Week 2 Assignment – Logistic Regression And SVM

Module: CS7CS4/CSU44061 Machine Learning

Student Name : Indrajeet Kumar
Student Number : 23345983

Part A : Logistic Regression

1. Visualise the data

The dataset was imported from a CSV file using the Python pandas library, and it was stored in a DataFrame called ‘data’ with the following code:
data = pd.read_csv('week2.csv')

Following this, the 'X1' and 'X2' columns are designated as feature variables, while the 'y' column is designated as the target variable.

We generated a scatter plot to visually represent the dataset. In this plot, we mapped the values of the first feature (X1) on the X-axis and the values of the second feature (X2) on the Y-axis. For data points with a target value of +1, we marked them with a '+' symbol, while data points with a target value of -1 were represented by a 'o' symbol. To provide clarity, we included a legend that explains the markers and colors used for +1 and -1. Figure 1 displays this data visualization

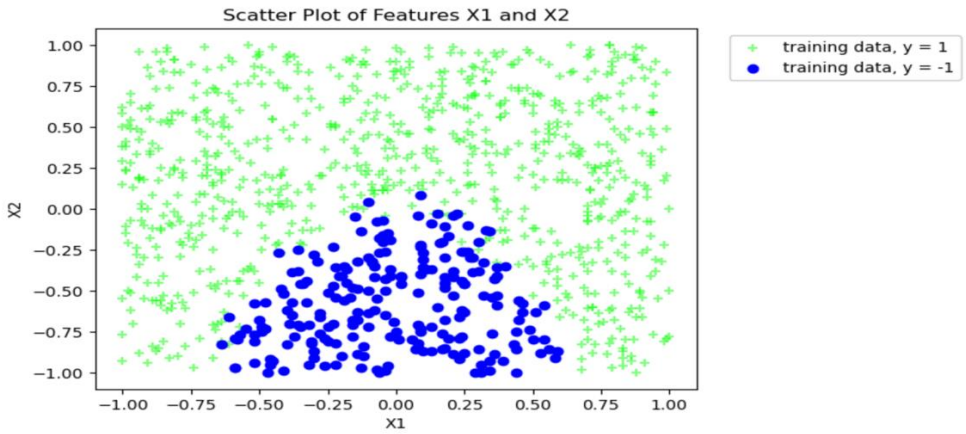

Figure 1: Visualisation Of Data

2. Logistic Regression Model

The logistic regression function available in the sklearn library was utilized to train a logistic regression classifier with the dataset. The following code snippet illustrates the process:

Parameter Values of Trained Model:
In the logistic regression model, Feature2 has the greatest influence on the prediction. A one-unit increase in Feature2 leads to an approximately 3.54 increase in the log-odds of the predicted outcome. This means that raising Feature2 significantly enhances the likelihood of the predicted outcome. Feature1 also positively influences the prediction, but its impact is smaller compared to Feature2. A one-unit increase in Feature1 results in an approximately 0.16 increase in the log-odds of the predicted outcome. While it contributes positively, it doesn't have as substantial an effect on the prediction as Feature2.

3. Make Predictions

The logistic classifier, after being trained, was utilized to predict the target values within the training dataset. Decision Boundary Calculation is shown in Figure 2. The following code snippet demonstrates the implementation of a decision boundary:

