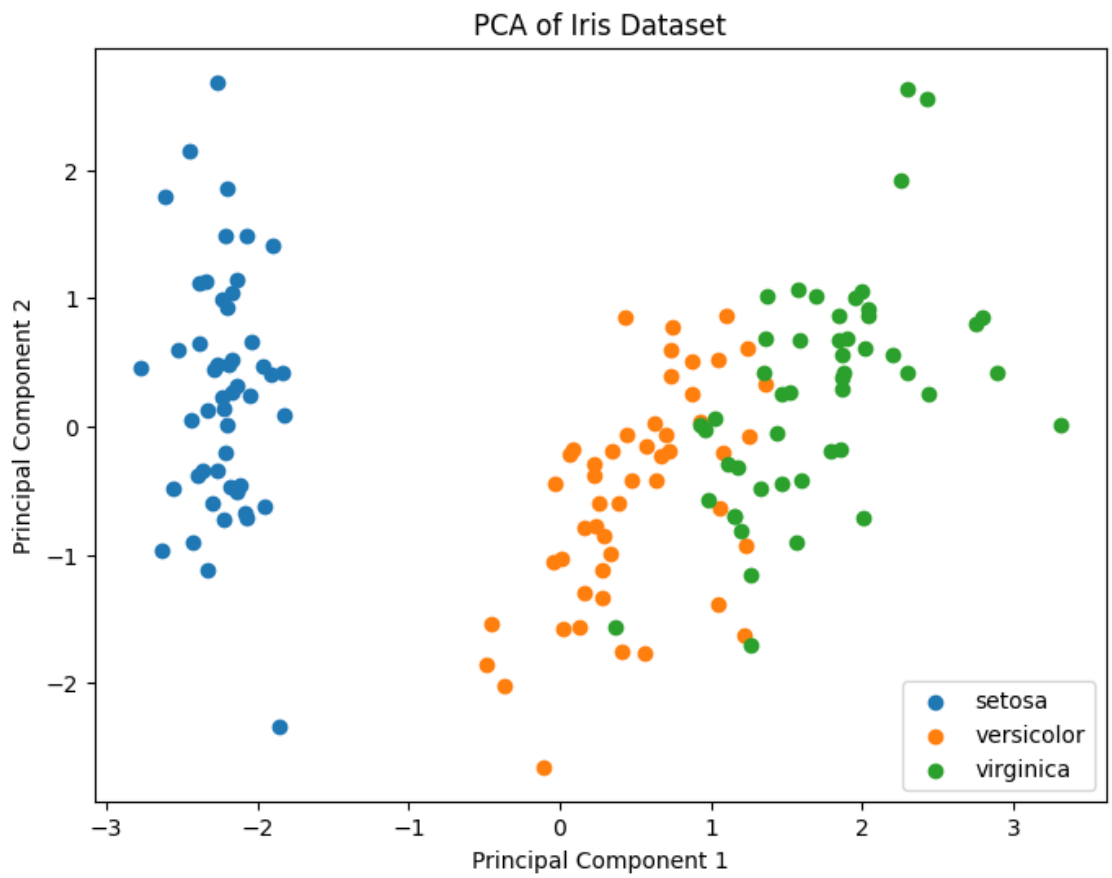
Conclusion: Principal component Analysis implemented

A

Solution

exp8-principal-component-analysis

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Initialize PCA to reduce the data to 2 principal components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Plot the data in the reduced 2D PCA space
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[y == 0, 0], X_pca[y == 0, 1], label=data.target_names[0])
plt.scatter(X_pca[y == 1, 0], X_pca[y == 1, 1], label=data.target_names[1])
plt.scatter(X_pca[y == 2, 0], X_pca[y == 2, 1], label=data.target_names[2])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.legend()
plt.show()

# Print the explained variance ratio
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
```




```
Explained Variance Ratio: [0.72962445 0.22850762]
```

Machine Learning

Mini Project

Title: Plant Disease Detection

BE C11 - Group 1

Group Members:

Karan Adwani – 2103009
Priyanka Bajaj - 2103012
Hardik Bhagat - 2103017
Akshay Jagiasi - 2103062

14/10/22

Problem Statement

Project Title: Plant Disease Detection Using Machine Learning and ResNet

This project focuses on building a machine learning model for detecting diseases in plants using image analysis. The model allows users to upload an image of a plant's affected part, and it identifies the disease accurately. The solution is built using a deep learning model, specifically the ResNet (Residual Networks) architecture, which is highly effective for image classification tasks.

The project automates the manual process of disease detection in agriculture, providing a scalable solution that reduces human error, enhances efficiency, and offers early intervention in crop disease management. By leveraging machine learning, farmers and researchers can benefit from timely and accurate diagnosis of plant diseases.

Motivation

- This project aims to assist the agricultural community by:
- Providing a tool for fast and accurate detection of plant diseases.
- Reducing reliance on expert inspections and minimizing delays in disease identification.
- Enhancing crop health management through early detection and intervention.

Problem description

Manual Detection of Plant Diseases

Traditional plant disease detection involves visual inspection by experts, which is prone to delays, human error, and subjectivity. In large-scale farming, this manual process becomes infeasible. Moreover, many farmers lack access to skilled professionals for timely disease identification, leading to the unchecked spread of diseases and crop loss.

Challenges in Image-Based Detection

One of the main challenges is the accurate classification of diseases based on visual symptoms. Different diseases may cause similar symptoms like spots or discoloration, and the system needs to distinguish between them effectively. The model must also work with images of varying quality, taken under different environmental conditions, and still provide reliable results.

Effective Disease Classification Using ResNet

ResNet, a state-of-the-art deep learning architecture, addresses these challenges by using residual learning techniques to improve the model's ability to extract features from images. By employing skip connections in its layers, ResNet can train very deep networks without performance degradation, making it ideal for complex image classification tasks.

System Requirements

Functional Requirements

The chatbot must be able to:

- Accept natural language questions from users.
- Process the input using NLP techniques to understand the intent of the query.
- Retrieve information from a pre-trained dataset that includes data from the TSEC website.
- Provide accurate and relevant responses based on the question.
- Handle a wide range of topics, such as departments, teachers, subjects, fees, and general college information.

