Diksha Srivastava (21235117) and Saumya Goel (21238562) – CSD1

Logistic Regression

Definition

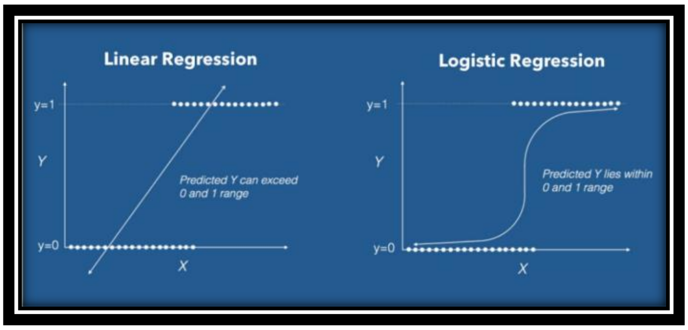
Logistic regression is a supervised classification algorithm used for categorical variables which can take only discrete values as the output such as 0/1, True/False, Yes/No, etc.

Use of logistic regression for classification

Logistic regression creates a regression model by computing the probability of the given observation belonging to a particular category. It becomes a classification algorithm when a decision threshold is set making threshold value an important element which in turn is affected by the values of precision and recall.

Linear Regression vs Logistic Regression

Linear regression and logistic regression are similar in the essence that both are parametric regressions but that is it. Linear Regression computes continuous values for the target/output variable whereas Logistic Regression computes discrete values for the target/output variable. Also, Linear Regression uses a linear function which results in a straight line graph whereas Logistic Regression uses a sigmoid function which results in a curved line graph.



(Graph)

Types of logistic regression

There are 3 types of Logistic regression based on the number of categories which are as follows:

a) Binomial: The target variable has only two possible output values. For eg: Yes/No, 0/1, Win/Loss
b) Multinomial: The target variable has three possible unordered output values. For eg: "Disease A" vs "Disease B" vs "Disease C"
c) Ordinal: The target variable has ordered output values. For eg: "Good", "Better", "Best" or "Excellent" (Types of Logistic Regression)

Which type are we using for our purpose?

We need binomial logistic regression since there are only two classes/categories (Fire: Yes/No)

Core of logistic regression

Logistic regression has three main functions:

a) Hypothesis
b) Cost
c) Gradient Descent

The notation that is going to be used for the algorithm explanation is as follows:

X	input data matrix of shape m x n
m	number of observations
n	number of features

y	output/target value
x(i), y(i)	i^{th} training example
θ	regression coefficient
α	Learning rate

Hypothesis

To start building the hypothesis for the logistic regression algorithm, regression coefficients need to be introduced.

Regression coefficients are associated with the features which tel006C how important a feature is for the classification problem.

The hypothesis for linear regression is:

$$h_\theta(x) = \theta^T X$$

Logistic regression also uses the hypothesis function which is slightly different because the hypothesis for linear regression is a linear function that can have values greater than 1 or less than 0. Therefore, the sigmoid function is used which maps the predicted real values to other values between 0 and 1. In other words, the sigmoid function maps predictions to probabilities.

The formula for **sigmoid function (σ)** is:

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\exp(-z)}$$

For logistic regression, $\theta = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \ldots + \theta_n X_n$

$h_\theta(x) = \sigma(\theta) = \sigma(\theta_0 + \theta_1 X_1 + \theta_2 X_2 + \ldots + \theta_n X_n)$

i.e.

$$\boldsymbol{h_\theta(x) = \frac{1}{1+e^{\theta^T X}}}$$

Cost Function

The objective of the cost function is optimization i.e., evaluate the errors the logistic model is going to make so that we can develop a model with minimum error.

The cost function for Linear regression is:

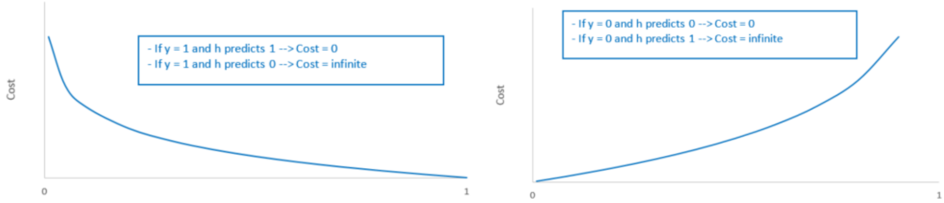
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

However, the cost function for linear regression can't be used for logistic regression as it gives multiple local minimums whereas we need to minimize the cost value and find the global minimum.

