Plot a ROC Curve in Python (Step-by-Step)

Authored by

Mohammed looti

November 5, 2025

Logistic Regression is a cornerstone technique in **statistical modeling**, widely employed across machine learning for tackling **binary classification problems**. Unlike its linear counterpart, which aims to predict continuous values, logistic regression calculates the probability that a given observation belongs to a specific category--for instance, predicting whether a financial customer will default on a loan or successfully repay it. Because this model yields probability scores rather than hard classifications, we require highly descriptive metrics to accurately judge its ability to differentiate between the two possible outcomes.

When assessing the overall effectiveness of any classification model, a crucial consideration is the inherent trade-off associated with selecting a specific classification threshold. The model's performance hinges on striking a balance between two fundamentally opposing metrics that define its discrimination capabilities: **Sensitivity** and **Specificity**. Understanding these metrics is essential before diving into visualization techniques.

Sensitivity (True Positive Rate - TPR): This metric measures the proportion of actual positive instances that the model correctly identifies. Achieving high sensitivity is paramount in scenarios where minimizing **false negatives**--missing a positive case--is the highest priority (e.g., medical diagnosis).
Specificity (True Negative Rate - TNR): Conversely, specificity quantifies the proportion of actual negative cases that are correctly identified. High specificity is critical when the primary objective is to minimize **false positives**--incorrectly labeling a negative case as positive (e.g., fraud detection where unnecessary alerts are costly).

A robust evaluation of a classification model should never rely on a single metric. Data scientists invariably seek a comprehensive visualization that illustrates the relationship between these two critical measures--TPR and FPR (False Positive Rate, which is 1 - Specificity)--across every conceivable classification threshold. This visualization is achieved through the creation of the **ROC curve**, an acronym for "Receiver Operating Characteristic" curve. By plotting the True Positive Rate against the False Positive Rate, the ROC curve provides an aggregate view of the model's performance independent of the chosen threshold.

This detailed guide offers a step-by-step methodology for generating, plotting, and interpreting the ROC curve using the leading tools in the Python data science ecosystem. We will leverage core libraries such as Scikit-learn for modeling and metric calculation, and Matplotlib for producing high-quality graphical outputs. Following this tutorial ensures a thorough understanding of this essential model evaluation technique.

Step 1: Preparing the Python Environment and Dependencies

The initial phase of any machine learning project involves setting up the development environment by importing all necessary Python packages. These dependencies are essential as they provide

the underlying functionality required for handling structured data, performing complex numerical computations, fitting the model, calculating performance metrics, and, finally, generating the required visualizations. A failure to import these correctly will halt the entire workflow.

For this specific task--building a logistic regression model and plotting its performance curve--we rely on a standard suite of libraries proven reliable in data science applications. **Pandas** is utilized for efficient data manipulation and loading our sample dataset; **NumPy** provides the backbone for high-level numerical operations; Scikit-learn (often imported as `sklearn`) is indispensable for model construction and calculating the ROC metrics; and **Matplotlib** is the standard library used for plotting and graphical representation.

Execute the following block of Python code to ensure all dependencies are correctly loaded and aliases (like `pd` for Pandas and `plt` for Matplotlib) are established within your environment. This sets the stage for data processing in the subsequent steps.

```
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
import matplotlib.pyplot as plt
```

Step 2: Data Preparation, Splitting, and Model Training

With the environment configured, the next critical phase involves acquiring and structuring the data. We begin by loading a sample dataset, often retrieved directly from a remote source like a GitHub repository, into a Pandas DataFrame. After successful loading, we must clearly define the input features (predictors), denoted as **X**, and the binary outcome variable (target), denoted as **y**. In this example, the predictors include 'student', 'balance', and 'income', and the target is 'default'.

A fundamental practice in responsible machine learning evaluation is partitioning the dataset into distinct training and testing subsets. This split is vital to ensure that the model's performance assessment is truly indicative of its ability to **generalize to unseen data**, preventing overfitting. We will employ a common split ratio: 70% of the data allocated for training the model and the remaining 30% reserved for unbiased testing. The `train_test_split` utility from Scikit-learn simplifies this process, handling the random allocation and ensuring data integrity.