Non-Functional Requirements

- The system should be:

- **User-friendly**: The interface should be simple, enabling users to easily upload images and access results.
- **Fast**: The system should analyze images and return diagnoses in a timely manner.
- **Scalable**: The model should handle multiple requests simultaneously and scale to accommodate various types of plant diseases.
- **Accurate**: High classification accuracy is critical to ensure reliable diagnoses.
- **Maintainable**: The system should be flexible enough to allow for the integration of new data or diseases as required.

Design and Architecture

System Overview

1. The plant disease detection system consists of several components:
2. **User Interface**: A web-based or mobile application interface that allows users to upload plant images. The user interface is designed to be accessible and easy to navigate for farmers and agricultural professionals.
3. **Image Preprocessing Module**: Once the image is uploaded, the system preprocesses it by resizing, normalizing, and enhancing its features to ensure consistent input quality for the machine learning model.
4. **ResNet Model for Disease Classification**: The core of the system is a ResNet-based model trained on a dataset of plant disease images. The ResNet architecture, known for its deep learning capabilities, is used to analyze the input image and extract meaningful features for classification.
5. **Disease Classification Module**: The system classifies the detected disease into predefined categories based on the features extracted by the ResNet model.
6. **Diagnosis and Feedback**: After classification, the system provides the user with the diagnosis of the disease, along with relevant information and possible treatment recommendations.

System Flow

1. **Image Upload**: Users upload an image showing the affected plant part.
2. **Image Preprocessing**: The image undergoes preprocessing, including resizing and normalization, to prepare it for model analysis.
3. **Model Analysis**: The image is fed into the ResNet model, which analyzes the visual features and detects the disease.
4. **Disease Classification**: The system classifies the disease based on the patterns and features recognized by ResNet.
5. **Diagnosis and Feedback**: The system provides a detailed diagnosis and offers suggestions for disease management.

Machine Learning Techniques

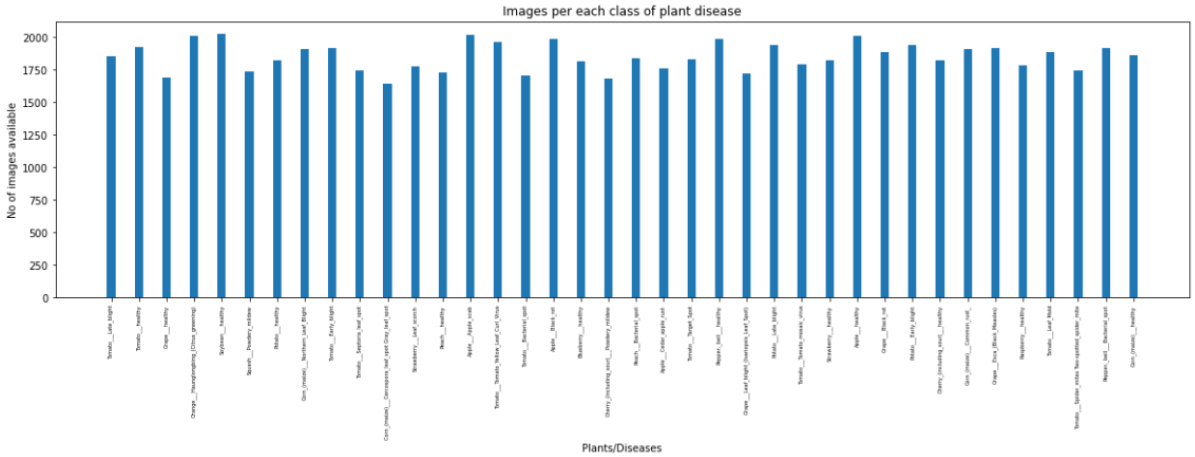
ResNet (Residual Networks)

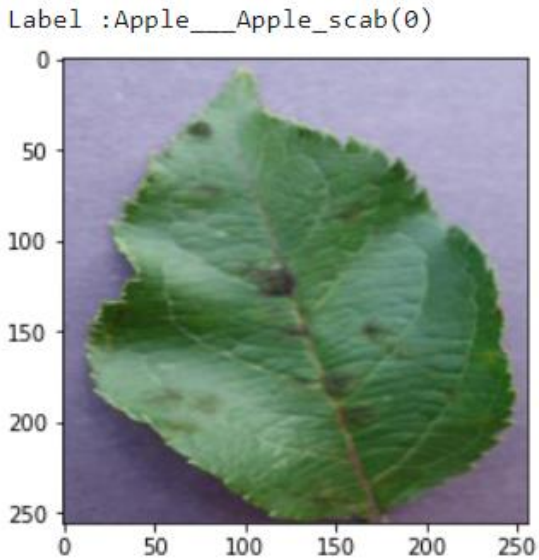
ResNet is a powerful deep learning architecture that overcomes the vanishing gradient problem common in deep networks. By introducing residual connections, ResNet enables the training of very deep networks (with layers exceeding 100) while maintaining high performance. This makes it ideal for image classification tasks, especially when subtle visual patterns, like those found in plant diseases, need to be detected.