For logistic regression, the cost function is as follows:

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

The graph for the cost function is shown below:



For a binary classification problem, the cost function can be simplified as:

$$\boldsymbol{J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1 - y^{(i)})\log\left(1 - h_\theta(x^{(i)})\right)\right]}$$

Gradient Descent

Gradient descent is an optimization technique that finds the optimal coefficients and minimizes the cost function. In this, we iterate through the training set repetitively to find the gradient of the cost function at the current point and move in the opposite direction.

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) * x_j{}^{(i)}$$

Design Decision

a) **Threshold Value:** The selection of threshold value depends on error preferences which are: false positive and false negative. High threshold value results in a more false negative error and a low threshold value results in a more false positive error.
In the wildfires.txt data, we need to predict fire where we can't afford to have more false negative but also, it would be an overhead to have more false positives as it would take up more resources. To reach a balance, we have chosen the threshold value to be 0.5.
$P(Y=1|X) \geq 0.5$, class = Yes, and if P(Y=1|X) < 0.5 $P(Y=1|X) < 0.5$, class = No

b) **Type of gradient descent:** Batch guide gradient descent is used for this data. The reason behind choosing this is as follows:
   - **i)** Coefficients are updated after computing the error gradient for the entire training set.
   - **ii)** It makes the least noisy updates to the coefficients when compared to other gradient descents.
   - **iii)** Since for batch guide gradient descent, we need to loop through the entire data which makes the calculation slow and expensive and takes time to reach convergence but, in our case, there are just 136 observations in the training set which is not huge.

c) **Learning rate:** It determines the magnitude of the amount to move in gradient descent. The change in coefficient is learning rates times the gradient (Martin, 2021). This we need to specify explicitly. If it is too small, the algorithm will converge slowly but if it is large, it can lead to non-convergence i.e., bouncing back and forth between the convex function of gradient descent but never reaching the local minimum. The learning rate used in our code is **0.01**.

d) **Convergence:** Gradient descent minimizes the cost and when the minimum cost is achieved, it means it has converged. In the code, convergence is used to find the minimum cost instead of iterating it a specific number of times. The value of convergence used in our code is **0.01**.

e) **Normalization:** It helps to model the data correctly by creating new values. It maintains the general distribution and ratios in the source data. It keeps values within a scale that is applied to all numeric columns in the model. The normalization used in our code is Z-normalization. It normalizes every value of the dataset in a way that mean of all the values is 0 and the standard deviation is 1.

$$Z = \frac{x - \mu}{\sigma}$$

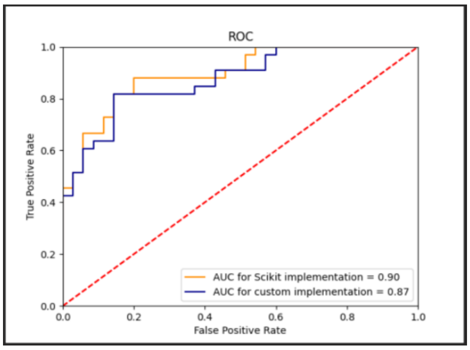
Z = standard score
x = observed value
μ = mean of the sample
σ = standard deviation of the sample

f) Define x_0 = 1in the input matrix as we need to perform matrix product between feature vector and coefficients vector and there was no value to be multiplied with θ_0.