The final action in this step is the instantiation and training of the model itself. We create an instance of the `LogisticRegression` class and then use the `fit()` method, exclusively feeding it the training features (`X_train`) and the training target labels (`y_train`). This fitting procedure

allows the model to learn the mathematical relationships between the predictors and the probability of the positive class outcome.

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

# Define the predictor variables and the response variable
X = data]
y = data

# Split the dataset into training (70%) and testing (30%) sets
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)

# Instantiate the model
log_regression = LogisticRegression()

# Fit the model using the training data
log_regression.fit(X_train,y_train)
```

Step 3: Calculating Metrics and Generating the ROC Curve Plot

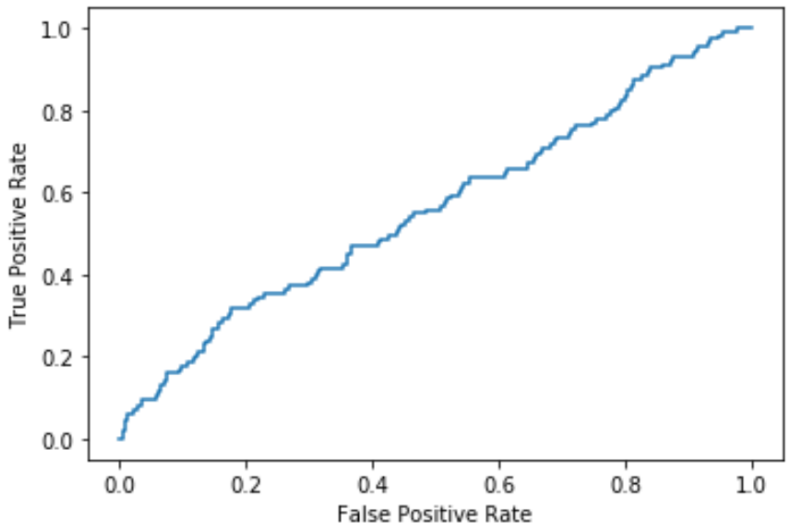
To construct the ROC curve, we must first obtain the predictive probabilities from our trained logistic regression model applied to the test set. Crucially, we utilize the `predict_proba()` method instead of the standard `predict()` method. While `predict()` yields binary classifications (0 or 1), `predict_proba()` returns the raw probability scores for both classes. Since we are interested in the positive class probability (the likelihood of 'default' in this case), we extract the scores from the second column (index 1) of the output array.

These predicted probabilities, along with the true labels (`y_test`), are then passed to Scikit-learn's `roc_curve` function, found within the `metrics` module. This function automatically iterates through every possible classification threshold defined by the probability scores and calculates the corresponding **True Positive Rate (tpr)** and the **False Positive Rate (fpr)**. The output consists of three arrays, but the `fpr` and `tpr` arrays provide the exact coordinates needed for plotting the curve.

The final stage of this step is the visualization itself, achieved using the `matplotlib.pyplot` library. We plot the calculated False Positive Rate (FPR) on the horizontal (x) axis against the True Positive Rate (TPR) on the vertical (y) axis. This graphical representation immediately provides a visual summary of how effectively our model discriminates between positive and negative instances across various thresholds.

```
# Define metrics
y_pred_proba = log_regression.predict_proba(X_test)
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)

# Create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Step 4: Interpreting the Visual Output of the ROC Plot

The interpretation of the generated ROC curve is paramount for understanding the model's quality. The curve fundamentally maps the relationship between the benefits (True Positives) and the costs (False Positives) associated with different decision boundaries. Ideally, a perfect classification model would exhibit maximum performance, reaching the top-left corner of the plot. This mythical point signifies a True Positive Rate (Sensitivity) of 1.0 (100% of positives correctly identified) simultaneously with a False Positive Rate (1 - Specificity) of 0.0 (100% of negatives correctly identified).