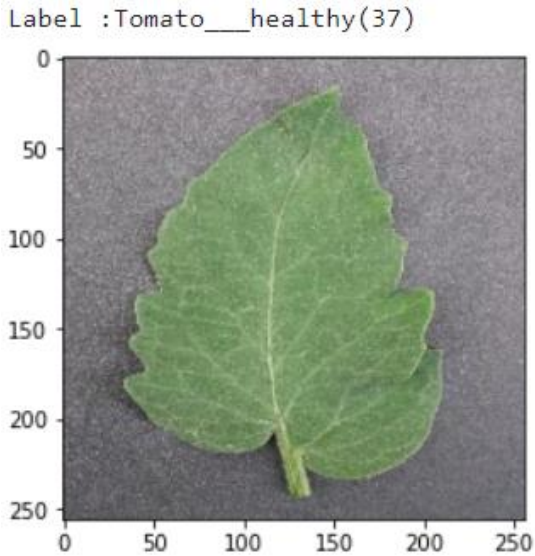
ResNet's architecture consists of:

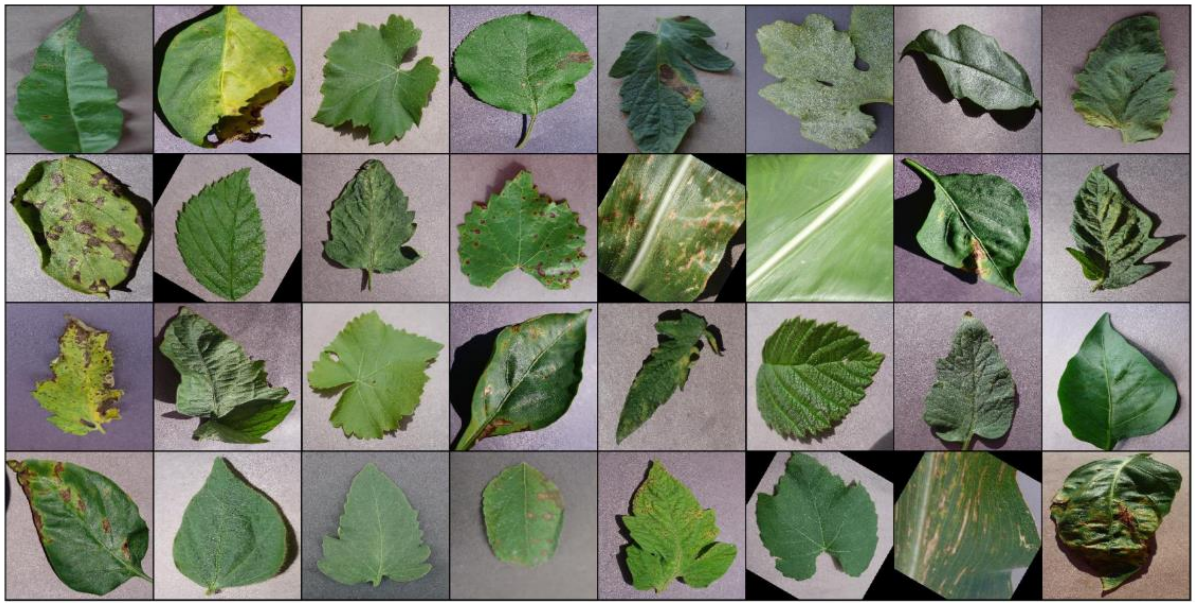
- **Residual Blocks**: These blocks use skip connections, allowing the model to learn identity mappings and improve performance with deep layers.
- **Convolutional Layers**: ResNet applies convolution operations to the input image, extracting features such as edges, textures, and patterns that are important for disease detection.
- **Pooling Layers**: These layers reduce the spatial dimensions of the feature maps, making the model more computationally efficient while retaining important information.
- **Fully Connected Layers**: The fully connected layers take the features learned by the convolutional layers and use them to make predictions about the plant disease.

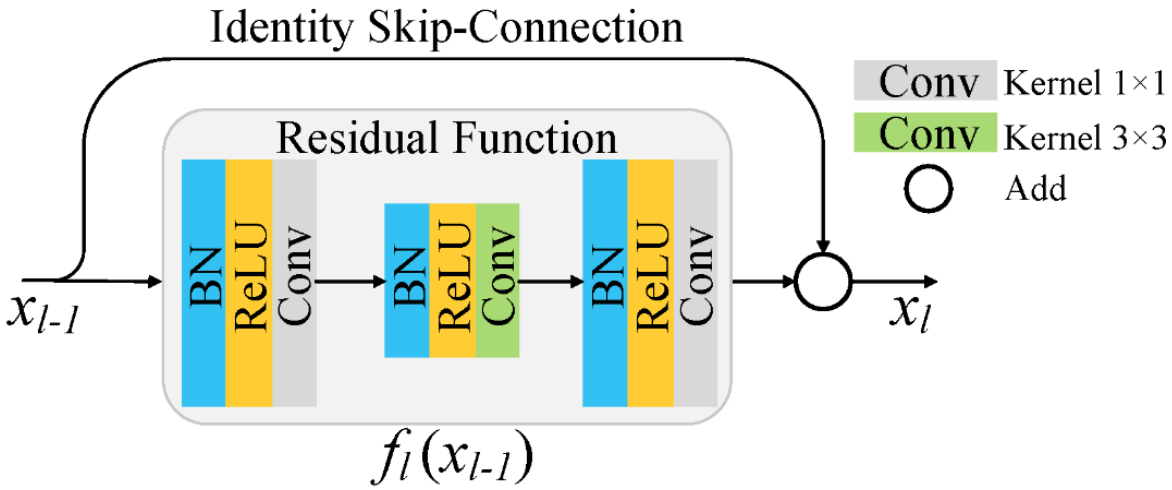
Training and Dataset




Label :Apple___Apple_scab(0)
0
50
100
150
200
250
0
50
100
150
200
250


Label :Tomato___healthy(37)
0
50
100
150
200
250
0
50
100
150
200
250



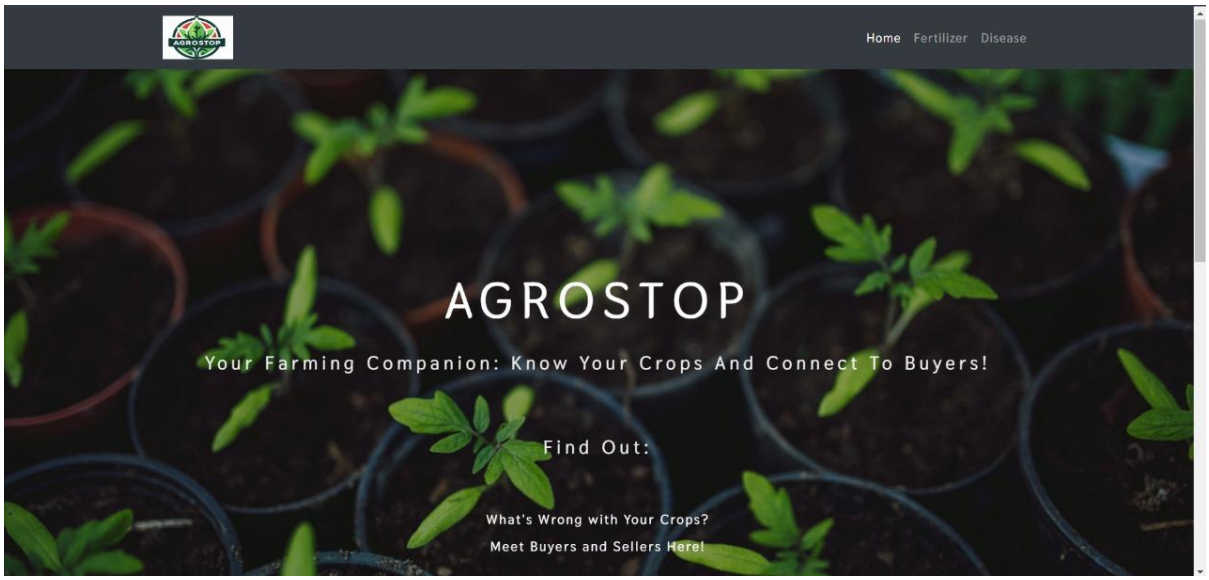

Identity Skip-Connection
Residual Function
BN
ReLU
Conv
BN
ReLU
Conv
BN
ReLU
Conv
x_{l-1}
x_l
$f_l(x_{l-1})$
Conv Kernel 1×1
Conv Kernel 3×3
Add

Conclusion

The plant disease detection system developed in this project uses a deep learning approach based on ResNet to detect plant diseases from images with high accuracy. By automating the disease detection process, the system offers a scalable solution to assist farmers in diagnosing plant diseases early, helping prevent crop loss and improving agricultural productivity.