Comparison between Sklearn Implementation and Custom Implementation

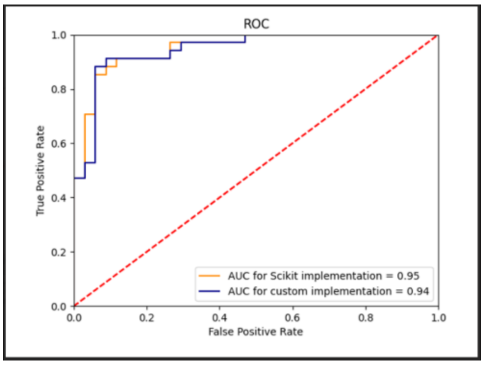
Scikit package has been used as a reference package.
As instructed, the data is split randomly into training data and test data in the ratio 2:1, and the algorithms are executed 10 times.
The difference between the accuracies achieved by the two implementations for different learning rates and convergence is shown in the table below.

accuracy_sk	accuracy_lr
83.82352941	75
83.82352941	83.82352941
76.47058824	79.41176471
94.11764706	82.35294118
79.41176471	79.41176471
77.94117647	70.58823529
83.82352941	72.05882353
89.70588235	88.23529412
79.41176471	76.47058824
82.35294118	72.05882353
Average	
83.08823529	77.94117647



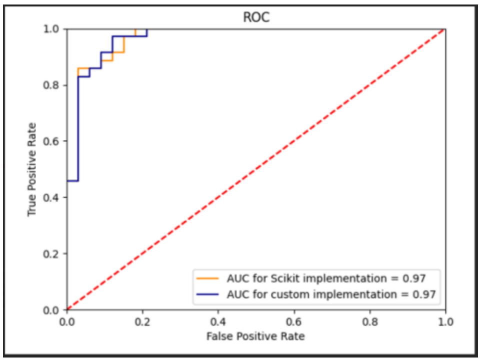
α = 0.01, convergence = 0.01

accuracy_sk	accuracy_lr
88.23529412	88.23529412
88.23529412	88.23529412
82.35294118	82.35294118
79.41176471	79.41176471
86.76470588	86.76470588
86.76470588	80.88235294
85.29411765	80.88235294
85.29411765	76.47058824
83.82352941	83.82352941
88.23529412	89.70588235
Average	
85.44117647	83.67647059



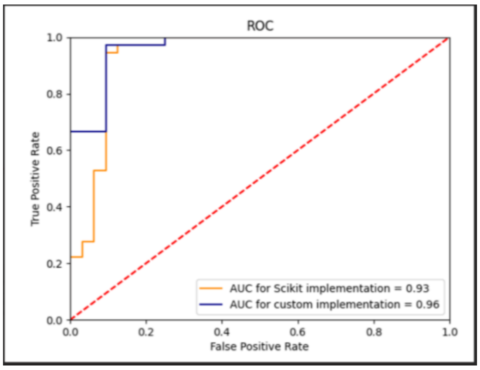
α = 0.01, convergence = 0.001

accuracy_sk	accuracy_lr
82.35294118	89.70588235
85.29411765	79.41176471
83.82352941	89.70588235
82.35294118	82.35294118
89.70588235	80.88235294
82.35294118	86.76470588
86.76470588	88.23529412
85.29411765	80.88235294
77.94117647	88.23529412
91.17647059	91.17647059
Average	
84.70588235	85.73529412



α = 0.1, convergence = 0.001

accuracy_sk	accuracy_lr
91.17647059	85.29411765
82.35294118	63.23529412
83.82352941	86.76470588
91.17647059	88.23529412
88.23529412	86.76470588
88.23529412	82.35294118
88.23529412	89.70588235
85.29411765	85.29411765
91.17647059	85.29411765
91.17647059	85.29411765
Average	
88.08823529	83.82352941



α = 0.1, convergence = 0.00001

Note: The excel sheets containing the detailed output for the four test cases shown above are available in the .zip folder which can be used for reference.

Observations

- **a)** The accuracy achieved with custom implementation is quite close to the accuracy achieved with the sklearn implementation.
- **b)** The difference between accuracies stems from the reason that sklearn implementation uses regularization which prevents the algorithm from overfitting the training dataset. The regularization is not used in the custom implementation because we have a small dataset that is being split into training and test data and the chances of overfitting are less.
- **c)** Sklearn implementation uses a number of iterations = 100, as well as convergence (it is called tolerance in sklearn implementation), and the default value for this is 0.0001. As per our understanding, the algorithm runs to find the minimum cost but if it is not achieved in 100 iterations, it considers the cost computed at the 100^{th} execution as the minimum cost.
- **d)** With different values of learning rate and convergence, we achieved different accuracies but the difference is not huge.
- **e)** Decreasing the value of convergence results in more time being taken to find the minimum cost.
- **f)** Accuracy can be used as an analysis tool in our model since the data is not unbalanced.

