
# Green University Of Bangladesh

**Department Of Computer Science and Engineering (CSE)**

Faculty of Sciences and Engineering

Semester: (Fall, Year: 2023), B.Sc. in CSE (DAY)

## LAB REPORT NO - 07

**Course Title:** Data Mining Lab

**Course Code:** CSE-436 **Section:** D2

**Lab Experiment Name:** Linear and Logistic Regression Implementation Using Python

## Student Details

| | Name | ID |
|---|---|---|
| 1 | Shamim Ahmed | 201902067 |

**Lab Date** : 3rd November 2023

**Submission Date** : 1st December 2023

**Course Teacher Name** : Rezwanul Haque

## Lab Report Status

**Mark:.....** **Signature:.....**

**Comments:.....** **Date:.....**

# 1 INTRODUCTION

In this lab report we are going to develop a Colab using Machine Learning Concept: Linear and Logistic Regression. We will Choose an appropriate dataset Then we will implement the Linear and Logistic Regression and examined the result visualize with confusion metrics.

# 2 OBJECTIVE

The aim of this lab is to learn about Machine Learning Concept: Linear and Logistic Regression and to evaluate its. Here we will have a clear idea about the Linear and Logistic Regression and we are going to compare this two with a dataset and evalute the model with evalate matric

# 3 THEORY

## 3.1 Linear Regression

Linear Regression is a machine learning algorithm used for predicting continuous outcomes based on the linear relationship between dependent and independent variables. It minimizes the difference between predicted and actual values during training. The model's predictions are evaluated using metrics like Mean Squared Error and R-squared Score. It is widely used due to its simplicity, but its effectiveness relies on meeting assumptions like linearity and independence of residuals.

## 3.2 Logistic Regression

Logistic Regression is a supervised machine learning algorithm primarily used for binary classification tasks, although it can be extended to handle multiclass classification. Unlike linear regression, which predicts continuous outcomes, logistic regression predicts the probability that an instance belongs to a particular class. The logistic regression model applies a logistic (sigmoid) function to the linear combination of input features, resulting in a probability value between 0 and 1. A decision boundary is set to classify instances into different classes based on this probability. Logistic Regression is commonly employed in various fields, including healthcare (disease prediction), marketing (customer churn), and natural language processing (sentiment analysis). It is interpretable, easy to implement, and well-suited for problems with two classes

# 4 IMPLEMENTATION

## 4.1 DATASET

For this experiment, I used the "Synthetic Linear Regression" dataset. The dataset consists of 1000 samples. Each sample represents an observation or data point. Each sample has only one feature denoted as 'X'. This means that the dataset is univariate, containing a single predictor variable.

## 4.2 Data Splitting

### 4.2.1 Training Set (80% of the data)

80% of the dataset is used for training the linear regression model. The training set contains pairs of 'X' and corresponding target values 'y'.

### 4.2.2 Testing Set (20% of the data)

20% of the dataset is reserved for evaluating the performance of the trained model. The testing set also contains pairs of 'X' and corresponding target values 'y'.

## 4.3 Linear Regression with 20 % data split

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,
    r2_score
from sklearn.datasets import make_regression

# Generate a larger synthetic dataset for linear regression
X_regression, y_regression = make_regression(n_samples=1000, n_features=1,
     noise=10, random_state=42)
linear_df = pd.DataFrame(data=np.column_stack((X_regression, y_regression)
    ), columns=['X', 'y'])

# Split the dataset into training and testing sets (20% test data)
X_train_linear, X_test_linear, y_train_linear, y_test_linear =
    train_test_split(
     linear_df[['X']], linear_df['y'], test_size=0.2, random_state=42
)