The ResNet model provides reliable and accurate classification of diseases, and the system's user-friendly interface makes it accessible to a wide audience. This project demonstrates the potential of machine learning to revolutionize agriculture by enabling fast, accurate, and scalable plant disease detection.

Output



About Us

At AgroStop, we are committed to enhancing agriculture, improving livelihoods, and increasing farmers' profits. We utilize machine learning to guide you through your entire farming journey. Our focus is on helping you make informed decisions by providing crop health Join us in cultivating success for your farming

Agrostop

Find out which disease has been caught by your plant

Please Upload The Image

Choose File plan3.webp

Predict

Crop: Pepper

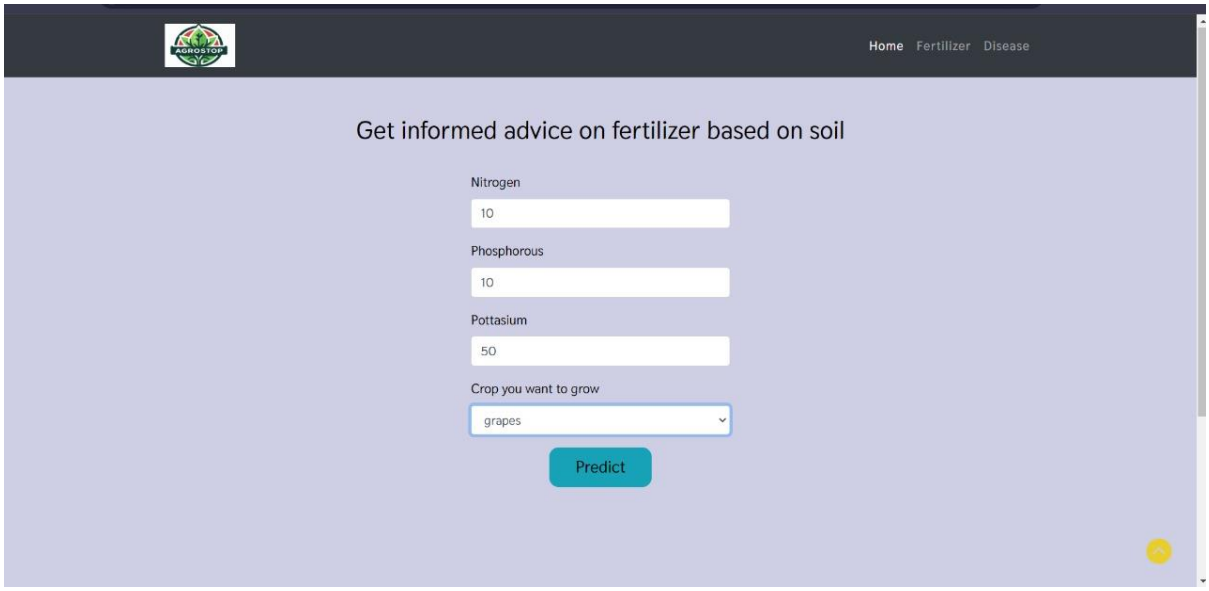
Disease: Bacterial Spot

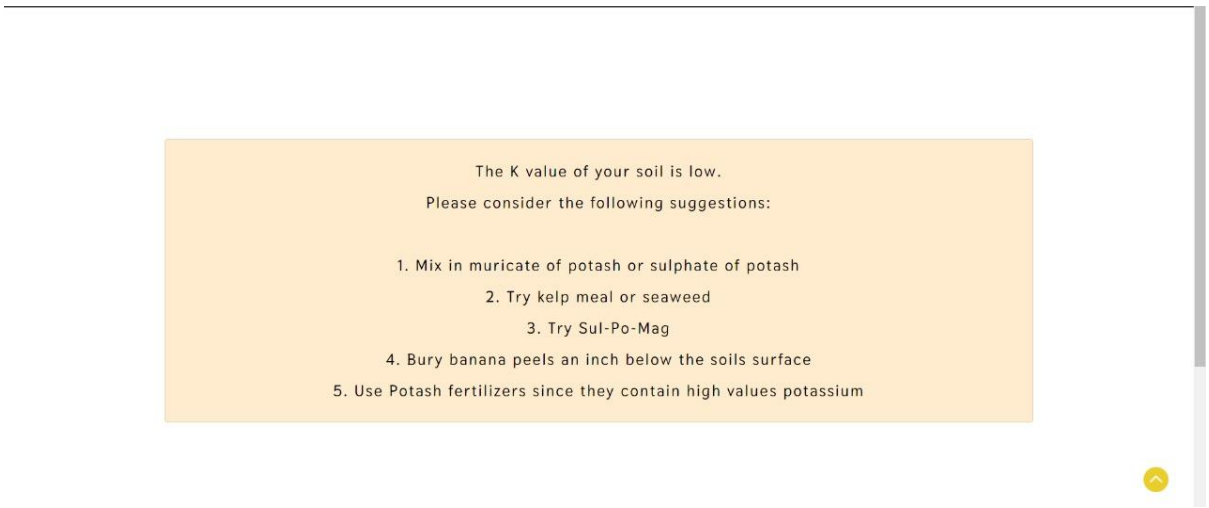
Cause of disease:

1. Bacterial spot is caused by several species of gram-negative bacteria in the genus Xanthomonas.
2. In culture, these bacteria produce yellow, mucoid colonies. A "mass" of bacteria can be observed oozing from a lesion by making a cross-sectional cut through a leaf lesion, placing the tissue in a droplet of water, placing a cover-slip over the sample, and examining it with a microscope (~200X)..

How to prevent/cure the disease

1. The primary management strategy of bacterial spot begins with use of certified pathogen-


Home
Fertilizer
Disease
Get informed advice on fertilizer based on soil
Nitrogen
10
Phosphorous
10
Pottasium
50
Crop you want to grow
grapes
Predict


The K value of your soil is low.
Please consider the following suggestions:
1. Mix in muricate of potash or sulphate of potash
2. Try kelp meal or seaweed
3. Try Sul-Po-Mag
4. Bury banana peels an inch below the soils surface
5. Use Potash fertilizers since they contain high values potassium

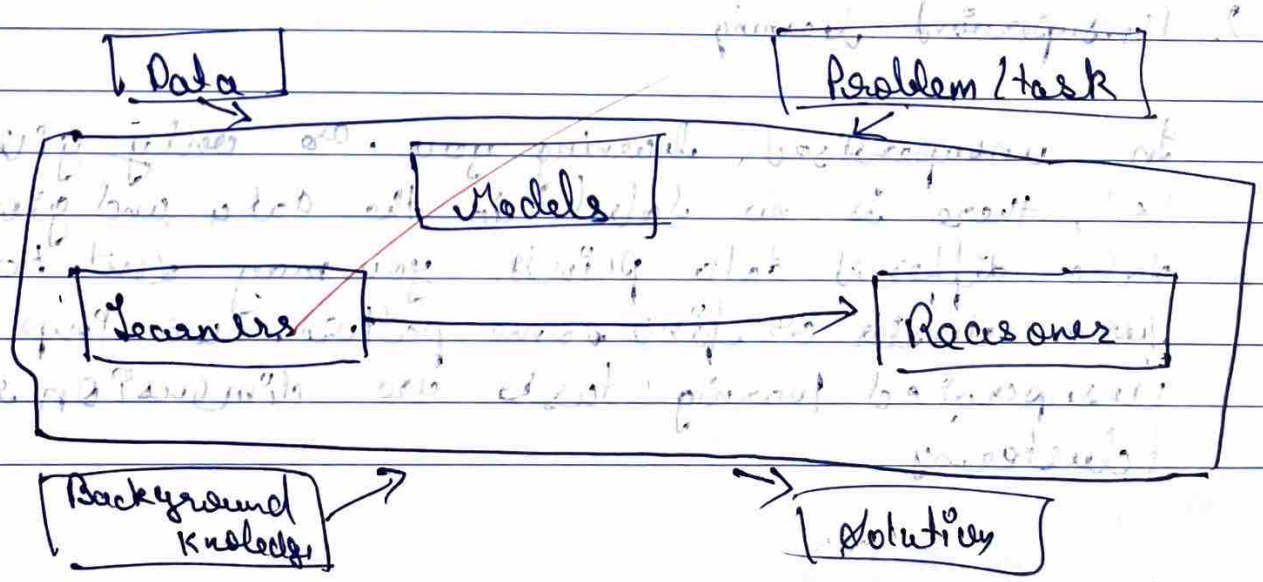
THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Assignment 1

Q1) What is learning ? Explain different types of learning with example.

⇒ Machine learning is a category of Artificial Intelligence. In Machine learning computers have the ability to learn themselves. explicit programming is not required. Machine learning focuses on the study and development of algorithm that can learn themselves, explicit programming is not required. Machine learning focuses on the study and development of algorithm that can learn from data and also make predictions on data

Machine learning mainly focuses on the design and development of computer programs that can teach themselves to grow and change when exposed to new data. Using machine learning we can collect information from a dataset by asking the computer to make some sense from data. M L is turning data into information

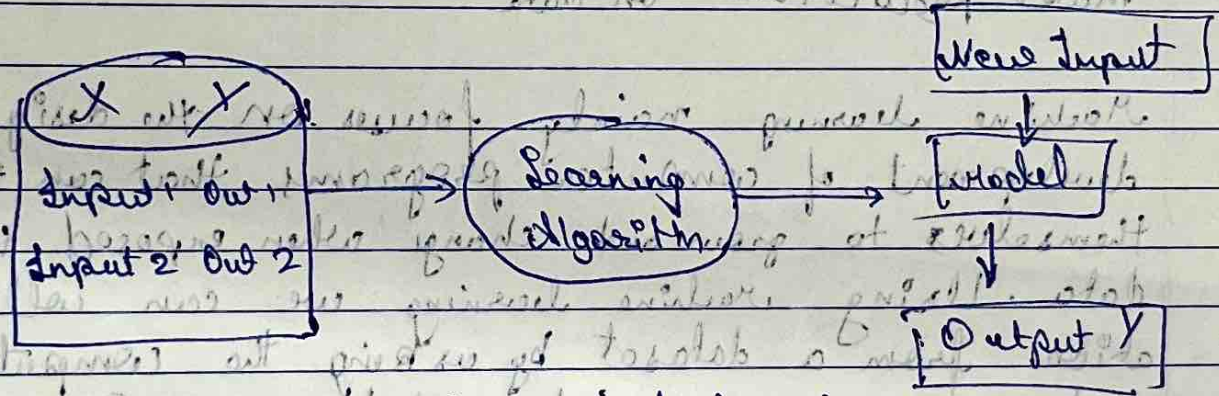


types of machine learning are:

1. Supervised learning

