

# ML\_Lab3\_Logistic\_Regression

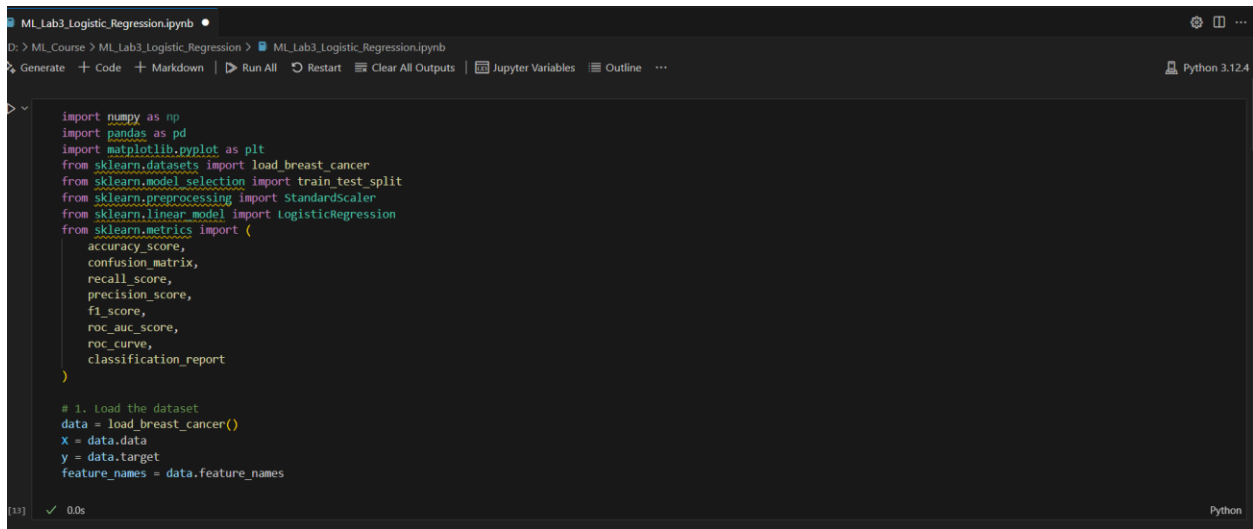
Student Name: Abdul Malik

Student Number: 2211011098

## Objective:

The main objective of this lab is to demonstrate how to preprocess a dataset which is a dataset of Breast Cancer patients' entries, train a logistic regression model, and evaluate it through multiple performance metrics (accuracy, confusion matrix, precision, recall, F1-score, and AUC-ROC), and finally decide whether the cancer is malignant or benign. Ultimately gaining practical insights into the classification capabilities of logistic regression.

## Procedure:



```
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.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    recall_score,
    precision_score,
    f1_score,
    roc_auc_score,
    roc_curve,
    classification_report
)

# 1. Load the dataset
data = load_breast_cancer()
X = data.data
y = data.target
feature_names = data.feature_names
```

The purpose for using different libraries and their packages are:

- **numpy** and **pandas**: Used for handling numerical computations and data structures (arrays, DataFrames).
- **matplotlib.pyplot**: Used for visualizing the results, such as plotting the confusion matrix and ROC curve.
- **sklearn.datasets.load\_breast\_cancer**: Loads the Breast Cancer Wisconsin dataset from scikit-learn's built-in datasets (an API from scikit-learn).
- **sklearn.model\_selection.train\_test\_split**: Splits the dataset into training and testing subsets.

- **sklearn.preprocessing.StandardScaler**: Performs feature scaling, a common preprocessing step for many machine-learning algorithms (including logistic regression).
- **sklearn.linear\_model.LogisticRegression**: Implements logistic regression for classification.
- **sklearn.metrics**: A collection of various metrics for model evaluation, such as accuracy, confusion matrix, recall, precision, F1-score, AUC, etc.

Second cell explanation:

- **data**: The breast cancer dataset is loaded into a Bunch object.
- **X = data.data**: This is a NumPy array of shape (n\_samples, n\_features). Each row is a data instance and each column is a feature (e.g., radius, texture, smoothness, etc.).
- **y = data.target**: This array contains the labels (0 = malignant, 1 = benign) for each data instance.
- **feature\_names = data.feature\_names**: The names of the features.