In practical terms, the closer the ROC curve tracks the upper-left boundary of the plotting area, the superior the model's discrimination capability is considered to be. Conversely, the diagonal line, often referred to as the line of no-discrimination or the baseline, represents a classifier that performs no better than random chance. Any model whose ROC curve falls near or below this diagonal line is generally deemed ineffective or flawed.

By visually examining the plot produced in the previous step, we can make an initial qualitative assessment. Since the plotted curve appears relatively close to the diagonal baseline, it visually suggests that this specific logistic regression model, utilizing the chosen feature set ('student', 'balance', 'income'), possesses only marginal predictive power. To move beyond this subjective visual assessment and gain a rigorous, quantitative measure of performance, we must proceed to calculate the Area Under the Curve (AUC).

Step 5: Quantifying Performance with Area Under the Curve (AUC)

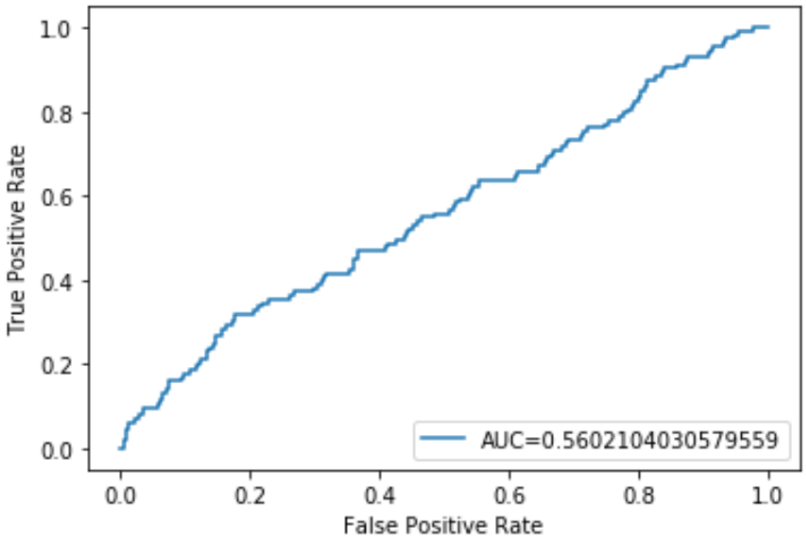
While the visual ROC curve provides an excellent summary of performance across all thresholds, the **AUC**--Area Under the Curve--compresses this complex information into a single, easily interpretable scalar metric. The AUC value can be fundamentally understood as the probability that the classifier will assign a higher score (rank) to a randomly selected positive instance than to a randomly selected negative instance. It is the most common single metric derived from the ROC analysis.

The range of the AUC score is constrained between 0 and 1. A perfect classifier achieves an AUC of 1.0, signifying flawless separation of classes. An AUC of 0.5 indicates performance equivalent to random guessing, which is the performance level of the diagonal baseline. Critically, AUC values falling below 0.5 suggest that the model is performing worse than chance, often pointing to severe issues such as incorrect feature encoding or inverted class labeling during implementation.

To calculate this metric, we employ Scikit-learn's dedicated `roc_auc_score` function, feeding it the true test labels and the predicted probabilities. For enhanced clarity in our visualization, we update the plotting code to incorporate this calculated AUC value directly into the plot's legend. This addition provides immediate, quantitative context for the shape and position of the ROC curve, finalizing our comprehensive evaluation.

```
# Define metrics
y_pred_proba = log_regression.predict_proba(X_test)
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

# Create ROC curve with AUC score in legend
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



The calculated AUC score for this particular analysis using the logistic regression model stands at **0.5602**. This result is only marginally better than the 0.5 baseline, quantitatively confirming the initial visual assessment that the model lacks substantial predictive differentiation. Achieving higher performance would necessitate several strategic improvements, such as performing advanced **feature engineering**, fine-tuning the model using **hyperparameter optimization**, or migrating to a more complex classification algorithm like Support Vector Machines or Gradient Boosting.

Further Reading: For data science scenarios involving model selection, it is often necessary to compare the performance of multiple machine learning models simultaneously. Mastering the technique of visualizing their relative ROC curves is a critical skill for making informed decisions.

Related: How to Plot Multiple ROC Curves in Python