CMR INSTITUTE OF TECHNOLOGY

Affiliated to Visvesvaraya Technological University(VTU)
Sri Nivasa Reddy Layout, AECS Layout,
Marathahalli,Bengaluru, Karnataka 560037

Department of

Information Science and Engineering

LAB MANUAL

MACHINE LEARNING LAB (BCSL606)

Academic Year : 2025-26

Semester-VI

Scheme-2022

Course Objectives (Defined by the university):

At the end of the course, the student will be able to :

1. Describe the machine learning techniques, their types, and the data analysis framework.

2. Apply mathematical concepts for feature engineering and perform dimensionality reduction to enhance model performance.

3. Develop similarity-based learning models and regression models for solving classification and prediction tasks.

4. Build probabilistic learning models and design neural network models using perceptrons and multilayer architectures

5. Utilize clustering algorithms to identify patterns in data and implement reinforcement learning techniques.

List of Problems/Experiments

	Experiments
	List of problems for which students should develop the program and execute it in the laboratory
1	Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.
2	Develop a program to Compute the correlation matrix to understand the relationships between pairs offeatures. Visualize the correlation matrix using a heatmap to know which variables have strongpositive/negative correlations. Create a pair plot to visualize pairwise relationships between features. Use California Housing dataset.
3	Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.
4	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.
5	Develop a program to implement k-Nearest Neighbour algorithm to classify the randomly generated 100 values of x in the range of [0,1]. Perform the following based on dataset generated. a. Label the first 50 points {x1,.....,x50} as follows: if (xi ≤ 0.5), then xi ∈ Class1, else xi ∈ Class1 b. Classify the remaining points, x51, ... ,x100 using KNN. Perform this for k=1,2,3,4,5,20,30
6	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs
7	Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression.
8	Develop a program to demonstrate the working of the decision tree algorithm. Use Breast Cancer Data set for building the decision tree and apply this knowledge to classify a new sample.

9	Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets.
10	Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result.

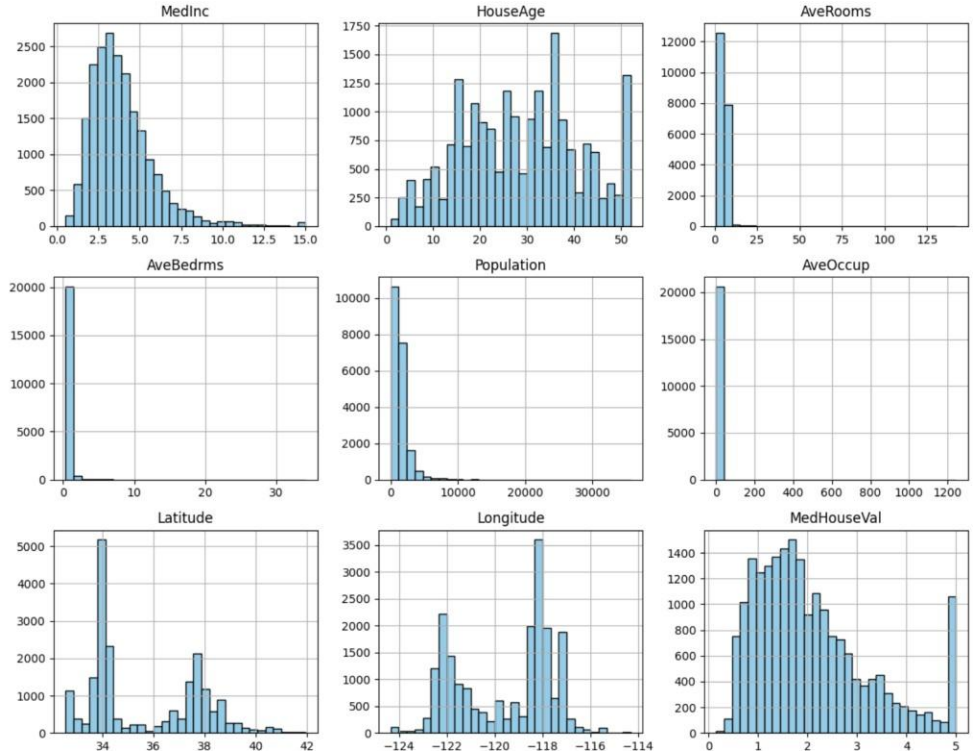
Experiment 1: Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import fetch_california_housing

# Load the California Housing dataset

data = fetch_california_housing(as_frame=True)

df = data.frame

# Display the first few rows of the dataset

print("Dataset Sample:")

print(df.head())

# Plot histograms for all numerical features

def plot_histograms(df):

   df.hist(figsize=(12, 10), bins=30, color='skyblue', edgecolor='black')

   plt.suptitle('Histograms of Numerical Features', fontsize=16)

   plt.tight_layout(rect=[0, 0, 1, 0.97])

   plt.show()

# Plot box plots for all numerical features

def plot_boxplots(df):

   plt.figure(figsize=(14, 10))

   for i, column in enumerate(df.columns, 1):

       plt.subplot(3, 3, i)
```

```
        sns.boxplot(y=df[column], color='skyblue')
        plt.title(f"Box Plot of {column}", fontsize=12)
        plt.tight_layout()
    plt.show()


# Analyze distribution and detect outliers
def analyze_features(df):
    print("\nFeature Analysis:")
    for column in df.columns:
        print(f"\nFeature: {column}")
        print(f"Mean: {df[column].mean():.2f}, Median: {df[column].median():.2f}, Std Dev: {df[column].std():.2f}")
        q1 = df[column].quantile(0.25)
        q3 = df[column].quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr
        outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
        print(f"Number of Outliers: {len(outliers)}")
# Plot histograms and boxplots, and analyze features
plot_histograms(df)
plot_boxplots(df)
analyze_features(df)
```

Output:


MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseVal

Experiment 2: Develop a program to Compute the correlation matrix to understand the relationships between pairs offeatures. Visualize the correlation matrix using a heatmap to know which variables have strongpositive/negative correlations. Create a pair plot to visualize pairwise relationships between features. Use California Housing dataset.

Code:

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import fetch_california_housing

# Load the California Housing dataset

data = fetch_california_housing(as_frame=True)

df = data.frame

# Display the first few rows of the dataset

print("Dataset Sample:")

print(df.head())

# Compute the correlation matrix

correlation_matrix = df.corr()

# Print the correlation matrix

print("\nCorrelation Matrix:")

print(correlation_matrix)

# Visualize the correlation matrix using a heatmap

def plot_heatmap(corr_matrix):

    plt.figure(figsize=(10, 8))

    sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm",
cbar=True, square=True, linewidths=0.5)

    plt.title("Correlation Matrix Heatmap", fontsize=16)

    plt.show()
```

```
# Create a pair plot to visualize pairwise relationships

def plot_pairplot(df):

   sns.pairplot(df, diag_kind="kde", corner=True, plot_kws={'alpha': 0.5},
diag_kws={'fill': True})

   plt.suptitle("Pair Plot of Numerical Features", y=1.02, fontsize=16)

   plt.show()

# Plot the heatmap and pair plot

plot_heatmap(correlation_matrix)

plot_pairplot(df)
```