In this type of learning we use data which is comprises of input and corresponding output. In supervised learning training data is labelled with the correct answers eg. 'spam' or 'ham'

two most important types of supervised learning are classification (where the output are discrete labels, as in spam filtering) and regression (where the output are real valued)

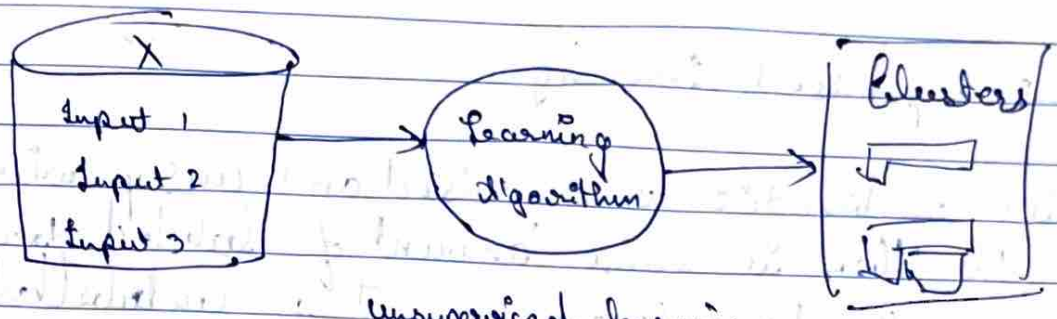


i.e. Supervised learning

2. Unsupervised learning

In unsupervised learning you are only given input 'x', there is no label to the data and given the data different data points, you may want to form cluster or find some pattern. Two imp unsupervised learning tasks are dimension reduction & clustering

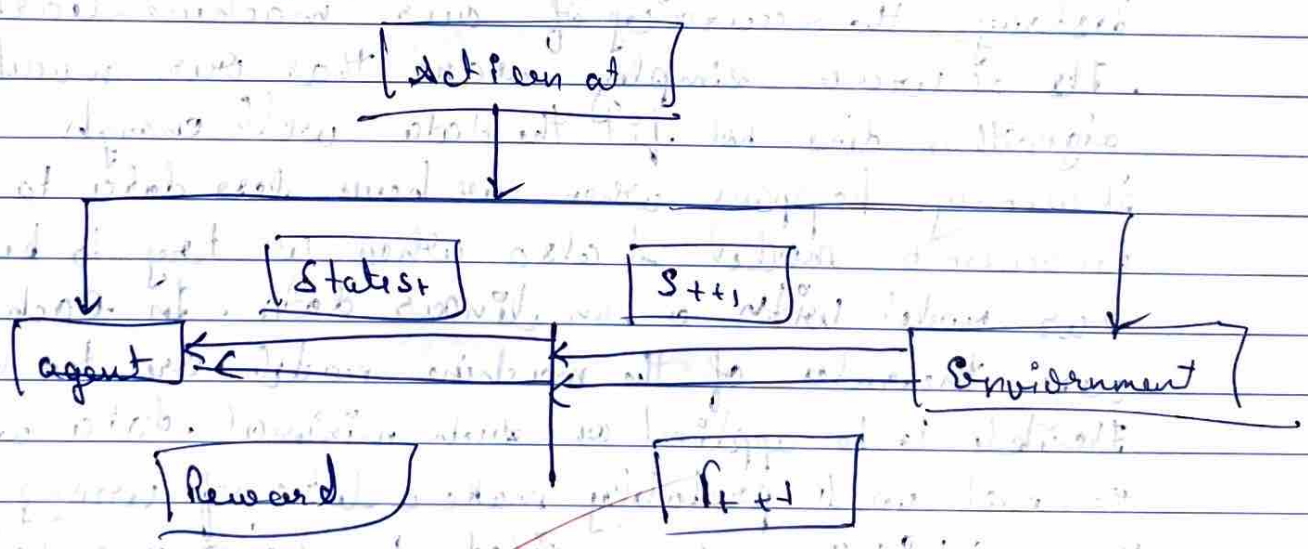
THADOMAL SHAHANI
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE



unsupervised learning

3. Reinforcement Learning

In reinforcement learning you have an agent who is acting in an enviornment and you want to findout what action the agent must take based on the reward or penalty that the agent gets. In this an agent, Example a robot or controller, seeks to learn the optimal actions based on the outcomes of past actions



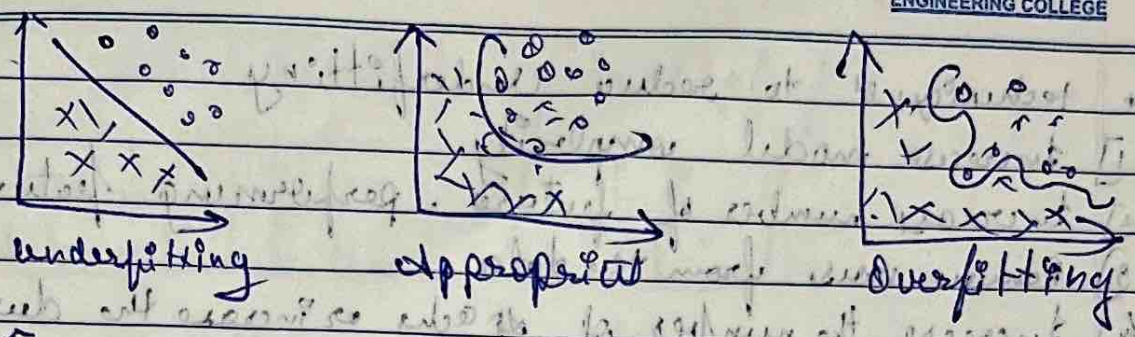
Reinforcement learning

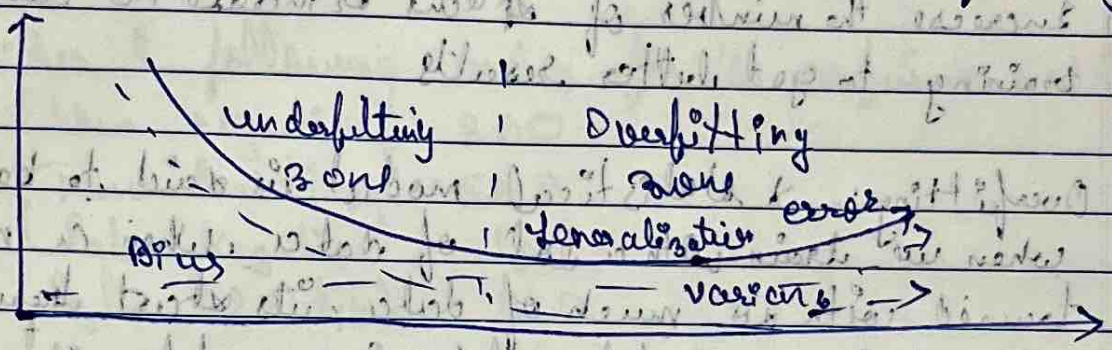
4] Semi Supervised Learning :

It is a combination of supervised and unsupervised learning. In this there is some amount of labeled train data and also you have large amount of unlabelled data and you try to come up with some learning algorithm that converts even when training data is not labelled eg : text classification image classification, where small amount of labelled

Q2] Define Overfitting & Underfitting.