Conclusion

Logistic regression works well with linearly separable data. It is easier to implement, interpret, and efficient to train. Logistic regression assigns weights to the features of the data which is used to determine the impact of the feature along with the direction of association (positive or negative) and can be used to eliminate features that are less important in the case of a very large dataset. This makes logistic regression a suitable classification algorithm.

Division of Work

Section	Task	Contributor
Code	sigmoid, cost, predict and normalize	Saumya Goel
Code	gradient_descent, logistic_gradient, fit, accuracy	Diksha Srivastava
Report	Introduction to logistic regression (definition, use of logistic regression for classification, Linear vs Logistic Regression, Types of Logistic Regression, which type are we using for our purpose?) Core of logistic regression (Hypothesis)	Diksha Srivastava
Report	Core of logistic regression (Cost and gradient descent), design decisions (Threshold value, type of gradient descent, learning rate, convergence and normalization)	Saumya Goel
Report	Comparison, Observation and Conclusion	Worked together

References

Graph, L. v. (n.d.). *Introduction to Logistic Regression*. Retrieved from towardsdatascience: towardsdatascience.com/introduction-to-logistic-regression-66248243c148

Martin, D. J. (2021, September 21). *Logistic Regression*. Retrieved from Stanford: www.web.stanford.edu/~jurafsky/slp3/5.pdf

Understanding Logistic Regression. (n.d.). Retrieved from Geeksfor geeks: www.geeksforgeeks.org/understanding-logistic-regression/

Notes Logistic Regression - Standford ML Andrew Ng (joparga3.github.io)

APPENDIX

```python
# Custom Implementation
import numpy as np

"""
Assignment 2
Name: Diksha Srivastava (21235117)
      Saumya Goel (21238562)
Course: Msc Data Analytics (CSD1)
"""
class Logistic:
    def normalize(self, X):
        """
        This method applies Z normalization over the input array X and returns the normalized array.
        @author: Saumya Goel
        :param X: input training observations
        :return: normalized training observations
        """
        n = X.shape[1]
        for i in range(n):
            X = (X - X.mean(axis=0))/X.std(axis=0)
        return X

    def sigmoid(self, theta, X):
        """
        This method returns the hypothesis of logistic regression which is the probability
        ranging between 0 and 1. This probability can be further used for classification
        task based on threshold value.
        @author: Saumya Goel
        :param theta: regression coefficient
        :param X: input training observations
        :return: hypothesis of logistic regression i.e probabilities ranging between 0 and 1
        """
        return 1.0 / (1 + np.exp(-np.dot(X, theta.T)))

    def cost(self, theta, X, y):
        """
        This method calculates the cost i.e. evaluate the errors a logistic model is going to make.
        @author: Saumya Goel
        :param theta: regression coefficient
        :param X: input training observations
        :param y: labels/target value
        :return: cost value
        """
        y_hat = self.sigmoid(theta, X)
        y = np.squeeze(y)
        prob_y_1 = y * np.log(y_hat)
        prob_y_0 = (1 - y) * np.log(1 - y_hat)
        cost = -prob_y_1 - prob_y_0
        return np.mean(cost)

    def logistic_gradient(self, theta, X, y):
        """
        This method calculates the gradient using formula (h(x) - y) . X
        @author: Diksha Srivastava
        :param theta: regression coefficient
        :param X: input training observations
        :param y: labels/target value
        :return: the gradient
        """
```

```
        first_gradient = self.sigmoid(theta, X) - y.reshape(X.shape[0], -1)
        final_gradient = np.dot(first_gradient.T, X)
        return final_gradient

    def gradient_descent(self, X, y, theta, learning_rate, converge):
        """
        This method is used to find the optimal coefficient i.e., by minimizing the
        cost function.
        @author: Diksha Srivastava
        :param X: input training observations
        :param y: labels/target value
        :param theta: regression coefficient
        :param learning_rate: magnitude of the amount to move in gradient descent
        :param converge: the rate through which convergence can be achieved
        :return: optimized regression coefficient
        """
        cost = self.cost(theta, X, y)
        change_in_cost = 1
        m = X.shape[0]
        while change_in_cost > converge:
            prev_cost = cost
            derivative = self.logistic_gradient(theta, X, y)/m
            theta = theta - (learning_rate * derivative)
            cost = self.cost(theta, X, y)
            change_in_cost = prev_cost - cost
        return theta

    def fit(self, X, y):
        """
        This method calculates the theta coefficient at 0.01 learning rate and
        0.01 convergence.
        @author: Diksha Srivastava
        :param X: input training observations
        :param y: labels/target value
        :return: regression coefficient
        """
        # stacking columns with 1's in feature matrix
        X = np.hstack((np.matrix(np.ones(X.shape[0])).T, X))
        # initializing theta values
        theta = np.matrix(np.zeros(X.shape[1]))
        theta = self.gradient_descent(X, y, theta, 0.01, 0.01)
        return theta

    def predict(self, theta, X):
        """
        This methods makes prediction on the trained model and return the predicted
        value.
        @author: Saumya Goel
        :param theta: regression coefficient
        :param X: input training observations
        :return: predictions and predicted probabilities
        """
        # stacking columns with 1's in matrix
        X = np.hstack((np.matrix(np.ones(X.shape[0])).T, X))
        y_hat = self.sigmoid(theta, X)
        predictions = np.where(y_hat >= 0.5, 1, 0)
        return y_hat, np.squeeze(predictions)

    def accuracy(self, y_test, y_pred):
        """
        This method counts the number of correct predictions and calculates the
        accuracy score.
        @author: Diksha Srivastava
        :param y_test: actual label
        :param y_pred: predicted label
        :return: accuracy
        """
```

```
        # counter
        correctly_classified = 0
        for count in range(np.size(y_pred)):
            if y_test[count] == y_pred[count]:
                correctly_classified = correctly_classified + 1
        return (correctly_classified / len(y_test)) * 100

# Sklearn Implementation
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

"""
Assignment 2
Name: Diksha Srivastava (21235117)
      Saumya Goel (21238562)
Course: Msc Data Analytics (CSD1)
"""
class LogisticSklearn:
    def normalize(self, X_train, X_test):
        """
        This method performs standard scaling and returns the normalized value.
        @author: Diksha Srivastava
        :param X_train: input training observations
        :param X_test: input test observations
        :return: normalized training and test set
        """
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)
        return X_train, X_test

    def train_and_predict(self, X_train, X_test, y_train, y_test):
        """
        This method trains the classifier with X_train and y_train and predicts on the
        y_test value.
        @author: Saumya Goel
        :param X_train: input training observations