Output

```
Dataset Sample:
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude  MedHouseVal
0    -122.23        4.526
1    -122.22        3.585
2    -122.24        3.521
3    -122.25        3.413
4    -122.25        3.422

Correlation Matrix:
              MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  \
MedInc       1.000000 -0.119034  0.326895  -0.062040    0.004834  0.018766
HouseAge    -0.119034  1.000000 -0.153277  -0.077747   -0.296244  0.013191
AveRooms     0.326895 -0.153277  1.000000   0.847621   -0.072213 -0.004852
AveBedrms   -0.062040 -0.077747  0.847621   1.000000   -0.066197 -0.006181
Population   0.004834 -0.296244 -0.072213  -0.066197    1.000000  0.069863
AveOccup     0.018766  0.013191 -0.004852  -0.006181    0.069863  1.000000
Latitude    -0.079809  0.011173  0.106389   0.069721   -0.108785  0.002366
Longitude   -0.015176 -0.108197 -0.027540   0.013344    0.099773  0.002476
MedHouseVal  0.688075  0.105623  0.151948  -0.046701   -0.024650 -0.023737

              Latitude  Longitude  MedHouseVal
MedInc      -0.079809  -0.015176     0.688075
HouseAge     0.011173  -0.108197     0.105623
AveRooms     0.106389  -0.027540     0.151948
AveBedrms    0.069721   0.013344    -0.046701
Population  -0.108785   0.099773    -0.024650
AveOccup     0.002366   0.002476    -0.023737
Latitude     1.000000  -0.924664    -0.144160
Longitude   -0.924664   1.000000    -0.045967
MedHouseVal -0.144160  -0.045967     1.000000
```

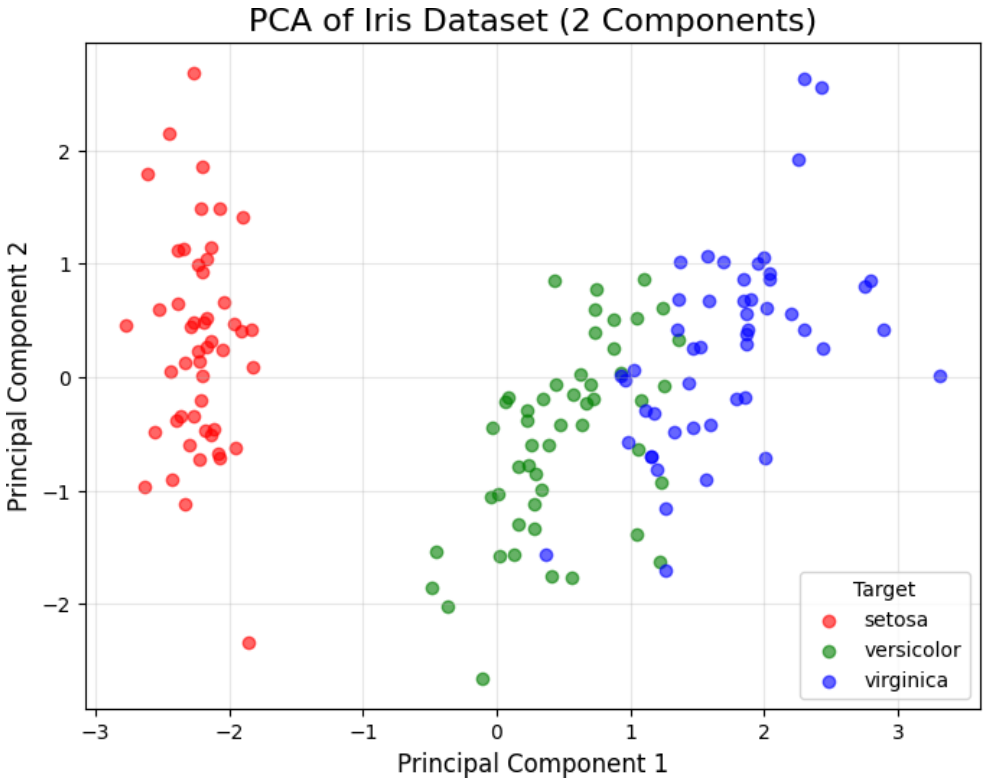
Experiment 3: Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.

Code:

```
import pandas as pd

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

iris = load_iris()

X = iris.data

y = iris.target

target_names = iris.target_names

# Standardize the features (mean = 0, variance = 1)

scaler = StandardScaler()

X_standardized = scaler.fit_transform(X)

# Apply PCA to reduce dimensions from 4 to 2

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_standardized)

# Create a DataFrame for the PCA results

pca_df = pd.DataFrame(data=X_pca, columns=['Principal Component 1',
'Principal Component 2'])

pca_df['Target'] = y

# Plot the PCA results

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

```
for target, color, label in zip(range(len(target_names)), ['r', 'g', 'b'],
target_names):
   plt.scatter(
       pca_df.loc[pca_df['Target'] == target, 'Principal Component 1'],
       pca_df.loc[pca_df['Target'] == target, 'Principal Component 2'],
       color=color,
       alpha=0.6,
       label=label
   )
plt.title('PCA of Iris Dataset (2 Components)', fontsize=16)
plt.xlabel('Principal Component 1', fontsize=12)
plt.ylabel('Principal Component 2', fontsize=12)
plt.legend(title='Target', loc='best')
plt.grid(alpha=0.3)
plt.show()
```

Output:


PCA of Iris Dataset (2 Components)
Principal Component 2
Principal Component 1
Target
setosa
versicolor
virginica
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

Experiment 4: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

Code:

```
import pandas as pd

# Define the data as a dictionary

data = {

   "Weather": ["Sunny", "Sunny", "Rainy", "Sunny"],

   "Temperature": ["Warm", "Warm", "Cold", "Warm"],

   "Humidity": ["Normal", "High", "High", "High"],

   "Wind": ["Strong", "Strong", "Strong", "Weak"],

   "PlayTennis": ["Yes", "Yes", "No", "Yes"]

}

# Convert the data into a DataFrame

df = pd.DataFrame(data)

# Save the DataFrame to a CSV file

file_path = "training_data.csv"

df.to_csv(file_path, index=False)

print(f"CSV file 'training_data.csv' has been created successfully!")

def find_s_algorithm(data):

   # Extract features and labels

   features = data.iloc[:, :-1].values    # All columns except the last
(attributes)

   labels = data.iloc[:, -1].values     # The last column (class/label)

   # Initialize the hypothesis as the first positive example

   hypothesis = None
```

```
    for i, label in enumerate(labels):
        if label == "Yes":  # Look for a positive example
            hypothesis = features[i].copy()
            break

    # If no positive examples, return empty hypothesis
    if hypothesis is None:
        return "No positive examples found."

    # Generalize hypothesis for each positive example
    for i, label in enumerate(labels):
        if label == "Yes":  # Process only positive examples
            for j in range(len(hypothesis)):
                if hypothesis[j] != features[i][j]:  # If attributes differ, make it '?'
                    hypothesis[j] = '?'

    return hypothesis

# Load training data from a CSV file
file_path = "training_data.csv"  # Replace with your CSV file path
data = pd.read_csv(file_path)

# Display the training data
print("Training Data:")
print(data)
```

```
# Run the Find-S algorithm

final_hypothesis = find_s_algorithm(data)



# Output the final hypothesis

print("\nFinal Hypothesis:")

print(final_hypothesis)
```

Output

```
Training Data:
  Weather Temperature Humidity    Wind PlayTennis
0   Sunny        Warm   Normal  Strong        Yes
1   Sunny        Warm     High  Strong        Yes
2   Rainy        Cold     High  Strong         No
3   Sunny        Warm     High    Weak        Yes

Final Hypothesis:
['Sunny' 'Warm' '?' '?']
```

Experiment 5: Develop a program to implement k-Nearest Neighbour algorithm to classify the randomly generated 100 values of x in the range of [0,1]. Perform the following based on dataset generated.

a. Label the first 50 points {x1,.....,x50} as follows: if (xi ≤ 0.5), then xi ∈ Class1, else xi ∈ Class1

b. Classify the remaining points, x51,..... ,x100 using KNN. Perform this for k=1,2,3,4,5,20,30

Employee(E_id, E_name, Age, Salary)

Code:

```
import numpy as np

from sklearn.neighbors import KNeighborsClassifier

# Step 1: Generate random dataset

np.random.seed(42)  # For reproducibility

x = np.random.rand(100)  # 100 random values in the range [0, 1]

# Step 2: Label the first 50 points

labels = np.where(x[:50] <= 0.5, 1, 2)  # Class1 = 1, Class2 = 2

# Combine the labeled data into a training set

x_train = x[:50].reshape(-1, 1)

y_train = labels

# The remaining 50 points to classify

x_test = x[50:].reshape(-1, 1)

# Step 3: k-NN classification

k_values = [1, 2, 3, 4, 5, 20, 30]

print("k-NN Classification Results:")

for k in k_values:

    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(x_train, y_train)  # Train the k-NN model

    y_pred = knn.predict(x_test)  # Predict classes for the test points

    # Output the predictions for this k value
```

```
    print(f"\nk = {k}:")

    print(f"Predicted Classes: {y_pred}")
```

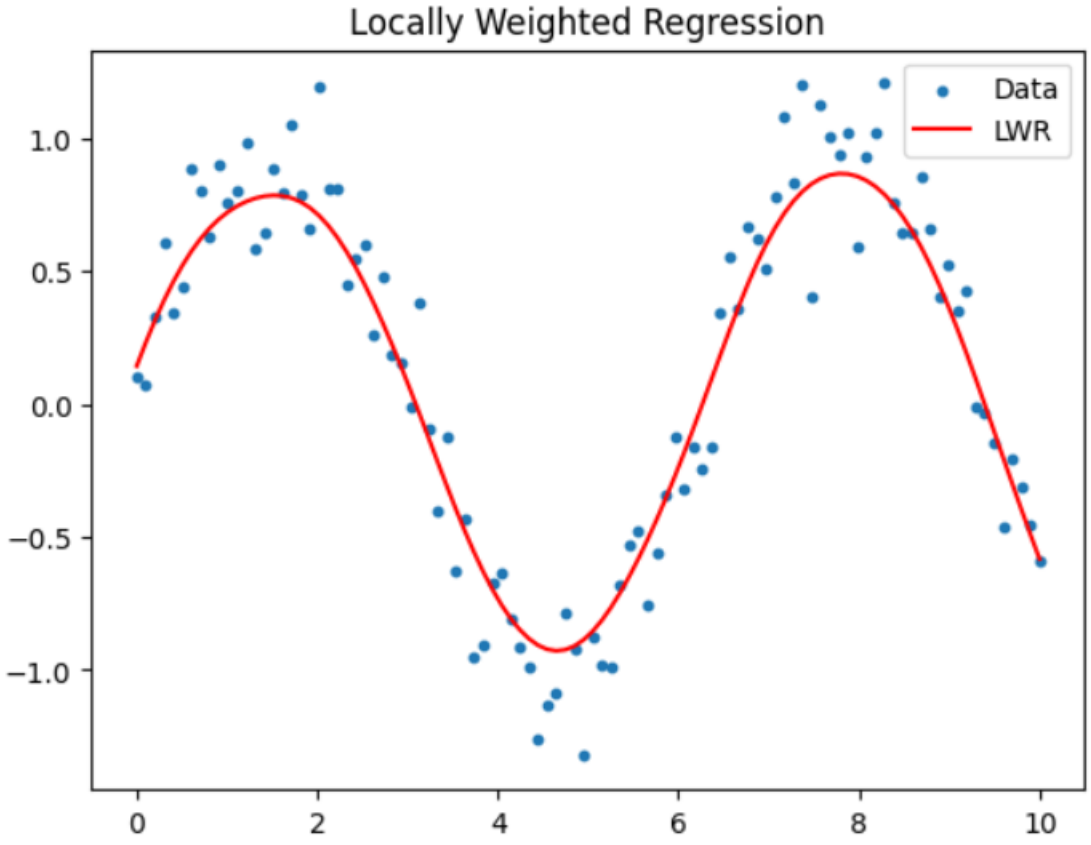
Output