# Linear Regression
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train_linear, y_train_linear)
```

```
21
22 # Predictions
23 y_pred_linear = linear_reg_model.predict(X_test_linear)
24
25 # Evaluate the linear regression model
26 linear_mse = mean_squared_error(y_test_linear, y_pred_linear)
27 linear_mae = mean_absolute_error(y_test_linear, y_pred_linear)
28 linear_r2 = r2_score(y_test_linear, y_pred_linear)
29
30 # Plot the regression line
31 plt.scatter(X_test_linear, y_test_linear, color='black', label='Actual')
32 plt.plot(X_test_linear, y_pred_linear, color='blue', linewidth=3, label='
    Regression Line')
33 plt.xlabel('X')
34 plt.ylabel('y')
35 plt.legend()
36 plt.title('Linear Regression Prediction')
37 plt.show()
38
39 # Print evaluation metrics
40 print(f"Linear Regression MSE: {linear_mse}")
41 print(f"Linear Regression MAE: {linear_mae}")
42 print(f"Linear Regression R-squared (R2): {linear_r2}")
```

Listing 1: Linear Regression with 20 % data split using Python

## 4.4 Logistic Regression with 20 % data split

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.model_selection import train_test_split
6 from sklearn.linear_model import LogisticRegression
7 from sklearn.metrics import confusion_matrix, precision_score,
    recall_score, f1_score, accuracy_score
8 from sklearn.datasets import make_classification
9
10 # Generate a larger synthetic dataset for logistic regression
11 X_classification, y_classification = make_classification(
12     n_samples=1000, n_features=1, n_informative=1, n_redundant=0,
13     n_clusters_per_class=1, n_classes=2, random_state=42
14 )
15 logistic_df = pd.DataFrame(data=np.column_stack((X_classification,
    y_classification)), columns=['X', 'y'])
16
17 # Split the datasets into training and testing sets (20% test data)
18 X_train_logistic, X_test_logistic, y_train_logistic, y_test_logistic =
    train_test_split(
19     logistic_df[['X']], logistic_df['y'], test_size=0.2, random_state=42
```

```
)

# Logistic Regression
logistic_reg_model = LogisticRegression()
logistic_reg_model.fit(X_train_logistic, y_train_logistic)

# Predictions
y_pred_logistic = logistic_reg_model.predict(X_test_logistic)

# Evaluate the logistic regression model
conf_matrix_logistic = confusion_matrix(y_test_logistic, y_pred_logistic)
precision_logistic = precision_score(y_test_logistic, y_pred_logistic)
recall_logistic = recall_score(y_test_logistic, y_pred_logistic)
f1_logistic = f1_score(y_test_logistic, y_pred_logistic)
accuracy_logistic = accuracy_score(y_test_logistic, y_pred_logistic)

# Plot Confusion Matrix for Logistic Regression with Class Names
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_logistic, annot=True, fmt='d', cmap='Blues',
    xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
plt.title('Confusion Matrix - Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Print evaluation metrics
print("\nLogistic Regression Metrics:")
print(f"Precision: {precision_logistic}")
print(f"Recall: {recall_logistic}")
print(f"F1 Score: {f1_logistic}")
print(f"Accuracy: {accuracy_logistic}")
```

Listing 2: Logistic Regression with 20 % data split using Python

## 4.5 Data Splitting

### 4.5.1 Training Set (70% of the data)

70% of the dataset is used for training the linear regression model. The training set contains pairs of 'X' and corresponding target values 'y'.

### 4.5.2 Testing Set (30% of the data)

30% of the dataset is reserved for evaluating the performance of the trained model. The testing set also contains pairs of 'X' and corresponding target values 'y'.

## 4.6 Linear Regression with 30 % data split

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.datasets import make_regression

# Generate a larger synthetic dataset for linear regression
X_regression, y_regression = make_regression(n_samples=1000, n_features=1, noise=10, random_state=42)
linear_df = pd.DataFrame(data=np.column_stack((X_regression, y_regression)), columns=['X', 'y'])

# Split the dataset into training and testing sets (30% test data)
X_train_linear, X_test_linear, y_train_linear, y_test_linear = train_test_split(
    linear_df[['X']], linear_df['y'], test_size=0.3, random_state=42
)

# Linear Regression
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train_linear, y_train_linear)

# Predictions
y_pred_linear = linear_reg_model.predict(X_test_linear)

# Evaluate the linear regression model
linear_mse = mean_squared_error(y_test_linear, y_pred_linear)
linear_mae = mean_absolute_error(y_test_linear, y_pred_linear)
linear_r2 = r2_score(y_test_linear, y_pred_linear)

# Plot the regression line
plt.scatter(X_test_linear, y_test_linear, color='black', label='Actual')
plt.plot(X_test_linear, y_pred_linear, color='blue', linewidth=3, label='Regression Line')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Linear Regression Prediction')
plt.show()

# Print evaluation metrics
print(f"Linear Regression MSE: {linear_mse}")
print(f"Linear Regression MAE: {linear_mae}")
print(f"Linear Regression R-squared (R2): {linear_r2}")
```

Listing 3: Linear Regression with 30 % data split

## 4.7 Logistic Regression with 30 % data split

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import confusion_matrix, precision_score,
    recall_score, f1_score, accuracy_score
from sklearn.datasets import make_regression, make_classification

# Generate a larger synthetic dataset for linear regression
X_regression, y_regression = make_regression(n_samples=1000, n_features=1,
     noise=10, random_state=42)
linear_df = pd.DataFrame(data=np.column_stack((X_regression, y_regression)
    ), columns=['X', 'y'])

# Generate a larger synthetic dataset for logistic regression
X_classification, y_classification = make_classification(
    n_samples=1000, n_features=1, n_informative=1, n_redundant=0,
    n_clusters_per_class=1, n_classes=2, random_state=42
)
logistic_df = pd.DataFrame(data=np.column_stack((X_classification,
    y_classification)), columns=['X', 'y'])

# Split the datasets into training and testing sets (30% test data)
X_train_linear, X_test_linear, y_train_linear, y_test_linear =
    train_test_split(
    linear_df[['X']], linear_df['y'], test_size=0.3, random_state=42
)
X_train_logistic, X_test_logistic, y_train_logistic, y_test_logistic =
    train_test_split(
    logistic_df[['X']], logistic_df['y'], test_size=0.3, random_state=42
)

# Linear Regression
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train_linear, y_train_linear)

# Logistic Regression
logistic_reg_model = LogisticRegression()
logistic_reg_model.fit(X_train_logistic, y_train_logistic)

# Predictions
y_pred_linear = linear_reg_model.predict(X_test_linear)
y_pred_logistic = logistic_reg_model.predict(X_test_logistic)

# Evaluate the models
linear_mse = mean_squared_error(y_test_linear, y_pred_linear)

# Confusion Matrix, Precision, Recall, F1 Score for Logistic Regression
```

```
conf_matrix_logistic = confusion_matrix(y_test_logistic, y_pred_logistic)
precision_logistic = precision_score(y_test_logistic, y_pred_logistic)
recall_logistic = recall_score(y_test_logistic, y_pred_logistic)
f1_logistic = f1_score(y_test_logistic, y_pred_logistic)
accuracy_logistic = accuracy_score(y_test_logistic, y_pred_logistic)

# Plot Confusion Matrix for Logistic Regression with Class Names
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix_logistic, interpolation='nearest', cmap=plt.cm.
    Blues)
plt.title('Confusion Matrix - Logistic Regression')
plt.colorbar()

classes = ['Class 0', 'Class 1']
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)

plt.xlabel('Predicted')
plt.ylabel('Actual')

for i in range(len(classes)):
    for j in range(len(classes)):
        plt.text(j, i, str(conf_matrix_logistic[i, j]),
    horizontalalignment='center', verticalalignment='center')

plt.show()

print(f"Linear Regression MSE: {linear_mse}")
print("\nLogistic Regression Metrics:")
print(f"Precision: {precision_logistic}")
print(f"Recall: {recall_logistic}")
print(f"F1 Score: {f1_logistic}")
print(f"Accuracy: {accuracy_logistic}")
```

Listing 4: Logistic Regression with 30 % data split

# 5 OUTPUT

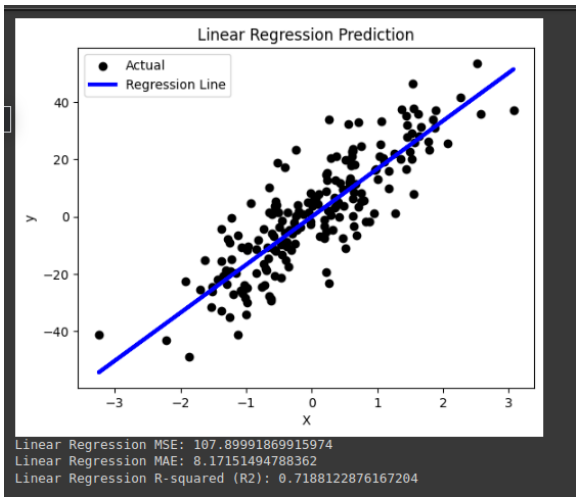


Figure 1: Linear Regression with 20 % data split

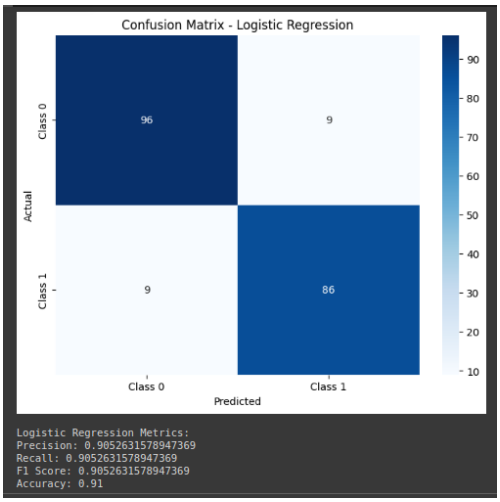


Figure 2: Logistic Regression With 20 % Data Split

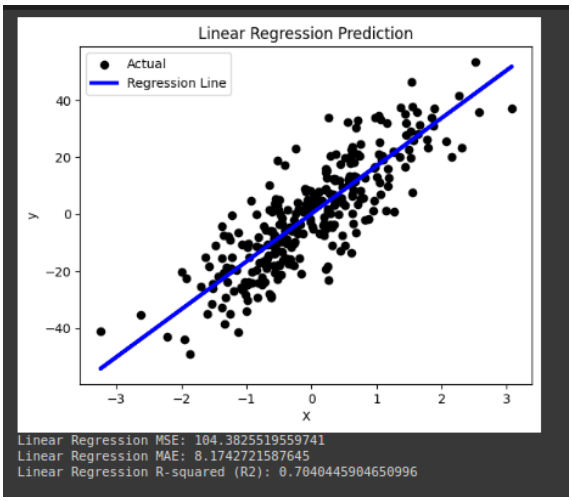


Figure 3: Linear Regression with 30 % data split

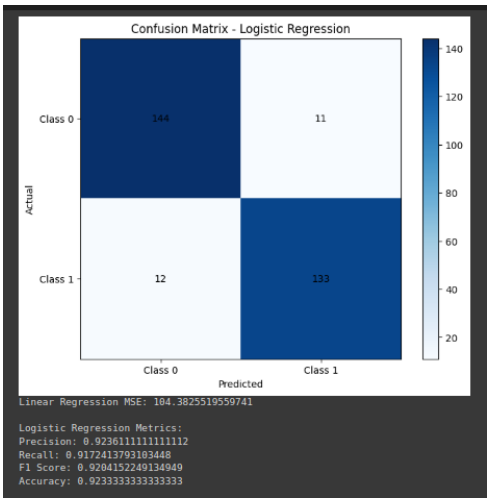


Figure 4: Logistic Regression With 30 % Data Split

# 6 CONCLUSION

In this lab report I implement Linear and Logistic Regression with "Synthetic Linear Regression" dataset. Here we split the training dataset 30% and 20%. Let's observe the mode

| Evaluation | 20% Testing Split | 30% Testing Split |
|---|---|---|
| MSE | 107.89 | 104.38 |
| MAE | 8.17 | 8.17 |
| R-Squared | 0.72 | 0.70 |

Table 1: Linear Regression Performance

| Evaluation | 20% Testing Split | 30% Testing Split |
|---|---|---|
| Precision | 0.91 | 0.923 |
| Recall | 0.91 | 0.917 |
| F1 Score | 0.91 | 0.92 |
| Accuracy | 0.91 | 0.923 |

Table 2: Logistic Regression Performance

Here we can see the impact of using two different techniques . The most difficult part was for me to compare those two logic with different split parcent training and testing data. But with my teacher's help I overcame that problem. Overall it was a great experiment to do.