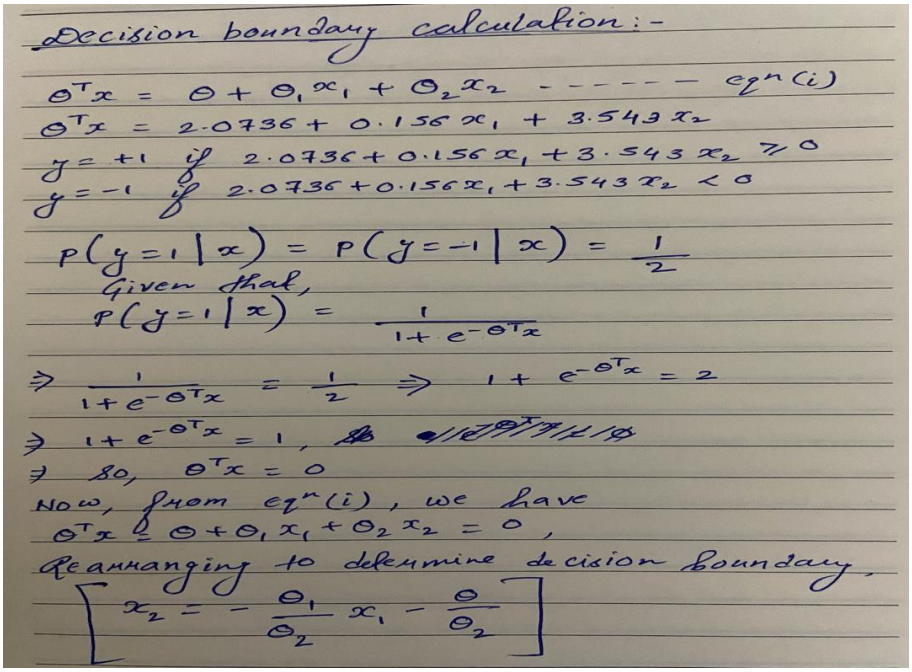

Figure 2 : Decision Boundary Calculation

```
# Decision boundary calculation
decision_boundary = (-coef[0, 0] / coef[0, 1]) * data['X1'] - intercept[0] / coef[0, 1]
```

Next, we proceed to visualize the logistic regression model by employing distinct colors and markers. This helps us easily differentiate between the training data and the model's predictions. Figure 3 displays data visualization of logistic regression model.

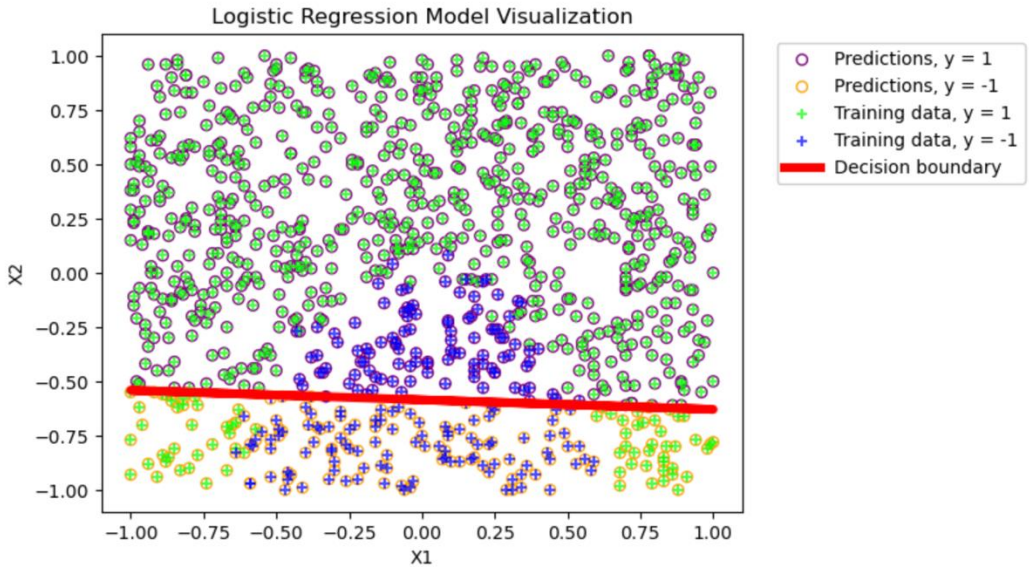

Figure 3 : Logistic Regression Model Visualization model

4. Comment

In this visual representation, we've used unique symbols and colors to effectively illustrate the model's predictions. For class 1 (y =1) predictions, we've used distinct purple circles with clearly defined outlines, while for class -1 predictions, we've opted for orange circles, also with well-defined edges. The training data points are similarly easy to distinguish, with green crosses representing class 1 training data and blue crosses for class -1 training data. When we superimpose these predictions onto the training data, noticeable patterns emerge. Training data points correctly classified as y = 1 are readily identifiable as green markers encircled by purple circles, indicating that they align with the predicted y = 1. However, some training data points within this class, represented by green markers surrounded by orange circles, reveal instances where our model's predictions were less accurate. Similarly, correctly classified training data points for class y = -1 are visible as blue markers encircled by orange circles, signifying predicted y = -1. On the other hand, the misclassified training data points within this class are marked by blue symbols (representing the training data with true class y = -1) and encircled by purple circles. This visualization offers an easy-to-understand way to evaluate how well the model performs in correctly categorizing training data points.

Part B : Support Vector Machine

1. Range of C Values and reporting model parameters

Linear SVM classifiers were employed with a diverse range of penalty parameter C values, encompassing C = 0.001, C = 1, and C = 100. These models were then utilized to generate predictions, and the parameter values associated with each of these trained models are detailed in Figure 4. The **train_and_collect_results** function trains LinearSVC models with various penalty parameter values (C) and stores the model's intercept and coefficients for each C value in a structured format. It takes feature data

X, target data y, and a list of C values to test. The results are returned as a DataFrame called df_svc_results, allowing for easy analysis and comparison of model performance and parameter values.