Variable Name	Role	Type	Description	Units	Missing Values
ID	ID	Categorical			no
Diagnosis	Target	Categorical			no
radius1	Feature	Continuous			no
texture1	Feature	Continuous			no
perimeter1	Feature	Continuous			no
area1	Feature	Continuous			no
smoothness1	Feature	Continuous			no
compactness1	Feature	Continuous			no
concavity1	Feature	Continuous			no
concave_points1	Feature	Continuous			no

This is the screenshot of the dataset of Breast Cancer Wisconsin (Diagnostic) taken from [Breast Cancer Wisconsin \(Diagnostic\) - UCI Machine Learning Repository](#)

```
# 2. Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

# 3. Data preprocessing (Standard Scaling)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Train the logistic regression model
model = LogisticRegression(max_iter=10000, random_state=42)
model.fit(X_train_scaled, y_train)
```

LogisticRegression(max\_iter=10000, random\_state=42)

In the above screenshot first cell:

- **train\_test\_split**: Takes the full dataset (X, y) and splits it into training and testing subsets.
- **test\_size=0.2**: Reserves 20% of the data for testing and uses 80% for training.
- **random\_state=42**: Sets a seed for reproducibility (ensures that the same data shuffle occurs every time we run the code).
- **stratify=y**: Ensures that both training and test sets have the same proportion of each class label as the original dataset (in this case, the ratio of malignant to benign cases is kept consistent).

In the above screenshot second cell:

- **StandardScaler**: Transforms each feature so that it has zero mean and unit variance. Logistic regression performs better when numerical features are on similar scales.
- **fit\_transform** on `X\_train`\*: Calculates the mean and standard deviation of each feature from the *training set* and applies the transformation.
- **transform** on `X\_test`\*: Uses the previously computed means and standard deviations from the training set to scale the test set. This ensures no data leakage from the test set.

In the above screenshot last cell:

- **LogisticRegression**: Initializes a logistic regression model.
- **max\_iter=10000**: Allows more iterations for convergence, especially if the dataset is large or the default iteration limit is too low.
- **random\_state=42**: Again, set for reproducibility in aspects like random weight initialization if using certain solvers.

- **fit(X\_train\_scaled, y\_train)**: Fits (trains) the logistic regression model to the scaled training data.

```

# 5. Make predictions on the test set
y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)[:, 1] # Probability of the positive class

# 6. Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc_roc = roc_auc_score(y_test, y_prob)

print("Accuracy:", accuracy)
print("Recall:", recall)
print("Precision:", precision)
print("F1-Score:", f1)
print("AUC-ROC:", auc_roc)

```

Accuracy: 0.9824561403508771  
Recall: 0.9861111111111112  
Precision: 0.9861111111111112  
F1-Score: 0.9861111111111112  
AUC-ROC: 0.9953703703703703

In the above screenshot first cell working is:

- **model.predict(X\_test\_scaled)**: Uses the trained logistic regression model to predict class labels (0 or 1) on the test set.
- **model.predict\_proba(X\_test\_scaled)**: Gives the *probabilities* of belonging to each class.

We use `[:, 1]` to extract the probabilities of the positive class (class “1” indicating benign or malignant, depending on how scikit-learn encodes them). These probabilities are needed for calculating metrics like AUC (Area Under the Curve).

In the above screenshot second cell’s explanation is:

- **accuracy\_score**: Proportion of correctly classified instances.
- **recall\_score**: The ability of the model to find all the positive samples (a measure of how many actual positives is correctly identified).
- **precision\_score**: Of all the samples that are classified as positive, how many are actually positive.
- **f1\_score**: The harmonic mean of precision and recall. Useful when we want a balance between precision and recall.
- **roc\_auc\_score**: The area under the Receiver Operating Characteristic curve, which measures how well the model distinguishes between classes across different probability thresholds.

```
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", cm)

print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

[19] ✓ 0.0s Python

...

Confusion Matrix:

```
[[41  1]
 [ 1 71]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.98	0.98	42
1	0.99	0.99	0.99	72
accuracy			0.98	114
macro avg	0.98	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114

For the cell shown above:

- **confusion\_matrix**: Shows a table of counts:
- Rows represent the true labels (0 or 1).
- Columns represent the predicted labels (0 or 1).
- **classification\_report**: Displays precision, recall, F1-score, and support (the number of true instances per class) for each class label, along with macro-averaged and weighted metrics.

As we can see from the output we have TP = 41, FP = 1, FN = 1 and TN = 71.

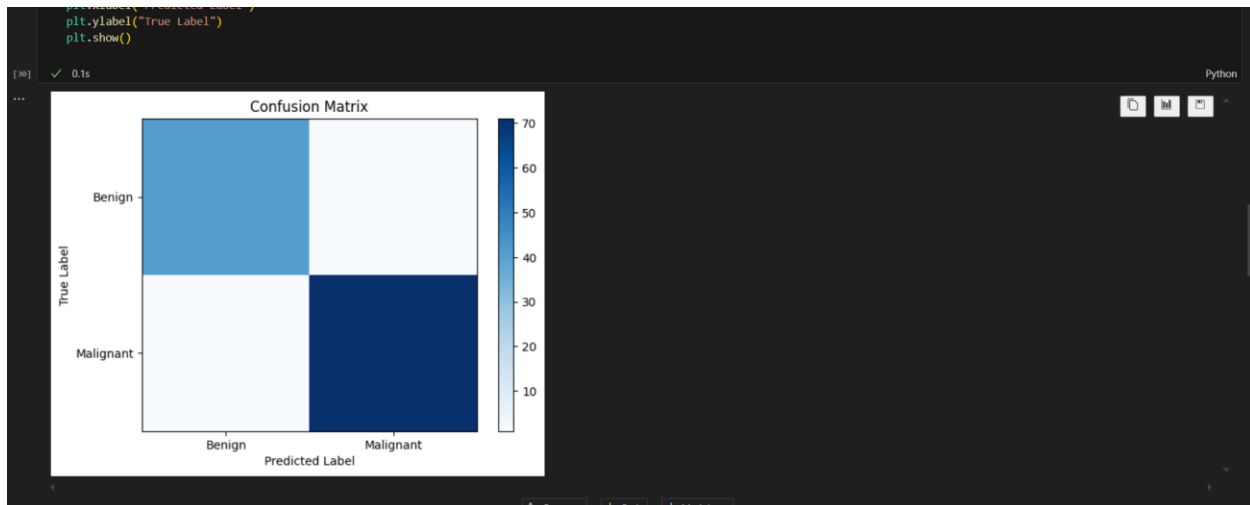
```
plt.figure()
plt.title("Confusion Matrix")
plt.imshow(cm, cmap='Blues', interpolation='nearest', aspect='auto')
plt.xticks([0, 1], ["Benign", "Malignant"]) # 0: Benign, 1: Malignant
plt.yticks([0, 1], ["Benign", "Malignant"]) # 0: Benign, 1: Malignant

plt.colorbar()
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

[30] ✓ 0.1s Python

...

Confusion Matrix

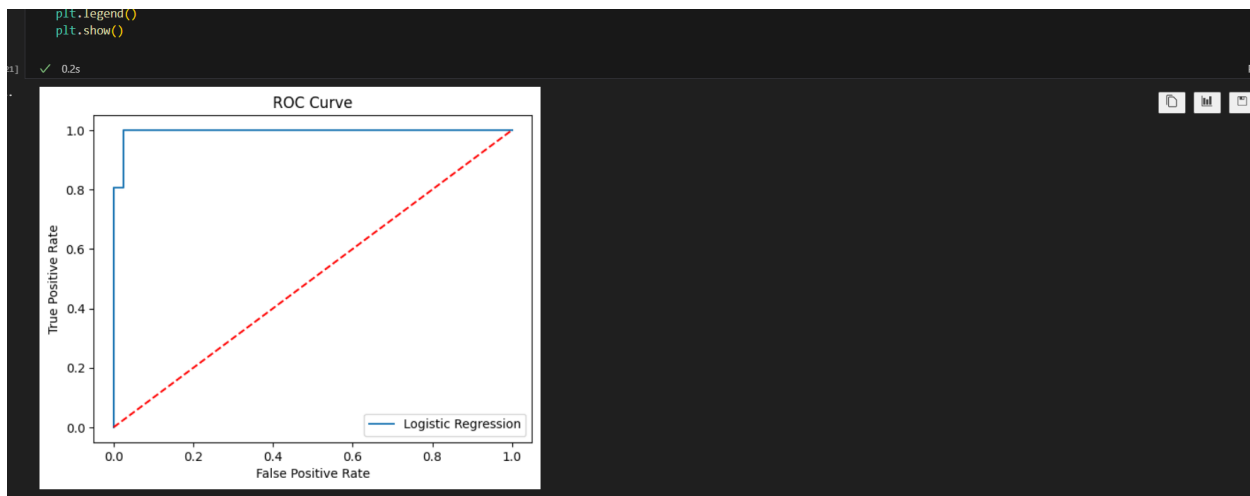


For the cell and plot above we have:

- **plt.figure():** Creates a new figure (a blank canvas) on which the plot will be drawn.
- **plt.title("Confusion Matrix"):** Sets the title at the top of the figure to “Confusion Matrix.”
- **plt.imshow(cm, cmap='Blues', interpolation='nearest', aspect='auto'):**
  - **cm** is our 2D confusion matrix (an array or list of lists).
  - **cmap='Blues'** uses a blue color scale to represent different values in the matrix.
  - **interpolation='nearest'** chooses the nearest-neighbor method for filling pixels between data points (makes each cell a solid square of color).
  - **aspect='auto'** automatically adjusts the aspect ratio so that the cells are more or less square and everything is shown nicely.
- **plt.xticks([0, 1], ["Benign", "Malignant"])** and **plt.yticks([0, 1], ["Benign", "Malignant"]):**
  - These lines label the x-axis (columns) and y-axis (rows) with “**Benign**” and “**Malignant**” instead of numeric 0 and 1.
  - By default, the confusion matrix will have indices 0 and 1 for the two classes; these xticks/yticks just replace those numbers with readable labels.

```
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
plt.figure()
plt.title("ROC Curve")
plt.plot(fpr, tpr, label='Logistic Regression')
plt.plot([0, 1], [0, 1], 'r--') # Random classifier line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

[71] ✓ 0.2s Python



For the cell and graph above we have:

- **roc\_curve**: Computes the False Positive Rate (FPR) and True Positive Rate (TPR) at different probability thresholds.
- **Plotting fpr vs. tpr**: Allows us to see how the model's sensitivity (TPR) trades off against its fall-out (FPR).
- **Diagonal line ([0, 1], [0, 1])**: Represents a random classifier (area under this line is 0.5). A good classifier should be clearly above this line.

```
# Showing the scaled dataset with all columns (features)
train_df = pd.DataFrame(X_train_scaled, columns=feature_names)
test_df = pd.DataFrame(X_test_scaled, columns=feature_names)

print("\nFeature Names:")
print(feature_names)

print("\nScaled Training Data (first 5 rows):")
print(train_df.head())

print("\nScaled Test Data (first 5 rows):")
print(test_df.head())
```

[22] ✓ 0.0s Python

```
ML_Lab3_Logistic_Regression.ipynb X
D: > ML_Course > ML_Lab3_Logistic_Regression > ML_Lab3_Logistic_Regression.ipynb
Generate + Code + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline ...
Python 3.12.4
[22] ✓ 0.0s
...
Feature Names:
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']

Scaled Training Data (first 5 rows):
  mean radius  mean texture  mean perimeter  mean area  mean smoothness \
0   -1.072001   -0.658425   -1.088080   -0.939274   -0.135940
1    1.748743    0.066502    1.751157    1.745559    1.274468
2   -0.974734   -0.931124   -0.997709   -0.867589   -0.613515
3   -0.145103   -1.215186   -0.123013   -0.253192    0.664482
4   -0.771617   -0.081211   -0.803700   -0.732927   -0.672282

  mean compactness  mean concavity  mean concave points  mean symmetry \
0   -1.008718   -0.968359   -1.102032    0.281062
1    0.842288    1.519852    1.994664   -0.293045
2   -1.138154   -1.092292   -1.243358    0.434395
3    0.286762   -0.129729   -0.098605    0.555635
...
3   -0.339891   -0.797251   -1.047159
4    1.614373   -1.314652   -0.695833

[5 rows x 30 columns]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings--
Generate + Code + Markdown
0 0
```

And finally, we create two **pandas DataFrames**:

- **train\_df** corresponds to the scaled training data (X\_train\_scaled)
- **test\_df** corresponds to the scaled test data (X\_test\_scaled).
- We name the columns using the original feature\_names from **data.feature\_names** so we can easily interpret which feature corresponds to each column.
- **print(feature\_names)**: Shows the list of all columns/features used.
- **train\_df.head(), test\_df.head()**: Prints the first 5 rows of each scaled dataset.

## Conclusion:

In this lab, we successfully applied the process of building a binary classification model using logistic regression on the Breast Cancer dataset. We performed data splitting, standard scaling, trained the logistic regression model, and evaluated its predictive power through various metrics including accuracy, recall, precision, F1-score, and AUC-ROC, which together provided a comprehensive understanding of the model's effectiveness. The confusion matrix gave us specific counts of true positives, true negatives, false positives, and false negatives, while the ROC curve illustrated how well the model discriminates between classes at different thresholds. Finally, by visualizing the data and the results, we gained clear insights into how logistic regression performs for this medical classification problem and how well it can generalize to unseen data.