        :param X_test: input test observations
        :param y_train: input training labels
        :param y_test: test labels
        :return: accuracy and predicted labels/probabilities
        """
        # logistic regression
        classifier = LogisticRegression(tol=0.01)
        classifier.fit(X_train, y_train.ravel())
        y_pred = classifier.predict(X_test)
        probs = classifier.predict_proba(X_test)
        prob_sk = probs[:, 1]
        accuracy = accuracy_score(y_test, y_pred) * 100
        return accuracy, y_pred, prob_sk


# Calling Class
import pandas as pd
from sklearn.model_selection import train_test_split
from Logistic import Logistic
from LogisticSklearn import LogisticSklearn
import sklearn.metrics as metrics
import matplotlib.pyplot as plt
import warnings

"""
Assignment 2
Name: Diksha Srivastava (21235117)
      Saumya Goel (21238562)
Course: Msc Data Analytics (CSD1)
```

```python
"""
class LogisticRegressionTest:
    warnings.simplefilter(action="ignore", category=UserWarning)

    def read_data(self, file):
        """
        This method reads wildfires.csv file and creates independent variable X
        containing features and dependent variable y containing target value.
        @author: Diksha Srivastava
        :param file: input wildfires.csv file
        :return: X, y which is an array of features and labels
        """
        # Data pre-processing.
        data = pd.read_csv(file)
        # Converting dependent variable 'fire' in binary form where 'yes' is 1 and 'no' is 0.
        data['fire'] = data['fire'].str.strip().map({'yes': 1, 'no': 0})
        X = data.iloc[:, 1:].values
        y = data.iloc[:, :1].values
        return X, y

    def save_predictions(self, writer, y_test, y_pred_sk, y_pred_lr, i):
        """
        This method writes the actual and predicted label achieved from sklearn and custom model
        to Excel sheet for 10 iterations.
        @author: Diksha Srivastava
        :param writer: it is the excel writer object of pandas
        :param y_test: test set of target values
        :param y_pred_sk: prediction array of sklearn model
        :param y_pred_lr: prediction array of custom model
        :param i: the number of iterations
        """
        y_test_flat = y_test.ravel()
        df = pd.DataFrame(columns=['y_test', 'y_pred_sk', 'y_pred_lr'])
        df['y_test'] = pd.Series(y_test_flat)
        df['y_pred_sk'] = pd.Series(y_pred_sk)
        df['y_pred_lr'] = pd.Series(y_pred_lr)
        df = df.replace({1: 'yes', 0: 'no'})
        df.to_excel(writer, sheet_name='Exec ' + str(i+1), index=False)

    def plot_roc(self, y_test, preds, color, label):
        """
        This method plots the ROC curve for the given model.
        @author: Saumya Goel
        :param y_test: test set of target values
        :param preds: predicted probability of y_test values
        :param color: provides different color to each model
        :param label: provides different label to each model
        """
        fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
        roc_auc = metrics.auc(fpr, tpr)
        plt.title('ROC')
        plt.plot(fpr, tpr, 'b', label=label+' = %0.2f' % roc_auc, color=color)
        plt.legend(loc='lower right')
        plt.plot([0, 1], [0, 1], 'r--')
        plt.xlim([0, 1])
        plt.ylim([0, 1])
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')

    def main(self):
        """
        This is the main method which creates the object of sklearn and custom implementation
        and drives the whole program.
        @author: Diksha Srivastava
```

```
        @author: Saumya Goel
        """
        global X_test_sk, y_test, y_hat, prob_sk
        # A dataframe to store the accuracies over 10 iterations and the average accuracy as well.
        df = pd.DataFrame(columns=['accuracy_sk', 'accuracy_lr'])
        writer = pd.ExcelWriter('Output.xlsx', engine='xlsxwriter')
        for i in range(0, 10):
            # Reading and splitting data
            X, y = self.read_data('wildfires.csv')
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1 / 3)

            # Training model through sklearn logistic regression
            sklearn_model = LogisticSklearn()
            X_train_sk, X_test_sk = sklearn_model.normalize(X_train, X_test)
            accuracy_sklearn, y_pred_sk, prob_sk = sklearn_model.train_and_predict(X_train_sk, X_test_sk, y_train,
y_test)

            # Training model through custom logistic regression
            lr_model = Logistic()
            X_train_lr = lr_model.normalize(X_train)
            X_test_lr = lr_model.normalize(X_test)
            theta = lr_model.fit(X_train_lr, y_train)
            y_hat, y_pred_lr = lr_model.predict(theta, X_test_lr)
            accuracy_lr = lr_model.accuracy(y_test, y_pred_lr)
            self.save_predictions(writer, y_test, y_pred_sk, y_pred_lr, i)

            # Storing accuracy in dataframe
            df.loc[i] = [accuracy_sklearn, accuracy_lr]

        print(df)
        print('Average Accuracy of sklearn model: ', df['accuracy_sk'].mean())
        print('Average Accuracy of custom model: ', df['accuracy_lr'].mean())

        # Writing the average accuracy to excel.
        df.to_excel(writer, sheet_name='Accuracy', index=False)
        writer.sheets['Accuracy'].activate()
        writer.sheets['Accuracy'].write(11, 0, 'Average')
        writer.sheets['Accuracy'].write(12, 0, df['accuracy_sk'].mean())
        writer.sheets['Accuracy'].write(12, 1, df['accuracy_lr'].mean())
        writer.save()

        # Plotting ROC curve
        self.plot_roc(y_test, prob_sk, 'darkorange', 'AUC for Scikit implementation')
        self.plot_roc(y_test, y_hat, 'darkblue', 'AUC for custom implementation')
        plt.show()


if __name__ == "__main__":
    main_obj = LogisticRegressionTest()
    main_obj.main()
```