	C_value	intercept	coef_0	coef_1
0	0.001	0.365515	0.008884	0.332214
1	1.000	0.716742	0.059212	1.262900
2	100.000	0.720880	0.060043	1.272762

Figure 4 : Linear SVM Model Parameters

2. Visualizing LinearSVC Predictions, Actual Targets, and Decision Boundary

Figure 5 shows support vector machine classifications for varying levels of the parameter C. It offers valuable insights into the LinearSVC model's performance across different penalty terms.

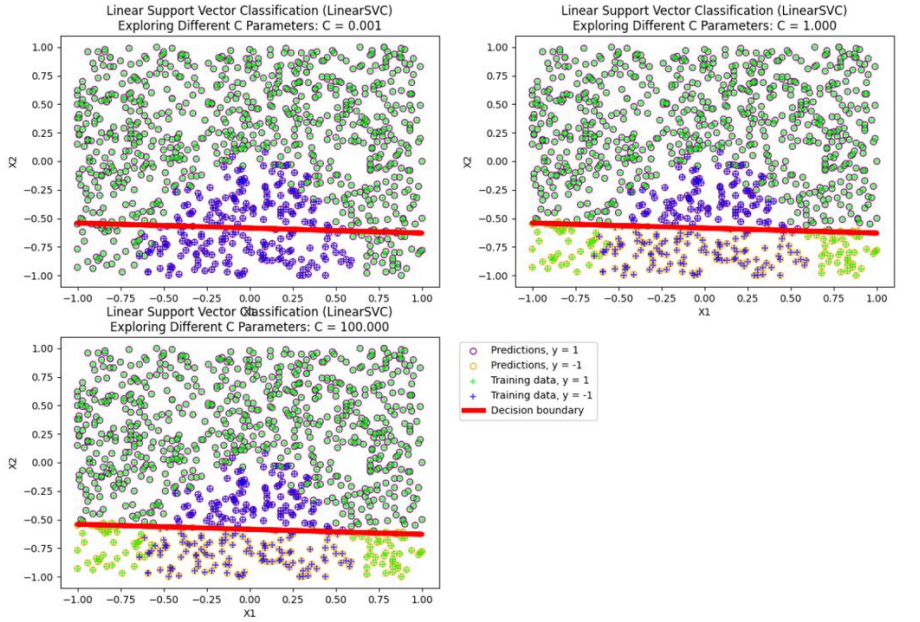

Figure 5 : Linear Support Vector Classification exploring different C Parameters

3. Impact on the model parameters of changing C Parameters , Overall impact on SVM Predictions

Based on the provided model parameters for different values of C in a Linear Support Vector Classification (LinearSVC) model, we can observe the following impacts:

- Intercept: As C increases, the intercept value also increases. This suggests that with higher values of C, the model becomes more biased toward classifying data points as the positive class (y = 1). In other words, it is shifting the decision boundary closer to the positive class.
- Coefficient for Feature X0 (coef_0): Like the intercept, the coefficient for the bias term (coef_0) increases as C increases. This indicates that the model assigns more importance to the bias term in the decision function. It affects the model's ability to capture the offset or bias in the data.
- Coefficient for Feature X1 (coef_1): The coefficient for feature X1 increases significantly as C increases. This means that the feature X1 has a greater influence on the decision boundary with higher values of C. The model becomes more sensitive to variations in X1.

Overall Impact:

When C is small (C = 0.001), the model is more relaxed and allows for a wider margin between classes. It focuses less on individual data points and is willing to tolerate misclassifications.

When C is moderate (C = 1.000), the model strikes a balance between margin width and correct classification. It becomes more sensitive to individual data points but still maintains a reasonable margin.

When C is large (C = 100.000), the model becomes stricter and aims to classify all training points correctly. It pays strong attention to each data point and minimizes the margin, and the model fits the training data closely.

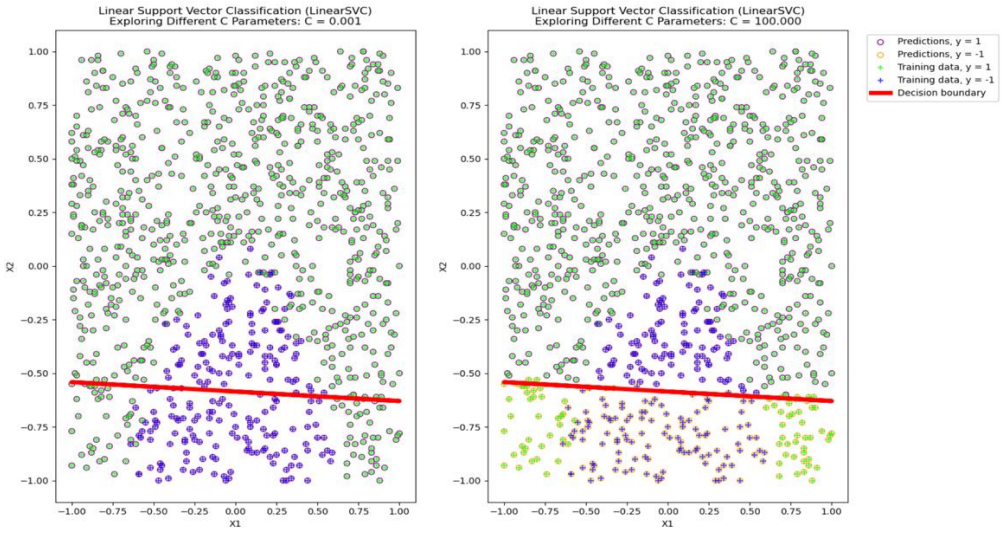

Figure 6: Linear Support Vector Classification for C = 0.001 and C = 100.000

4. Comparing SVM Model Parameters and Predictions to Logistic Regression

Intercept Values: Both Logistic Regression and Linear Support Vector Machine (Linear SVM) models use intercept values. However, they differ significantly in the magnitude of these intercepts. In the case of Logistic Regression, the intercept value is relatively large, specifically 2.07361736. In contrast, the Linear SVM's intercept varies with the

choice of the penalty parameter C. For smaller values of C (e.g., C = 0.001), the intercept is much smaller (0.365514), and it increases as C becomes larger (e.g., C = 1.000 and C = 100.000).

Coefficient Handling: Logistic Regression provides a fixed set of coefficients for the features, which are [0.15637684, 3.54346519]. These coefficients remain the same regardless of any parameter tuning. On the other hand, Linear SVM's coefficients are sensitive to the choice of the penalty parameter C. When C is small (e.g., 0.001), the coefficients for feature X1 are [0.008884, 0.332214]. However, these coefficients increase notably as C becomes larger (e.g., C = 1.000 and C = 100.000), reaching values of [0.059206, 1.262887] and [0.060040, 1.272764], respectively.

These differences highlight that Linear SVM has more flexibility in adjusting the position and slope of the decision boundary through the C parameter. This means that by changing C, you can make the Linear SVM model more or less sensitive to individual data points. A smaller C emphasizes maximizing the margin width between classes, while a larger C focuses on minimizing misclassifications, potentially leading to a narrower margin. Logistic Regression, in contrast, provides a fixed set of coefficients and does not have this level of parameter-driven flexibility. Therefore, the choice of C in Linear SVM directly influences the trade-off between misclassification and margin width in the model.

Comparing Predictions of Linear SVM and Logistic Regression Models:

1. Accuracy Score:

Logistic Regression: 81.98%
Linear SVM: 81.68%

Both models exhibit relatively similar accuracy scores, with the Logistic Regression model slightly outperforming the Linear SVM model by approximately 0.3%. This suggests that, in terms of overall prediction correctness, Logistic Regression demonstrates a slight advantage on the given dataset.

2. Confusion Matrices:

   Logistic Regression Confusion Matrix:

   - True Positives (TP): In this case, there are 704 instances where the model correctly predicted the positive class (y = 1).
   - True Negatives (TN): There are 115 instances where the model correctly predicted the negative class (y = -1).
   - False Positives (FP): The model incorrectly predicted the positive class 106 times when it was actually the negative class.
   - False Negatives (FN): The model incorrectly predicted the negative class 74 times when it was actually the positive class.

   Linear SVM Confusion Matrix:

   - True Positives (TP): In this case, there are 698 instances where the Linear SVM model correctly predicted the positive class (y = 1).

- True Negatives (TN): There are 118 instances where the model correctly predicted the negative class (y = -1).
- False Positives (FP): The model incorrectly predicted the positive class 103 times when it was actually the negative class.
- False Negatives (FN): The model incorrectly predicted the negative class 80 times when it was actually the positive class.

Based on the above metrics, comparing the Logistic Regression and Linear SVM models, Logistic Regression achieves slightly higher overall accuracy with fewer false negatives, while Linear SVM excels in reducing false positives.

Part C : Logistic Regression with additional features

1. Enhancing Logistic Regression with Squared Features and Trained Model Parameters

First of all, we created a duplicate of original dataset, and we added two additional features by squaring each feature. Afterward, we prepared the data for training a logistic regression model by combining the original features with these new squared features to form a complete set of features. Figure 7 shows the data frame with additional features. The model parameters are shown below.

	X1	X2	y	squared_X1	squared_X2
0	-0.35	-0.46	-1	0.1225	0.2116
1	-0.91	0.61	1	0.8281	0.3721
2	0.65	-0.41	1	0.4225	0.1681
3	-0.23	-0.23	-1	0.0529	0.0529
4	0.89	0.46	1	0.7921	0.2116

Figure 7 : DataFrame with additional feature

Model Parameters:

1. Intercept :

   log_reg_model.intercept_

   array([0.56692014])
2. Coefficients:

   log_reg_model.coef_

   array([[0.15628451, 5.47787471, 8.14975097, 0.60306955]])

2. Comparing Predicted vs. Actual Values with Trained Classifier on Original Features

After training the logistic regression classifier, it was used to make predictions for the target values in the training dataset. Figure 8 displays the logistic regression model with additional features. The plot is designed to be easily understood, with labels, a title, and a legend added for clarity, allowing for a straightforward comparison between the predicted and actual outcomes using the original features.

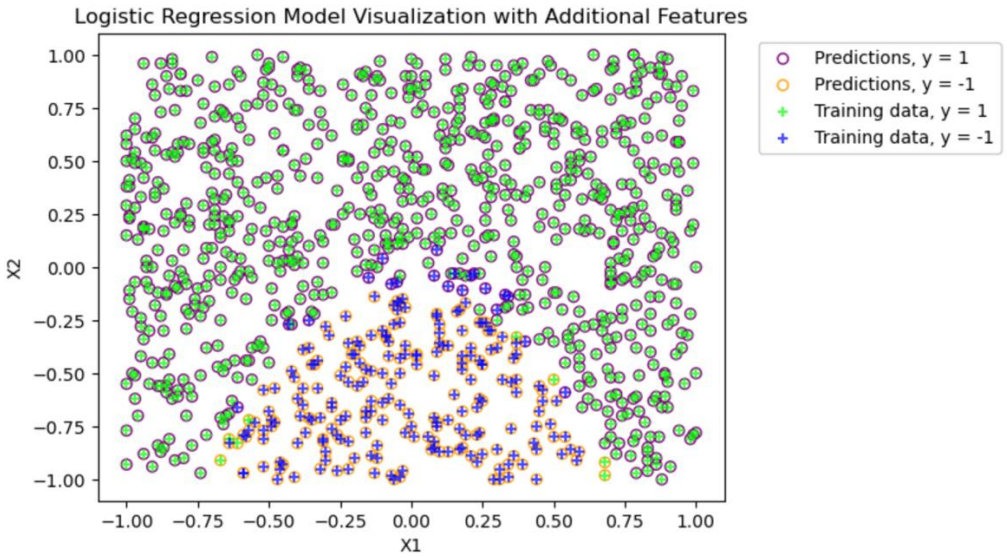

Figure 8 : Logistic Regression Model with Additional Features

Comparing Predictions: Logistic Regression with Two Features vs. Logistic Regression with Additional Squared Features

- Logistic Regression with Two Features (X1, X2):
  1. Accuracy Score: 81.98%
  2. This model was trained using only the original features 'X1' and 'X2.'
  3. It achieved an accuracy score of approximately 82%, indicating its ability to correctly classify data points into their respective classes using a simple linear decision boundary.
- Logistic Regression with Additional Features (Squared X1, Squared X2, X1, X2):
  1. Accuracy Score: 97.20%
  2. This model was trained using the original features 'X1' and 'X2,' as well as their squared counterparts 'Squared X1' and 'Squared X2.'
  3. It achieved a significantly higher accuracy score of approximately 97%, suggesting that the addition of squared features allowed it to capture more complex relationships in the data, resulting in improved predictive performance.

3. Performance Comparison: Classifier vs. Baseline Predictor (Most Common Class)

To start, we looked into how the different classes are distributed in the 'y' column of our DataFrame. This helps us understand whether there's a balance or imbalance in the classes for our classification task. What we found is that there are 778 instances with a '1' label and 221 instances with a '-1' label. This means that '1' is the most common class in our dataset, making it the dominant category. Additionally, we introduced a new column named 'predict_col_1' into the DataFrame 'df' and assigned a constant value of 1 to every row in this column. This step is often used to create a baseline prediction where all instances are predicted to belong to a specific class, in this case, class '1'. Furthermore, we have a function named plot_logistic_regression_baseline_results that helps visualize the results of this baseline prediction. Figure 9 displays the visualization of baseline results.

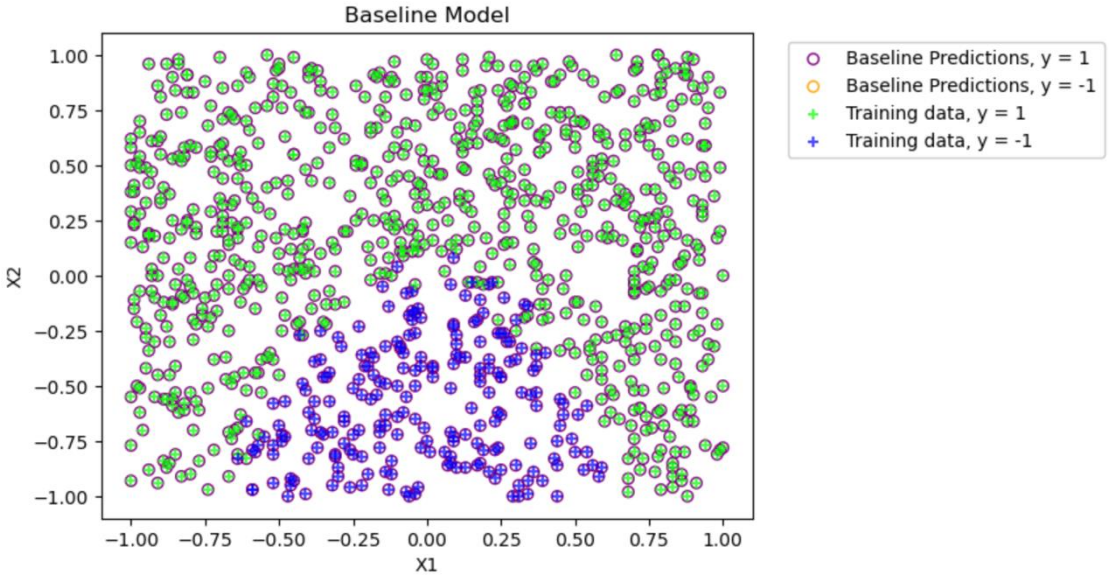

Figure 9: Displays the visualization of baseline model.

4. Comparisons

In the case of the baseline model, there are 778 instances labeled as '1' and 221 instances labeled as '-1.' Consequently, '1' emerges as the most prevalent class within our dataset, establishing it as the dominant category. When comparing the baseline model, as shown in Figure 9, with the logistic regression classifier, it becomes evident that the logistic regression classifier provides a significantly better fit to the data. However, it's worth delving deeper into the specifics of each model's performance.

The baseline model, while seemingly simple, achieves a correctness rate of 77.87%. In practical terms, this means it accurately classifies 77.87% of the observations. Specifically, it correctly predicts 778 instances as belonging to class '1' (y = 1). For a straightforward model like this one, this result is actually quite reasonable and notably surpasses the 75% accuracy threshold, which would be the result of random guessing. On the other hand, the logistic regression classifier outperforms the baseline model significantly, boasting an accuracy score of approximately 97.20%.

This indicates its remarkable ability to correctly classify instances.

Appendix Code

```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, confusion_matrix

#  Part A : Logistic Regression Model
# Part 1: Data Loading and Visualization
# Load the dataset from a CSV file
data = pd.read_csv('week2.csv')
data.columns = ['X1', 'X2', 'y']

# Scatter plot of the data
def plot_scatter(data):
   plt.scatter(data.loc[data['y'] == 1, 'X1'], data.loc[data['y'] == 1, 'X2'], marker='+',
c='lime', alpha=0.5, label='Training data, y = 1')
   plt.scatter(data.loc[data['y'] == -1, 'X1'], data.loc[data['y'] == -1, 'X2'], marker='o',
c='blue', label='Training data, y = -1')
   plt.xlabel('X1')
   plt.ylabel('X2')
   plt.title('Scatter Plot of Features X1 and X2')
   plt.legend(['Training data, y = 1', 'Training data, y = -1'], fancybox=True,
framealpha=1, bbox_to_anchor=(1.04, 1),
            loc="upper left")
   plt.show()

plot_scatter(data)
# Part 2: Logistic Regression Model
# Separate features and target
X = data[['X1', 'X2']]
y = data['y']
# Create a logistic regression model
logistic_model = LogisticRegression()
# Fit the model to the data
logistic_model.fit(X, y)
# Parameter Values
intercept = logistic_model.intercept_
coef = logistic_model.coef_
# Make predictions
predictions = logistic_model.predict(X)
data['predict'] = predictions
```

```
accuracy = accuracy_score(data['y'], data['predict'])*100
# print(accuracy)
confusion_mat = confusion_matrix(data['y'],data['predict'])
# print(confusion_mat)
# Decision boundary calculation
decision_boundary = (-coef[0, 0] / coef[0, 1]) * data['X1'] - intercept[0] / coef[0, 1]
# Visualize the logistic regression model with custom markers and colors
def plot_logistic_regression(data, coef, intercept):
  # Scatter plot of the model's predictions
  plt.scatter(data.loc[data['predict'] == 1, 'X1'], data.loc[data['predict'] == 1, 'X2'],
marker='o',
        facecolors='none', edgecolors='purple', label='Predictions, y = 1')
  plt.scatter(data.loc[data['predict'] == -1, 'X1'], data.loc[data['predict'] == -1, 'X2'],
marker='o',
        facecolors='none', edgecolors='orange', label='Predictions, y = -1')

  # Scatter plot of the training data
  plt.scatter(data.loc[data['y'] == 1, 'X1'], data.loc[data['y'] == 1, 'X2'], marker='+',
c='lime', alpha=0.7,
        label='Training data, y = 1')
  plt.scatter(data.loc[data['y'] == -1, 'X1'], data.loc[data['y'] == -1, 'X2'], marker='+',
c='blue', alpha=0.7,
        label='Training data, y = -1')

  # Plot Decision Boundary
  plt.plot(data['X1'], decision_boundary, linewidth=5, c='red', label='Decision
boundary')
  # Labels, Title, and Legend
  plt.xlabel('X1')
  plt.ylabel('X2')
  plt.title('Logistic Regression Model Visualization')
  plt.legend(fancybox=True, framealpha=1, bbox_to_anchor=(1.04, 1), loc="upper
left")
  plt.show()
# Visualize the logistic regression model
plot_logistic_regression(data, coef, intercept)
# Part B: Support Vector Machine
# Read data from 'week2.csv' and store it in a DataFrame 'df1'
df1 = pd.read_csv('week2.csv')
# Rename the columns of the DataFrame to 'X1', 'X2', and 'y'
df1.columns = ['X1', 'X2', 'y']
# Extract features 'X1' and 'X2' into 'svm_X'
svm_X = data[['X1', 'X2']]
# Extract target 'y' into 'svm_y'
svm_y = data['y']
# Create LinearSVC model
def train_linear_svm(svm_X, svm_y, c_param):
```

```
    svc_model = LinearSVC(C=c_param, dual=True, max_iter=10000)
    svc_model.fit(svm_X, svm_y)
    return svc_model
# Range of C values
c_values_to_test = [0.001, 1, 100]
# Train LinearSVC models for a range of C values and collect results
def train_and_collect_results(svm_X, svm_y, c_values):
    svc_results = []
    for c_param in c_values:
        svc_model = train_linear_svm(svm_X, svm_y, c_param)
        model_results = {
            'C_value': c_param,
            'intercept': svc_model.intercept_[0],
            'coef_0': svc_model.coef_[0, 0],
            'coef_1': svc_model.coef_[0, 1],
        }
        svc_results.append(model_results)
    return pd.DataFrame(svc_results)
# Implement & get dataframe
df_svc_results = train_and_collect_results(svm_X, svm_y, c_values_to_test)
# Part (ii) Plot Data, Predictions & Decision Boundary
# Function to visualize the impact of different C parameters on Linear Support Vector
Classification (LinearSVC)
def visualize_linear_svm_range(data, c_test, plot_dim):
    # Create a figure for plotting
    fig = plt.figure(figsize=(15, 10))
    count = 0
    # Extract features and target labels from the dataset
    X1 = df1['X1']
    X2 = df1['X2']
    svm_X = np.column_stack((X1, X2))
    svm_y = df1['y']
    # Loop through different C parameters and train LinearSVC models
    for c_param in c_test:
        count += 1
        # Train a LinearSVC model with the current C parameter
        svc_model = train_linear_svm(svm_X, svm_y, c_param)
        # Make predictions using the trained model
        predictions = svc_model.predict(svm_X)
        df1['predict'] = predictions
        # Create a subplot for the current C parameter
        plt.subplot(plot_dim[0], plot_dim[1], count)
        # Plot predicted points for class '1' as purple circles
        plt.scatter(df1.loc[df1['predict'] == 1, 'X1'], df1.loc[df1['predict'] == 1, 'X2'],
marker='o',
                    facecolors='none', edgecolors='purple', label='Predictions, y = 1')
        # Plot predicted points for class '-1' as orange circles
```

```
    plt.scatter(df1.loc[df1['predict'] == -1, 'X1'], df1.loc[df1['predict'] == -1, 'X2'], marker='o',
                facecolors='none', edgecolors='orange', label='Predictions, y = -1')
    # Plot actual points for class '1' as green plus signs
    plt.scatter(df1.loc[df1['y'] == 1, 'X1'], df1.loc[df1['y'] == 1, 'X2'], marker='+', c='lime', alpha=0.7,
                label='Training data, y = 1')
    # Plot actual points for class '-1' as blue plus signs
    plt.scatter(df1.loc[df1['y'] == -1, 'X1'], df1.loc[df1['y'] == -1, 'X2'], marker='+', c='blue', alpha=0.7,
                label='Training data, y = -1')
    # Decision boundary
    decision_boundary = (-coef[0, 0] / coef[0, 1]) * X1 - intercept[0] / coef[0, 1]
    plt.plot(df1['X1'], decision_boundary, linewidth=5, c='red', label='Decision boundary')
    # Labels
    plt.title('Linear Support Vector Classification (LinearSVC)\nExploring Different C Parameters: C = %.3f' % c_param)
    plt.xlabel('X1')
    plt.ylabel('X2')
  plt.legend(fancybox=True, framealpha=1, bbox_to_anchor=(1.04, 1), loc="upper left")
  plt.show()
# Range of C values
c_test = [0.001, 1, 100]
# Plot dimensions
plot_dim = [2, 2]
# Implement with a range of C values
visualize_linear_svm_range(data, c_test, plot_dim)
accuracy_for_svm_model = accuracy_score(df1['y'],df1['predict'])*100
# print(accuracy_for_svm_model)
confusion_mat_for_svm_model = confusion_matrix(df1['y'],df1['predict'])
# print(accuracy_for_svm_model)
# Focus on two specific values of C
c_test = [0.001, 100]
# Plot dimensions
plot_dim = [1, 2]
# Implement with specific C values
visualize_linear_svm_range(data, c_test, plot_dim)
#  Part C : Logistic Regression on additional features
# Create a copy of the dataset
df = data.copy()
# Part (i): Data Preprocessing and Logistic Regression Model Training
# Create additional features by squaring X1 and X2
df['squared_X1'] = df['X1'] ** 2
df['squared_X2'] = df['X2'] ** 2
# Define features and target
```

```
squared_X1 = df['squared_X1']
squared_X2 = df['squared_X2']
X_with_squared_features = np.column_stack((X, squared_X1, squared_X2))
y = df['y']
# Train a logistic regression model
log_reg_model = LogisticRegression()
log_reg_model.fit(X_with_squared_features, y)
# Get model coefficients
intercept = log_reg_model.intercept_
coefficients = log_reg_model.coef_
# Part (ii): Predictions and Plotting
# Make predictions using the trained model
predictions = log_reg_model.predict(X_with_squared_features)
# Create a column for predictions in the DataFrame
df['predict'] = predictions
accuracy_for_log_regression_with_additional_features = accuracy_score(df['y'], df['predict']) * 100
# print(accuracy_for_log_regression_with_additional_features)
def plot_logistic_regression_results(df, log_reg_model):
    # Plot of the model's predictions
    plt.scatter(df.loc[df['predict'] == 1, 'X1'], df.loc[df['predict'] == 1, 'X2'], marker='o', facecolors='none', edgecolors='purple', label='Predictions, y = 1')
    plt.scatter(df.loc[df['predict'] == -1, 'X1'], df.loc[df['predict'] == -1, 'X2'], marker='o', facecolors='none', edgecolors='orange', label='Predictions, y = -1')
    # Plot of Training Data
    plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'], marker='+', c='lime', alpha=0.7, label='Training data, y = 1')
    plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'], marker='+', c='blue', alpha=0.7, label='Training data, y = -1')
    # Labels, Title, and Legend
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.title('Logistic Regression Model Visualization with Additional Features')
    plt.legend(fancybox=True, framealpha=1, bbox_to_anchor=(1.04, 1), loc="upper left")
    plt.show()
plot_logistic_regression_results(df,log_reg_model)
# Part (iii): Baseline Model
# Check class distribution of the target variable
class_distribution = df['y'].value_counts()
df['predict_col_1'] = np.ones(len(y))
# Visualizes the results of a baseline model and actual training data using the 'X1' and 'X2' features.
def plot_logistic_regression_baseline_results(df, log_reg_model):
    # Plot Baseline Predictions
    plt.scatter(df.loc[df['predict_col_1'] == 1, 'X1'], df.loc[df['predict_col_1'] == 1, 'X2'], marker='o',facecolors='none', edgecolors='purple', label='Baseline Predictions, y = 1')
```

```
    plt.scatter(df.loc[df['predict_col_1'] == -1, 'X1'], df.loc[df['predict_col_1'] == -1, 'X2'], marker='o', facecolors='none', edgecolors='orange', label='Baseline Predictions, y = -1')
    # Plot of Training Data
    plt.scatter(df.loc[df['y'] == 1, 'X1'], df.loc[df['y'] == 1, 'X2'], marker='+', c='lime', alpha=0.7, label='Training data, y = 1')
    plt.scatter(df.loc[df['y'] == -1, 'X1'], df.loc[df['y'] == -1, 'X2'], marker='+', c='blue', alpha=0.7, label='Training data, y = -1')
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.title('Baseline Model')
    plt.legend(fancybox=True, framealpha=1, bbox_to_anchor=(1.04, 1), loc="upper left")
    plt.show()
plot_logistic_regression_baseline_results(df,log_reg_model)
```