Learning Guide: Calculating Area Under the Curve (AUC) for Logistic Regression in Python

Authored by

Mohammed looti

November 2, 2025

Logistic Regression stands as a cornerstone method in both statistical modeling and **machine learning**, specifically tailored for addressing **binary classification** challenges. It deviates fundamentally from linear regression by outputting the probability of an observation belonging to a particular class, rather than predicting a continuous value. This probabilistic approach is essential for modeling outcomes where the response variable is categorical, such as determining if a customer will default on a loan or not.

The rigorous evaluation of classification models requires moving beyond simplistic metrics like overall accuracy. Accuracy can be highly misleading, particularly when evaluating models trained on **imbalanced datasets** where one class significantly outnumbers the other. To gain a truly granular understanding of a model's predictive capabilities across various operating conditions, we must analyze metrics derived from the confusion matrix, which provides a detailed breakdown of correct and incorrect predictions.

Two pivotal metrics derived from the confusion matrix are utilized to assess model performance across different classification thresholds, offering insights into how well the model handles both positive and negative instances: **Sensitivity** and **Specificity**.

Sensitivity (True Positive Rate or Recall): This metric quantifies the model's success in correctly identifying instances of the positive class. It is calculated as the ratio of true positives to all actual positive cases. Maximizing sensitivity is paramount in scenarios where minimizing **False Negatives** (missed positive cases) is critical, such as in disease screening.
Specificity (True Negative Rate): Specificity assesses the model's ability to correctly identify negative instances. It represents the proportion of true negatives among all actual negative cases. High specificity is crucial when the cost associated with **False Positives** (incorrectly flagging a negative case as positive) is prohibitively high, for example, in fraud detection systems.

While Sensitivity and Specificity offer valuable insights, they are inherently dependent on the specific **classification threshold** chosen (e.g., 0.5). A single threshold provides only a snapshot of performance. To overcome this limitation and acquire a holistic view of the model's discrimination power across its entire operational range, we employ the **Receiver Operating Characteristic (ROC) curve**.

The ROC curve serves as a powerful graphical tool illustrating the trade-off between the true positive rate (Sensitivity) and the false positive rate (1 - Specificity) for a binary classification system. As the discrimination threshold is systematically varied from 0 to 1, the resulting plot maps the diagnostic capability of the classifier. A curve that bows significantly toward the upper-left corner of the plot indicates superior performance, demonstrating high true positive rates achieved with low false positive rates.

Quantifying Performance with Area Under the Curve (AUC)

Although the ROC curve offers rich visual feedback on classifier performance, data scientists often require a single, aggregated metric for easy comparison and model selection. This crucial metric is the **Area Under the Curve (AUC)**, frequently designated as ROC-AUC. The AUC essentially collapses the two-dimensional curve into a single scalar value, summarizing the model's overall discriminatory power across all possible decision thresholds.

Conceptually, the AUC holds a distinct probabilistic interpretation: it represents the likelihood that the classifier will rank a randomly selected positive example higher than a randomly selected negative example. Therefore, the AUC directly measures the degree of separation achieved by the model between the distributions of the positive and negative classes. A higher AUC signifies a greater ability to correctly order instances based on their likelihood of belonging to the positive class.

The score for the AUC metric is constrained to a range between 0 and 1. A score of 1.0 indicates a theoretical perfect classifier, where the model successfully separates all positive and negative instances without error. Conversely, an AUC of 0.5 implies that the model performs no better than random chance, effectively providing no discriminatory advantage. Models with AUC values below 0.5 typically suggest fundamental flaws in the data or model training, or potentially an inverted prediction. Generally, any effective model should strive for an AUC score substantially greater than 0.5.

The subsequent sections of this guide will walk through the practical implementation of calculating this essential metric. We will utilize a standard **Logistic Regression** model and leverage the highly efficient, industry-standard data science libraries available within the Python programming environment, ensuring a reproducible and robust calculation methodology.

Step 1: Importing Essential Python Packages

The initial requirement for any data science workflow in Python is the preparation of the computational environment through the importation of necessary libraries. Our approach relies fundamentally on the established ecosystem of tools designed for high-performance numerical computing and machine learning. Specifically, we integrate the powerful scikit-learn library, which provides robust, easy-to-use implementations of core machine learning algorithms and evaluation metrics.

We must import **pandas** for efficient data manipulation and structuring, which is crucial for handling tabular datasets, and **numpy** for foundational array and numerical operations. From scikit-learn, we specifically pull `train_test_split` for robust data partitioning, the `LogisticRegression` class for modeling, and the `metrics` module, which contains the essential function for calculating the AUC

score.