```
k-NN Classification Results:

k = 1:
Predicted Classes: [2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 2
 2 2 1 1 2 2 2 2 1 2 1 1 1]

k = 2:
Predicted Classes: [2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 2
 2 2 1 1 2 2 2 2 1 2 1 1 1]

k = 3:
Predicted Classes: [2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 2
 2 2 1 1 2 2 2 2 2 2 1 1 1]

k = 4:
Predicted Classes: [2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 2
 2 2 1 1 2 2 2 2 2 2 1 1 1]

k = 5:
Predicted Classes: [2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 2
 2 2 1 1 2 2 2 2 2 2 1 1 1]

k = 20:
Predicted Classes: [2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 2
 2 2 1 1 2 2 2 2 2 2 1 1 1]

k = 30:
Predicted Classes: [2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 2
 2 2 1 1 2 2 2 2 1 2 1 1 1]
```

Experiment 6: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

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

# Generate data
np.random.seed(42)
X = np.linspace(0, 10, 100).reshape(-1, 1)
y = np.sin(X).ravel() + np.random.normal(0, 0.2, 100)

# LWR function
def lwr(X, y, x0, tau):
    w = np.exp(-((X - x0)**2) / (2 * tau**2)).flatten()
    W = np.diag(w)
    Xb = np.hstack((np.ones_like(X), X))
    theta = np.linalg.pinv(Xb.T @ W @ Xb) @ Xb.T @ W @ y
    return np.dot([1, x0], theta)

# Predict for all
tau = 0.5
y_pred = np.array([lwr(X, y, x[0], tau) for x in X])

# Plot
plt.scatter(X, y, s=10, label="Data")
plt.plot(X, y_pred, color='red', label="LWR")
plt.legend()
plt.title("Locally Weighted Regression")
plt.show()
```

Output



Experiment 7:. Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression.

Code:

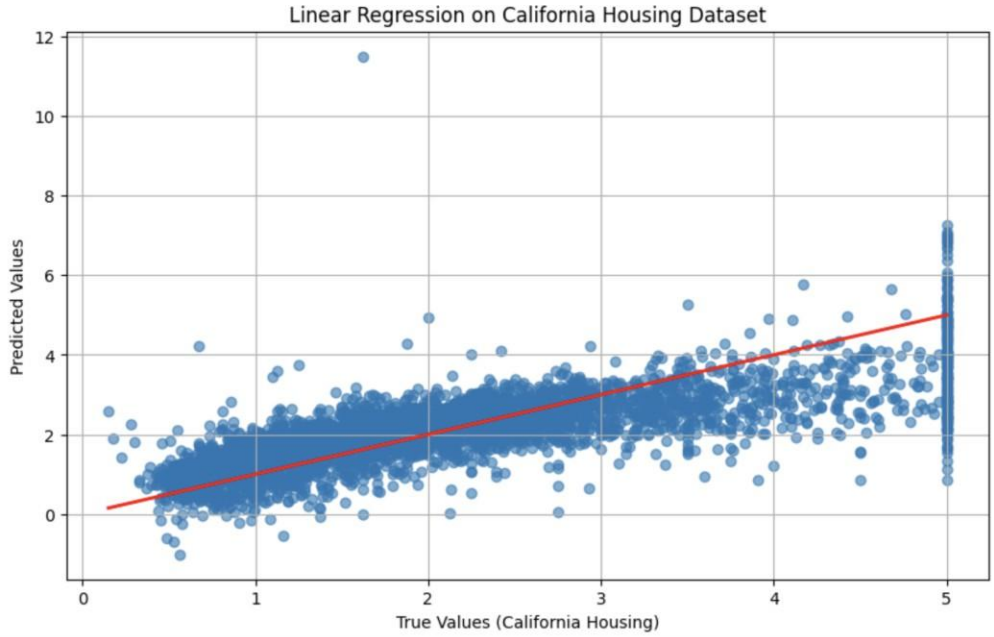
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing, fetch_openml
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# --- Linear Regression with California Housing Dataset ---
def linear_regression_california():
    # Load California Housing Dataset
    cali_housing = fetch_california_housing()
    X = cali_housing.data
    y = cali_housing.target

    # Split into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Train a linear regression model
    lin_reg = LinearRegression()
    lin_reg.fit(X_train, y_train)

    # Predictions
    y_pred = lin_reg.predict(X_test)

    # Performance metrics
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Plotting
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred, alpha=0.6)
```

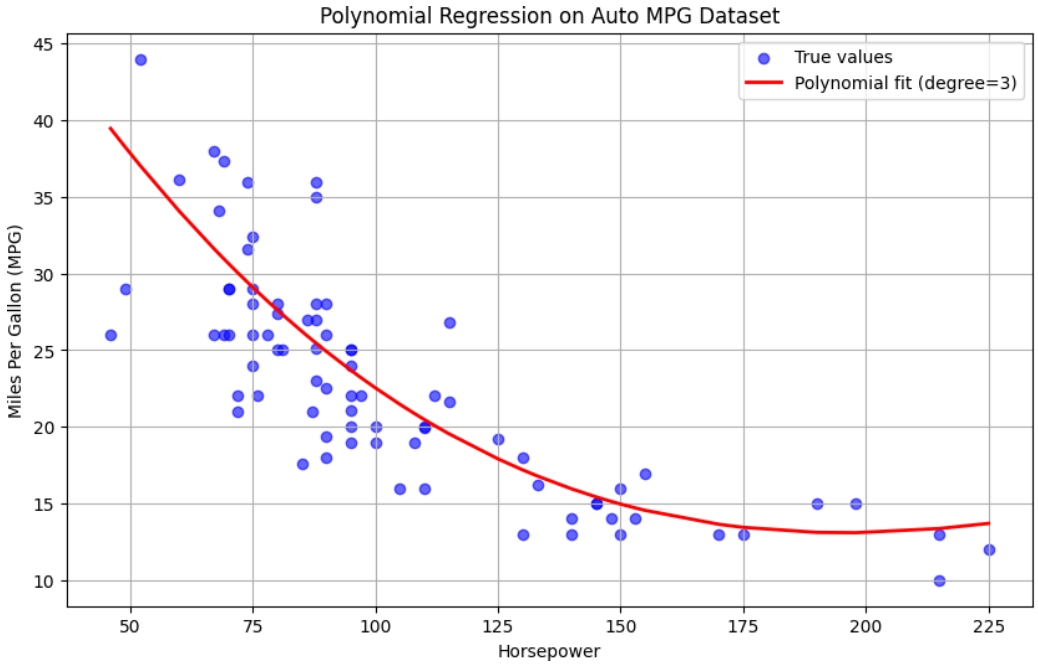
```
    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color="red", linewidth=2)
    plt.xlabel("True Values (California Housing)")
    plt.ylabel("Predicted Values")
    plt.title("Linear Regression on California Housing Dataset")
    plt.grid(True)
    plt.show()
    print(f"Linear Regression Results:")
    print(f"Mean Squared Error: {mse:.2f}")
    print(f"R^2 Score: {r2:.2f}")

# --- Polynomial Regression with Auto MPG Dataset ---
def polynomial_regression_auto_mpg():
    # Load Auto MPG Dataset
    auto_mpg = fetch_openml(name="autoMpg", version=1, as_frame=True)
    data = auto_mpg.data
    target = auto_mpg.target

    # Remove rows with missing 'horsepower' values from both data and
target
    data = data.dropna(subset=["horsepower"])
    target = target.loc[data.index]  # Ensure target is aligned with
filtered data
    # Select feature (Horsepower) and target (MPG)
    X_hp = data[["horsepower"]].astype(float)  # Horsepower as feature
    y_mpg = target.astype(float)  # Miles per gallon as target

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X_hp, y_mpg,
test_size=0.2, random_state=42)

    # Polynomial regression (degree=3)
    poly_features = PolynomialFeatures(degree=3)
    X_train_poly = poly_features.fit_transform(X_train)
    X_test_poly = poly_features.transform(X_test)

    # Train a linear regression model on transformed polynomial features
    lr_poly = LinearRegression()
    lr_poly.fit(X_train_poly, y_train)
```

```
    # Predictions
    y_pred_poly = lr_poly.predict(X_test_poly)
    # Performance metrics
    mse_poly = mean_squared_error(y_test, y_pred_poly)
    r2_poly = r2_score(y_test, y_pred_poly)
    # Sort for plotting
    X_test_sorted, y_test_sorted = zip(*sorted(zip(X_test.values.flatten(),
y_test)))
    y_pred_sorted =
lr_poly.predict(poly_features.transform(np.array(X_test_sorted).reshape(-
1, 1)))
    # Plot
    plt.figure(figsize=(10, 6))
    plt.scatter(X_test, y_test, color="blue", label="True values",
alpha=0.6)
    plt.plot(X_test_sorted, y_pred_sorted, color="red", label="Polynomial
fit (degree=3)", linewidth=2)
    plt.xlabel("Horsepower")
    plt.ylabel("Miles Per Gallon (MPG)")
    plt.title("Polynomial Regression on Auto MPG Dataset")
    plt.legend()
    plt.grid(True)
    plt.show()
    print(f"Polynomial Regression Results (Degree=3):")
    print(f"Mean Squared Error: {mse_poly:.2f}")
    print(f"R^2 Score: {r2_poly:.2f}")
# Run both demonstrations
def run_models():
    linear_regression_california()
    polynomial_regression_auto_mpg()
# Execute the functions
run_models()
```

Read Operation:

Output


Linear Regression on California Housing Dataset
Predicted Values
True Values (California Housing)
0
1
2
3
4
5
0
2
4
6
8
10
12


Polynomial Regression on Auto MPG Dataset
True values
Polynomial fit (degree=3)
Miles Per Gallon (MPG)
Horsepower
10
15
20
25
30
35
40
45
50
75
100
125
150
175
200
225

Experiment 8:. Develop a program to demonstrate the working of the decision tree algorithm. Use Breast Cancer Data set for building the decision tree and apply this knowledge to classify a new sample.

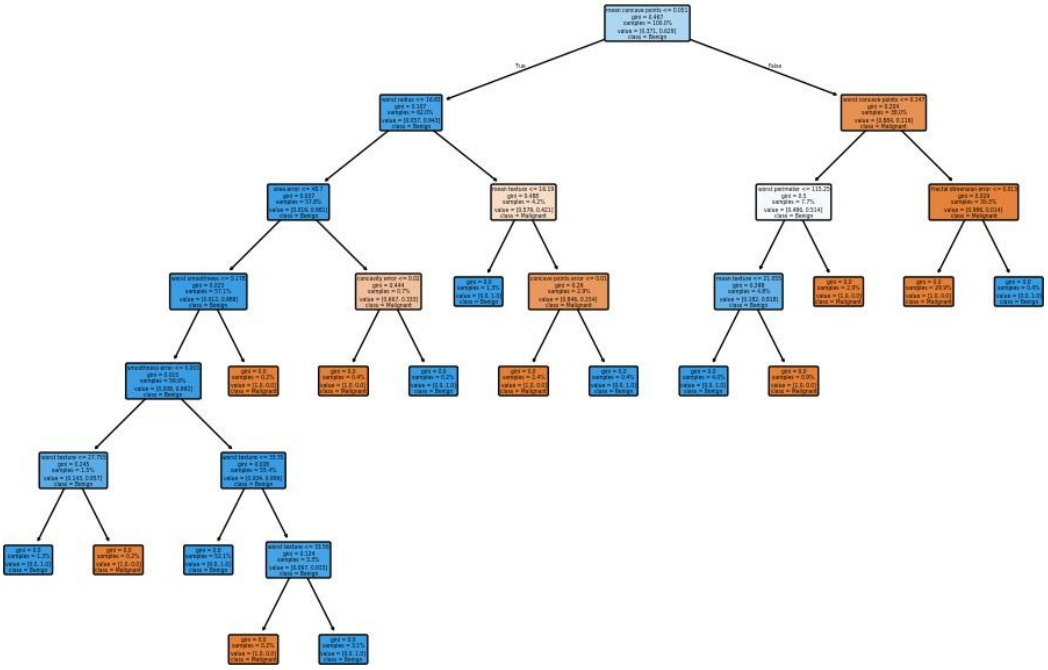
Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report

# Load the Breast Cancer dataset
def load_and_preprocess_data():
    cancer_data = load_breast_cancer()
    X = cancer_data.data
    y = cancer_data.target
    feature_names = cancer_data.feature_names
    target_names = cancer_data.target_names
    return X, y, feature_names, target_names

# Train a Decision Tree classifier
def train_decision_tree(X_train, y_train):
    clf = DecisionTreeClassifier(random_state=42)
    clf.fit(X_train, y_train)
    return clf

# Plot the Decision Tree
def plot_decision_tree(clf, feature_names):
    plt.figure(figsize=(12, 8))
    plot_tree(clf, filled=True, feature_names=feature_names,
class_names=["Malignant", "Benign"], rounded=True, proportion=True)
    plt.title("Decision Tree for Breast Cancer Classification")
    plt.show()

# Classify a new sample
def classify_new_sample(clf, sample):
    sample = np.array(sample).reshape(1, -1)
    prediction = clf.predict(sample)
```

```
    return prediction

# Main function to demonstrate Decision Tree Algorithm
def main():
    # Load and preprocess data
    X, y, feature_names, target_names = load_and_preprocess_data()

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Train the Decision Tree model
    clf = train_decision_tree(X_train, y_train)

    # Predict on the test set
    y_pred = clf.predict(X_test)

    # Print performance metrics
    print("Accuracy Score:", accuracy_score(y_test, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, target_names=target_names))

    # Plot the decision tree
    plot_decision_tree(clf, feature_names)

    # Classify a new sample (e.g., random sample from the test set)
    sample = X_test[0]  # You can replace this with any sample
    prediction = classify_new_sample(clf, sample)
    print(f"\nClassified sample: {'Benign' if prediction == 1 else
'Malignant'}")

# Run the main function
if __name__== "__main__":
    main()
```

Output

Decision Tree for Breast Cancer Classification



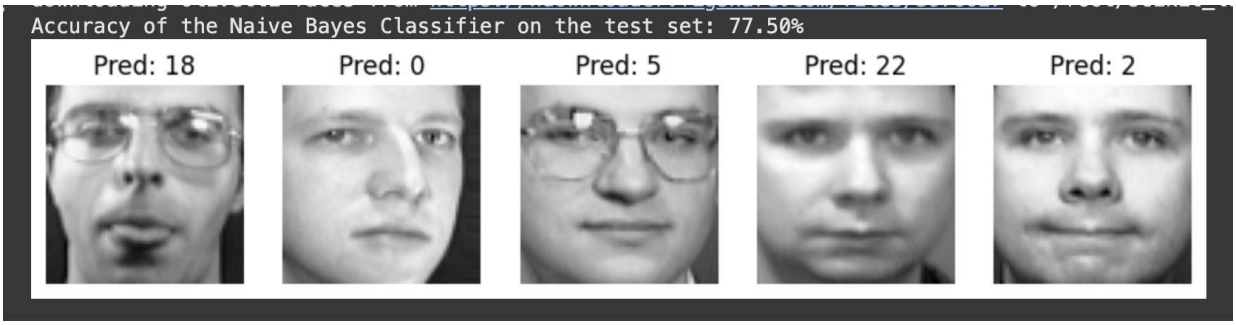
Experiment 9:. Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets.

Code:

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Function to load and preprocess the Olivetti Face Dataset
def load_and_preprocess_data():
   faces_data = datasets.fetch_olivetti_faces(shuffle=True,
random_state=42)
   X = faces_data.data  # 4096 features for each face (64x64 pixel images
flattened)
   y = faces_data.target  # Labels for each person (40 classes)
   return X, y

# Function to split the data into training and testing sets
def split_data(X, y, test_size=0.2):
   X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size, random_state=42)
   return X_train, X_test, y_train, y_test

# Function to train a Naive Bayes classifier
def train_naive_bayes(X_train, y_train):
   nb_classifier = GaussianNB()
   nb_classifier.fit(X_train, y_train)
   return nb_classifier

# Function to predict and calculate accuracy
def predict_and_evaluate(nb_classifier, X_test, y_test):
   y_pred = nb_classifier.predict(X_test)
   accuracy = accuracy_score(y_test, y_pred)
   return y_pred, accuracy
```

```
# Function to visualize a few test samples with predicted labels
def visualize_predictions(X_test, y_pred, n_samples=5):
    plt.figure(figsize=(10, 5))
    for i in range(n_samples):
        plt.subplot(1, n_samples, i + 1)
        plt.imshow(X_test[i].reshape(64, 64), cmap='gray')
        plt.title(f"Pred: {y_pred[i]}")
        plt.axis('off')
    plt.show()

# Main function to run the program
def main():
    # Load and preprocess data
    X, y = load_and_preprocess_data()

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = split_data(X, y)

    # Train the Naive Bayes classifier
    nb_classifier = train_naive_bayes(X_train, y_train)

    # Predict and evaluate the model
    y_pred, accuracy = predict_and_evaluate(nb_classifier, X_test, y_test)
    print(f"Accuracy of the Naive Bayes Classifier on the test set:
{accuracy * 100:.2f}%")

    # Visualize predictions on the test set
    visualize_predictions(X_test, y_pred)

# Run the program
if __name__== "__main__":
    main()
```

Output


Accuracy of the Naive Bayes Classifier on the test set: 77.50%
Pred: 18
Pred: 0
Pred: 5
Pred: 22
Pred: 2

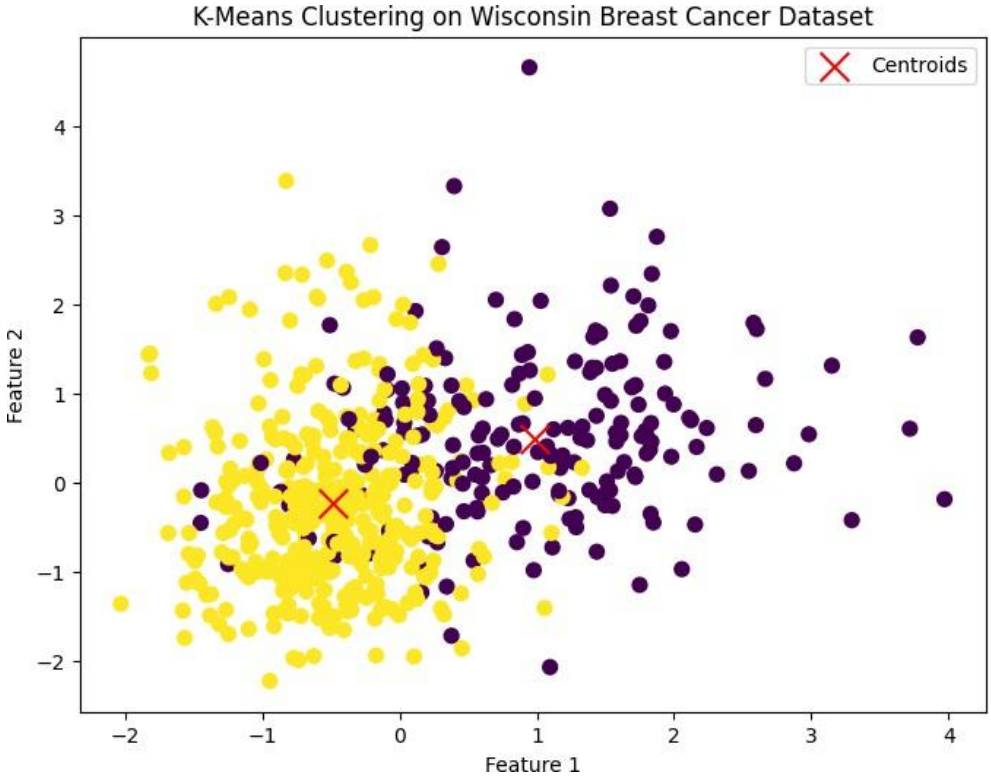
Experiment 10:. Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result.

Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Function to load and preprocess the dataset
def load_and_preprocess_data():
    # Load the Wisconsin Breast Cancer dataset
    data = load_breast_cancer()

    # Extract features (X) and labels (y)
    X = data.data
    y = data.target

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

    return X_scaled, y

# Function to apply K-Means clustering
def apply_kmeans(X, n_clusters=2):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(X)

    # Return the cluster centers and labels
    return kmeans.cluster_centers_, kmeans.labels_

# Function to visualize the clustering result
def visualize_clusters(X, labels, centers):
    plt.figure(figsize=(8, 6))

    # Plot the data points, colored by cluster label
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)
```

```
    # Plot the cluster centers
    plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', s=200,
label='Centroids')

    plt.title('K-Means Clustering on Wisconsin Breast Cancer Dataset')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.show()

# Main function to execute the program
def main():
    # Load and preprocess the data
    X, y = load_and_preprocess_data()

    # Apply K-Means clustering (using 2 clusters for simplicity)
    centers, labels = apply_kmeans(X, n_clusters=2)

    # Visualize the clustering results
    visualize_clusters(X, labels, centers)

# Run the program
if __name__== "__main__":
    main()
```

Output


K-Means Clustering on Wisconsin Breast Cancer Dataset
Centroids
Feature 2
Feature 1
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4

Extra Program 1: Develop a program to implement the Support Vector Machine (SVM) algorithm for binary classification. Use the Pima Indians Diabetes dataset for training and evaluation. Experiment with different kernel functions (linear, RBF, polynomial) and regularization parameters to find the best model.

Code:

```
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

# Load the Pima Indians Diabetes dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-
indians-diabetes.data.csv"
columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
data = pd.read_csv(url, names=columns)

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

# Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# Define the SVM models with different kernel functions
kernels = ['linear', 'rbf', 'poly']
C_values = [0.1, 1, 10]

# Store results
results = []
```

```
# Train and evaluate the models
for kernel in kernels:
   for C in C_values:
       print(f"Training SVM with {kernel} kernel and C={C}...")

       # Create and train the SVM model
       model = svm.SVC(kernel=kernel, C=C)
       model.fit(X_train, y_train)

       # Predict on the test set
       y_pred = model.predict(X_test)

       # Calculate accuracy
       accuracy = accuracy_score(y_test, y_pred)
       results.append((kernel, C, accuracy))

# Find the best model
best_model = max(results, key=lambda x: x[2])

# Print results
print("\nModel Evaluation Results:")
for result in results:
   print(f"Kernel: {result[0]}, C: {result[1]}, Accuracy:
{result[2]:.4f}")

print("\nBest Model:")
print(f"Kernel: {best_model[0]}, C: {best_model[1]}, Accuracy:
{best_model[2]:.4f}")
```

Output

```
Training SVM with linear kernel and C=0.1...
Training SVM with linear kernel and C=1...
Training SVM with linear kernel and C=10...
Training SVM with rbf kernel and C=0.1...
Training SVM with rbf kernel and C=1...
Training SVM with rbf kernel and C=10...
Training SVM with poly kernel and C=0.1...
Training SVM with poly kernel and C=1...
Training SVM with poly kernel and C=10...

Model Evaluation Results:
Kernel: linear, C: 0.1, Accuracy: 0.7597
Kernel: linear, C: 1, Accuracy: 0.7597
Kernel: linear, C: 10, Accuracy: 0.7597
Kernel: rbf, C: 0.1, Accuracy: 0.7532
Kernel: rbf, C: 1, Accuracy: 0.7273
Kernel: rbf, C: 10, Accuracy: 0.7078
Kernel: poly, C: 0.1, Accuracy: 0.7273
Kernel: poly, C: 1, Accuracy: 0.7468
Kernel: poly, C: 10, Accuracy: 0.7597

Best Model:
Kernel: linear, C: 0.1, Accuracy: 0.7597
```

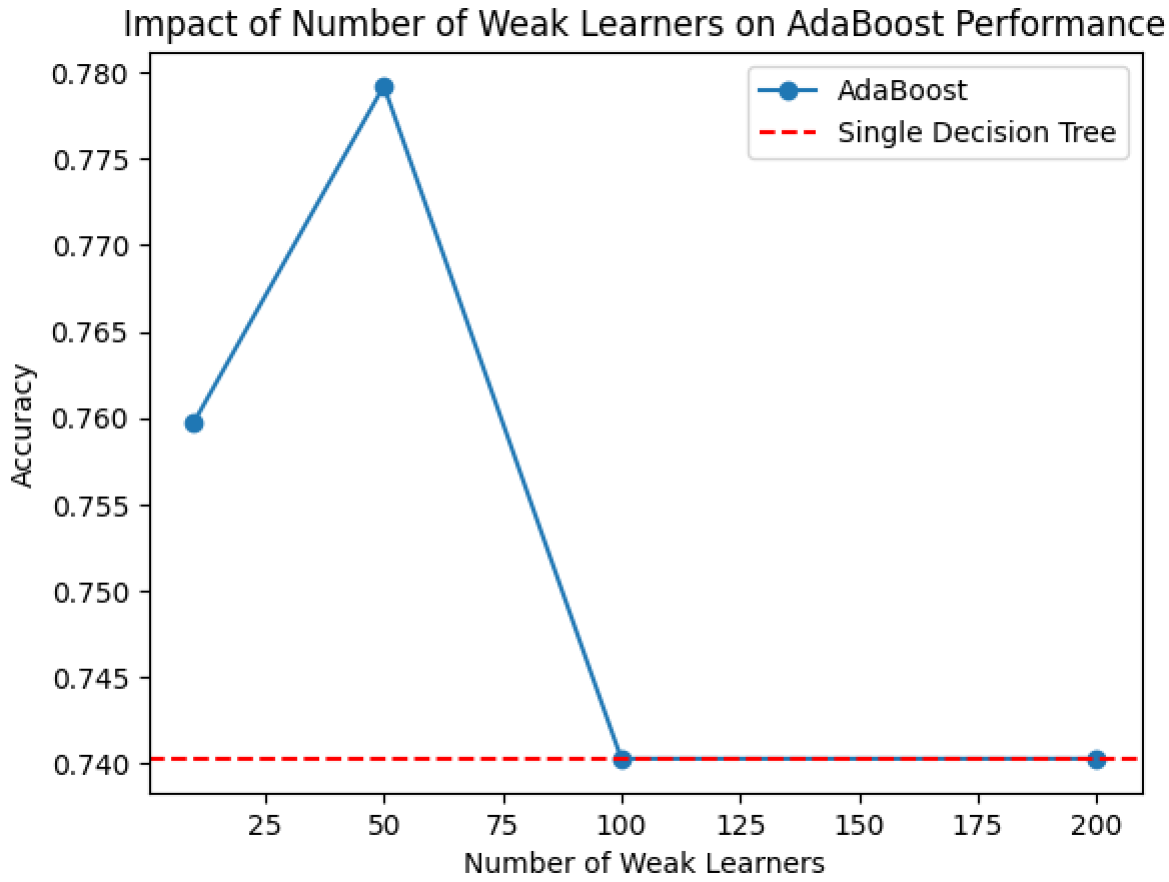
Extra Program 2: Develop a program to implement the AdaBoost algorithm for classification. Use the Pima Indians Diabetes dataset for training and evaluation. Analyze the impact of the number of weak learners on the model's performance and compare the performance of AdaBoost with a single decision tree classifier.

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the dataset (Pima Indians Diabetes dataset)
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
column_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
data = pd.read_csv(url, names=column_names)

# Split into features and target
X = data.drop('Outcome', axis=1)
y = data['Outcome']

# Normalize the data (important for algorithms like AdaBoost)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Initialize the base classifier (Decision Tree)
base_estimator = DecisionTreeClassifier(max_depth=1)

# AdaBoost with different numbers of weak learners (n_estimators)
n_estimators_list = [10, 50, 100, 200]
```

```
adaboost_scores = []

for n_estimators in n_estimators_list:
    # Train AdaBoost with the corrected parameter name 'estimator'
    adaboost = AdaBoostClassifier(estimator=base_estimator,
n_estimators=n_estimators, random_state=42)
    adaboost.fit(X_train, y_train)

    # Evaluate the performance
    y_pred_adaboost = adaboost.predict(X_test)
    report = classification_report(y_test, y_pred_adaboost,
output_dict=True)
    adaboost_scores.append(report['accuracy'])  # Collecting accuracy for
comparison

# Evaluate performance of a single Decision Tree
single_tree = DecisionTreeClassifier(max_depth=1, random_state=42)
single_tree.fit(X_train, y_train)
y_pred_tree = single_tree.predict(X_test)
single_tree_report = classification_report(y_test, y_pred_tree,
output_dict=True)

# Display performance comparison between AdaBoost and Single Decision Tree
print(f"Single Decision Tree Performance:\n{classification_report(y_test,
y_pred_tree)}")
print("\nAdaBoost Performance with different numbers of weak learners:")
for i, n_estimators in enumerate(n_estimators_list):
    print(f"\nAdaBoost with {n_estimators} weak learners:")
    print(f"Accuracy: {adaboost_scores[i]}")

# Plot the impact of number of weak learners on AdaBoost's performance
plt.plot(n_estimators_list, adaboost_scores, marker='o', label='AdaBoost')
plt.axhline(y=single_tree_report['accuracy'], color='r', linestyle='--',
label='Single Decision Tree')
plt.xlabel('Number of Weak Learners')
plt.ylabel('Accuracy')
plt.title('Impact of Number of Weak Learners on AdaBoost Performance')
plt.legend()
plt.show()
```

Output


Impact of Number of Weak Learners on AdaBoost Performance
AdaBoost
Single Decision Tree
Accuracy
0.780
0.775
0.770
0.765
0.760
0.755
0.750
0.745
0.740
25
50
75
100
125
150
175
200
Number of Weak Learners