⇒ Underfitting A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data. Underfitting destroys the accuracy of our machine learning model. Its occurence simply means that our model or the algorithm does not fit the data well enough
It usually happens when we have less data to build an accurate model & also when we try to build a linear model with a non linear data. In each such case the rules of the machine model are too easy & flexible to be applied on such minimal data and therefore the model will probably make a lot of wrong predictions. Underfitting can be avoided by using more data & also reducing the features by feature selection. Underfitting can be refined as high bias ~~and low var~~





a) Illustrate process of learning with the gradient descent for a univariate linear regression using a bell shaped error function. Explain how a step size is modulated on every iteration

⇒ Gradient descent is an iterative optimization algorithm that tries to find the optimum value (minimum or maximum) of an objective function. It is one of the most used optimization techniques in machine learning projects for updating the parameters of a model in order to minimize a cost function.

The main aim of gradient descent is to find the best parameter of a model which gives the highest accuracy of training as well as testing datasets. In gradient descent the gradient is a vector that points in the direction of the steepest increase of the function at a special point. Moving in the opposite direction of the gradient allows the algorithm to gradually descend towards lower values of the function and eventually reaching to the minimum of the function

* techniques to reduce Underfitting

1] Increase model complexity.
2] Increase number of features, performing features engineering
3] Remove noise from the data.
4] Increase the number of epochs or increase the duration of training to get better results

Overfitting :- A statistical model is said to be overfitted when we train with a lot of data, when a model gets trained with so much of data, its start learning from the noise and inaccurate data entries in our data set. then the model does not categories the data correctly because of too many details and noise. the causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the data set and therefore they can really build unrealistic models. a solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like maximal depth if we are using decision trees. Overfitting It can be defined as bias and low bias

techniques to reduce overfitting

1. Increase training data
2] Reduce model complexity
3] Early stopping during the training phase
4] Ridge Regularization & Lasso
5] use dropout for neural network to tackle over fitting

THADOMAL SHAHANI
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Consider the following values for the confusion matrix
TN (true negative) = 300
TP (True Positive) = 500
FN (False Negative) = 150
FP (False Positive) = 50

a] Accuracy

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

Accuracy is defined as the ratio of the number of correct predictions and the total number of predictions. It lies between (0, 1). higher accuracy means a better model (TP & TN). higher accuracy means better model (TP & TN) must be high

Accuracy is not a useful matrix in case of an imbalanced datasets, eg 960 patients without cancer & 40 patient with cancer

Accuracy using above values : $\frac{(500+300)}{(500+50+150+300)} = \frac{800}{1000}$ %

- Precision

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

Precision is useful when we want to reduce the number of false positives. Eg. a system that predicts if the email received is spam or not. taking spam as a positive, here, we not want our system to predict non spam emails

Precision using above values

$$= \frac{500}{500+50} = \frac{500}{550} = 90.90\%$$

c] Recall

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

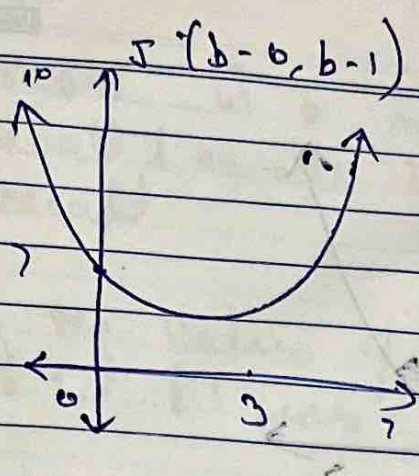
Recall is useful in case of cancer detection, where we want to minimize the number of false negative for any practical use since we dont want to miss a person with cancer as sick.

Recall using above values

$$= \frac{500}{500+150} = \frac{500}{650} = 76.92\%$$

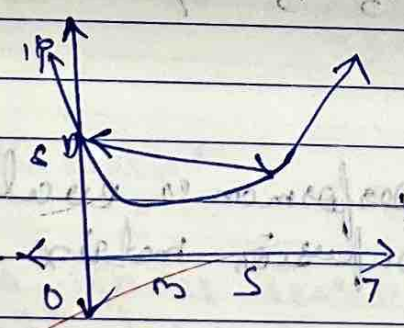
d] F-1 score

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

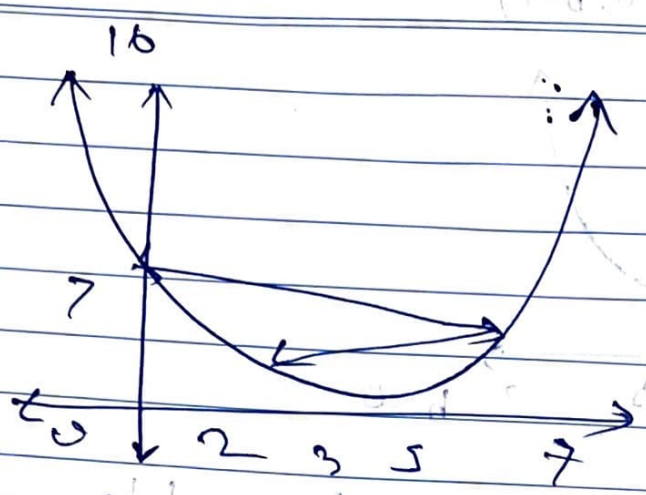


As we can see we have a simple parabola with a minimum at b-0 = 3 the model will use the gradient descent algorithm to gradually tune the values of b_0 & b_1
to start this process, our model will initialize all parameters to 0 this means our initial cost is 7

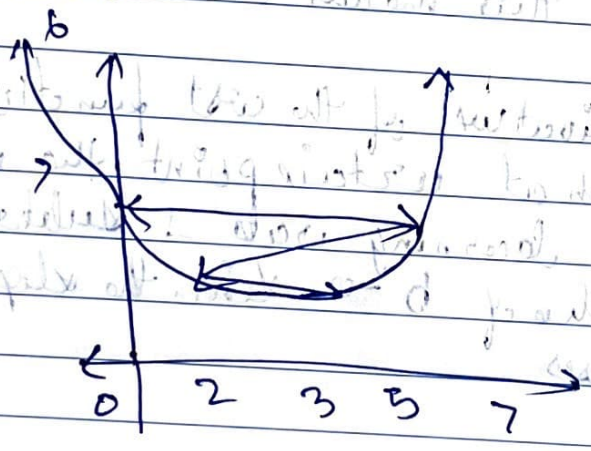
It takes the derivative of the cost function & returns the slope of the graph at certain point. the model multiplies the slope by the learning rate & subtracts the product from current value of b-0. Since the slope is negative, the value of b_0 increases



the value of b-0 becomes the algorithm will now take the derative of cost functions when b-0 is 5 & carry out the process again.



this process will continue until the derivative returns a slope that is very close to 0, signifying that model has reached a near optimal value for the parameter being tuned.



Q4] Explain the following performance evaluation parameters with the help of confusion matrix. Illustrate with appropriate example

⇒ A confusion matrix is a table that is often used to describe the performance of a classification model (or classifier) on a set of test data for which the true values are known

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

F 1 score is a metric that combines precision and recall & equals to the harmonic mean of precision and recall

Using the values of precision & Recall (0.9090 & 0.7692) F1 score = 0.8333 = 83.33%.

e] Specificity

Specificity is defined as the ratio of true Negatives & True negatives + False Positives. We want the value of specificity to be high

$$\text{Specificity} = \frac{\text{True Negative}}{\text{True Negative + False Positive}}$$

Thus specificity using above values will be

$$300/(300+50) = \frac{300}{350} = 85.71\%$$

f] AUC - ROC curve

AUC (Area Under Curve) - ROC (Receiver operating Characteristic) curve is one of the most important evaluation metrices for checking any classification models performance.
It is a plotted between FPR (X-axis) & TPR (Y-axis) if the value is less than 0.5 than the model is even worse than random

guessing model

$$TPR = \frac{TP}{FP + FN}$$

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

5/7/24

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Assignment 2

Q1) Discuss different ensemble learning techniques

=> Ensemble techniques combine individual models together, to improve the stability and predictive power of the model.

Individual models, also know as base models can have a high bias, high variance or both, which in turn is not good. Such models are know as weak models. When weak models are combined they can produce strong models having low bias & low variance.

- A diverse set of models in comparison to single models can make better predictions & this diversification in ML is achieved by a technique called ensemble learning.
- Ensemble methods → sequential

- Some ensemble learning techniques are

a) Max Voting: this method is generally used for classification problems. In this technique, multiple models are used to make prediction by each model are considered as a vote. The prediction we get from the majority of the models is used as a final prediction.

b) Averaging:- Multiple predictions are made for each data point and the average of all predictions is the final predictions.

c) Weighted averaging: this is an extension of

of averaging where all models are assigned weights

Boosting :- It trains multiple models sequentially with each new model focusing on the mistakes made by the previous model focusing on the mistakes made by the previous model. The idea is to correct to errors earlier models. The idea is to correct the errors earlier models & improve the overall prediction accuracy. The final model is usually a weighted average of all the model is usually weighted average of all the models prediction. Boosting can significantly improve model performance and reduce both bias & variance. It is particularly useful for complex datasets

Q2) Explain the following
a) Weak learners & strong learners
→ A weak learner is a model that performs slightly better than random guessing. It is typically a simple model with limited predictive power, often making predictions that are only marginally better than random choice. In decision tree algorithm, a shallow tree with a few nodes and a simple rule based classifier can be considered a weak learner & strong learner is a model that performs significantly better than random guessing. It has a high predictive accuracy and it can capture complex patterns in data
eg :- Complex models like deep neural networks

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

strong learner are often the good ensemble methods. they are built by combining weaker models or using sophisticated algo to achieve high predictive performance

b) Meta learning :
It refers to the process of learning how to learn. It involves developing models that can learn from various learning algorithms or datasets and adapt to new tasks more efficiently. The idea is to improve the learning process itself by using previous experience or data. eg Meta learning can involve learning the best hyperparameters for different algorithms, selecting appropriate models for various tasks or even adapting to new types of data with additional training

c) Random forest :
it is an ensemble learning method specifically designed for classification and regression tasks. It builds a number of decision trees & combines their predictions to make a final decision tree
Working Bagging - Random forest uses the bagging technique to train each decision tree on a different random subset of training data. During the training of each decision tree for regression tasks, the final prediction is the average of the prediction of all trees

Q3] Comp are Bagging, Boosting & Stacking

Bagging	Boosting	Stacking
1] Parallel training (independent models)	sequential training (correct errors)	sequential train (combines predictions)
2] Average or majority voting	Weighted average based on models	Meta learner combines base model predictions
3] Reduces variance	Reduces both bias and variance	Reduces depending on models
4] Less prone to overfitting	It prone to everything	Less prone to overfitting
5] Less computational	can be computational	can be computational
6] Simple to implement and understand	More complex sequential train	Complex, involves multiple model type
7] Eg:- Random forest	eg:- Adaboost	Eg:- model stacking with decision tree, logistic regression

Q4] Explain Stab ost

Q4) Explain Adaboost algorithm

⇒ Adaboost is a boosting algorithm that also works on the principle of stage wise addition method where multiple weak learner models are used to get strong learner models. The value of alpha parameters in this case will be indirectly proportional to the errors of weak learner

- Working

1) Initialization :- All training samples are given equal weights.
2) Training the Algorithm iteratively trains weak learners
3) Weighting : the influence of each weak learner is determined by its accuracy
4) Updating :- Weights of misclassified samples are increased for the next iteration
5) Final model : the final prediction is a weighted vote of all weak learners

- This algorithm enhances performance by focusing on outliers & it is generally more resistant to overfitting than other models

- Advantages
- simple & effective
- no overfitting
- can be used with various types of weak learners
- Disadvantages
- sensitive to noisy data
- It is computationally expensive

14/10/24