The following standard Python code block ensures all requisite packages and modules are successfully loaded and prepared for execution:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

Step 2: Preparing Data and Fitting the Logistic Regression Model

With the necessary libraries imported, the next critical stage involves loading the relevant data and structuring it correctly for machine learning consumption. For demonstration purposes, we utilize a standard publicly available dataset focused on predicting credit default risk based on features like student status, account balance, and income. It is essential to explicitly partition the dataset into independent variables (**features**, represented by X) and the dependent variable (the **target response**, y).

Adhering to best practices in model validation, we implement the train-test split methodology. This technique ensures that a designated subset of the data (in this case, 30%) is held back and remains completely unseen during the training process. This reserved test set is paramount because the final AUC score calculated on this data provides an unbiased estimate of the model's performance and its ability to generalize to new, real-world observations. We utilize a `random_state` parameter to guarantee reproducibility of the split across different runs.

Following the data partitioning, we proceed to instantiate the `LogisticRegression` object from scikit-learn and then train, or fit, the model exclusively on the designated training subsets (`X_train` and `y_train`). This training phase allows the algorithm to learn the underlying statistical relationship between the features and the target variable, preparing it for prediction on the test set.

```
# Import dataset from CSV file hosted externally (Github URL)
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/default.csv"
data = pd.read_csv(url)

# Define the predictor variables (features: student status, balance, income) and the response variable (default)
X = data]
y = data
```

Split the dataset into training (70%) and testing (30%) sets, using a random state for reproducibility
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)

# Instantiate the Logistic Regression model
log_regression = LogisticRegression()

# Fit the model using only the training data to prevent data leakage
log_regression.fit(X_train,y_train)

Step 3: Calculating the AUC Score

The core requirement for calculating the AUC metric is not the final hard classification (0 or 1), but rather the continuous probability scores assigned by the model. These scores determine the ranking of instances. We utilize the `predict_proba()` method provided by the scikit-learn model object, applying it to our previously defined test features (`X_test`). This method yields the probability that each observation belongs to each possible class.

The `predict_proba()` output is a two-column array. For binary classification problems, the first column holds the probability of the negative class (0), and the second column (``) holds the crucial probability of the positive class (1). It is this positive class probability vector (`y_pred_proba`) that we pair with the true labels of the test set (`y_test`).

Finally, we pass these two essential components--the true test labels and the predicted probability scores--into the highly optimized `metrics.roc_auc_score()` function. This function efficiently computes the area under the ROC curve, delivering the final, quantitative measure of model performance.

 # Use the fitted model to predict the probability that each observation belongs to the positive class (1)
y_pred_proba = log_regression.predict_proba(X_test)

# Calculate the AUC score using the true test labels and the predicted positive class probabilities
auc = metrics.roc_auc_score(y_test, y_pred_proba)

# Print the resulting AUC score for evaluation
print(auc)

0.5602104030579559

Interpreting the AUC Result

Upon execution of the calculation, we obtain an AUC score of approximately **0.5602**. Proper interpretation of this numerical result is essential for concluding the effectiveness of the deployed model. The AUC score must always be assessed against its established scale, where the baseline performance is fixed at 0.5.

The interpretive spectrum of AUC is clear: a score of **0.5** indicates a complete lack of discriminative ability, meaning the model's predictions are equivalent to a random guess. Generally, scores above 0.7 are considered acceptable for many real-world applications, while scores in the range of 0.8 to 0.9 are often categorized as very good, and scores exceeding 0.9 denote exceptional performance in separating the classes.

Given our specific result of **0.5602**, the model demonstrates only a marginal improvement over random chance. This outcome strongly implies that the selected features--student status, balance, and income--are weak predictors for credit default within the constraints of the standard **Logistic Regression** framework. To achieve meaningful predictive power, a data scientist would need to undertake substantial feature engineering, explore alternative, non-linear classification algorithms, or incorporate external, more informative variables into the model structure.

Conclusion and Additional Resources

The calculation of the **Area Under the Curve (AUC)** is a fundamental and indispensable procedure in the rigorous evaluation of any model dedicated to binary classification tasks. It delivers a powerful, threshold-independent summary metric that accurately reflects the model's overall ranking capability--its effectiveness in assigning higher probability scores to positive instances compared to negative instances. Relying on AUC, rather than accuracy alone, provides a complete picture of model performance, especially in scenarios involving class imbalance.

The methodology demonstrated here follows a standardized, efficient workflow in Python, leveraging the statistical power of the scikit-learn package. The key technical steps include: importing necessary modules like `LogisticRegression` and `metrics`, partitioning the data via the train-test split, and critically, using the `predict_proba()` output as input for the `metrics.roc_auc_score()` function. When interpreting the final score, always anchor the value against the random baseline of 0.5 to determine true predictive utility.

For those interested in deepening their understanding of the theoretical framework that supports this metric, the following resources provide additional, in-depth information concerning the mathematical derivation and practical applications of ROC curves and the associated AUC scores: