How to Easily Plot a ROC Curve in Python Using Scikit-learn

Authored by

stats writer

December 6, 2025

The scikit-learn library provides the most streamlined approach for plotting a ROC Curve in Python. Specifically, the `roc_curve()` function, housed within `sklearn.metrics`, efficiently calculates the necessary metrics--the True Positive Rate (TPR) and the False Positive Rate (FPR)--across a continuum of threshold values. These computed rates are then utilized to generate the visual representation of the ROC Curve. Beyond visualization, the derived outputs from this process are essential for quantifying the model's performance by calculating the Area Under the Curve (AUC) score.

Evaluating the effectiveness of a binary classification model, such as a Logistic Regression model, requires metrics that capture its predictive power across different operational thresholds. While accuracy offers a simple metric, it often fails to provide a complete picture, especially in datasets where class imbalance exists. For robust evaluation, we turn to metrics derived from the confusion matrix, which quantify the model's ability to discriminate between positive and negative outcomes.

Understanding Binary Classification Metrics

When assessing a classifier, two fundamental metrics derived from the model's predictions are critical for understanding its balance of correct positive and negative predictions. These metrics are inherently linked to the visual representation provided by the ROC Curve.

Sensitivity (True Positive Rate or Recall): This metric represents the probability that the model correctly predicts a positive outcome for an observation when the actual outcome is positive. It measures the proportion of actual positive cases that are correctly identified.
Specificity (True Negative Rate): This metric measures the probability that the model correctly predicts a negative outcome for an observation when the actual outcome is indeed negative. It reflects the model's ability to avoid false alarms.

The ROC curve, which stands for "Receiver Operating Characteristic" curve, is a powerful graphical tool. It plots the Sensitivity (TPR) against the (1 - Specificity), which is the False Positive Rate (FPR), across all possible classification thresholds. This visualization allows data scientists to assess the trade-off between the true positives and false positives generated by the model.

The following detailed, step-by-step example demonstrates the proper procedure for generating and interpreting an ROC Curve in a Python environment utilizing the standard data science toolkit.

Preparation: Importing Necessary Libraries and Tools

Before any statistical modeling or plotting can commence, it is essential to import the requisite libraries. These packages provide the core functionalities needed for data manipulation, statistical modeling, machine learning utilities, and visualization. We rely on industry-standard libraries such

as Pandas for data handling, NumPy for numerical operations, scikit-learn for the classification model and metric calculation, and Matplotlib for generating the final plot.

Ensure that all these packages are installed in your Python environment. The following code snippet demonstrates the required imports, setting the stage for the modeling process that follows.

Step 1: Import Necessary Packages

```
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
```

These imports cover the entire workflow: `pandas` handles the CSV data ingestion, `numpy` supports underlying numerical calculations, the modules from scikit-learn handle the model creation (`LogisticRegression`) and data partitioning (`train_test_split`), and `matplotlib.pyplot` is crucial for the actual graphical representation of the final curve.

Data Handling and Logistic Regression Model Fitting

The next critical phase involves acquiring the dataset, defining the features (predictors) and the target variable (response), and training the classification model. For this demonstration, we utilize a sample dataset related to credit default prediction. A Logistic Regression model is chosen as it natively outputs probability scores, which are necessary for calculating the ROC components.

Before fitting, the data is typically split into training and testing sets. This ensures that the model's performance evaluation, including the generation of the ROC Curve, is based on unseen data, providing a fair assessment of its generalization ability. We allocate 70% of the data for training and reserve 30% for testing.

Step 2: Fit the Logistic Regression Model

We first load the dataset directly from a GitHub repository, define the feature matrix `X` (comprising 'student', 'balance', and 'income') and the target vector `y` ('default'). Subsequently, the data is split, the `LogisticRegression` estimator is instantiated, and finally, it is fitted using the training subset of the data.

```
#import dataset from CSV file on Github
url = "https://raw.githubusercontent.com/arabpsychology/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)
```

Upon successful execution of this step, the `log_regression` object holds the trained model parameters, ready to generate probability predictions on the unseen test set (`X_test`).

Generating Probabilities and Calculating the ROC Curve Components

The calculation of the ROC Curve requires predicting the probability of the positive class for the test data, rather than just the final binary classification (0 or 1). The `predict_proba()` method from scikit-learn provides these probabilities.

Once the probabilities are obtained, the `roc_curve()` function takes the true labels (`y_test`) and these predicted probabilities (`y_pred_proba`) as input. It then iterates through all possible thresholds, calculating the corresponding False Positive Rate (FPR) and True Positive Rate (TPR) pairs. These pairs form the coordinates that define the shape of the ROC curve.

Step 3: Plot the ROC Curve

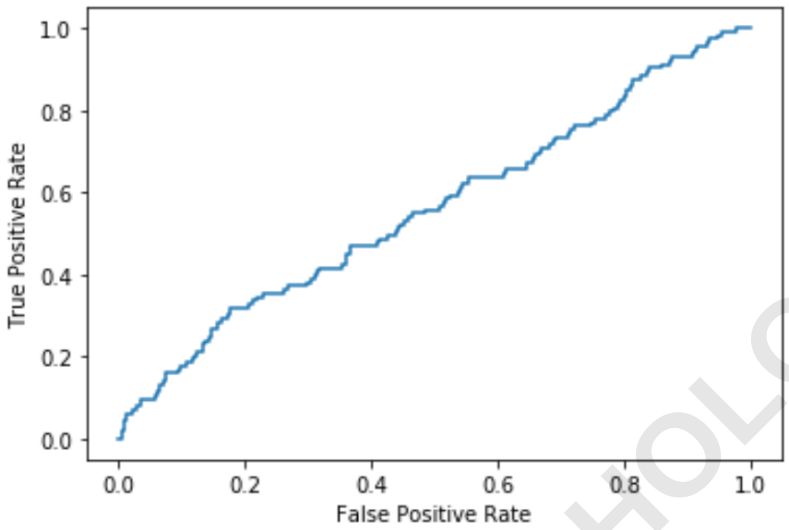
The following code block first calculates the required metrics using the trained model on the test data. It extracts the probabilities for the positive class (index 1) and passes them to `metrics.roc_curve`. Finally, Matplotlib is used to plot the resulting FPR and TPR vectors, generating the initial ROC curve visualization.

```
#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()
```

Execution of this code produces a visualization of the classifier's performance across various discrimination thresholds.



A key visual interpretation of the ROC curve is its proximity to the top-left corner of the plot. A curve that closely "hugs" the upper-left boundary indicates a highly effective classifier, as it achieves a high True Positive Rate while maintaining a low False Positive Rate.

Conversely, a curve that lies close to the diagonal line (representing a random classifier) suggests poor discriminatory power. Observing the plot generated by our current Logistic Regression model, it appears that the curve deviates significantly from the ideal upper-left corner, implying that this specific model configuration performs inadequately at classifying the data categories.

Quantifying Performance: Calculating the Area Under the Curve (AUC) Score

While the visual assessment of the ROC Curve is informative, a single, quantifiable metric is often required for direct comparison between models. This metric is the Area Under the Curve (AUC). The AUC represents the probability that the model will rank a randomly chosen positive instance higher than a randomly chosen negative instance.

The AUC score ranges from 0 to 1. An AUC of 1.0 indicates a perfect classifier, while an AUC of 0.5 suggests that the model is performing no better than random guessing. Calculating the AUC

allows us to numerically quantify the generalized performance illustrated by the curve.

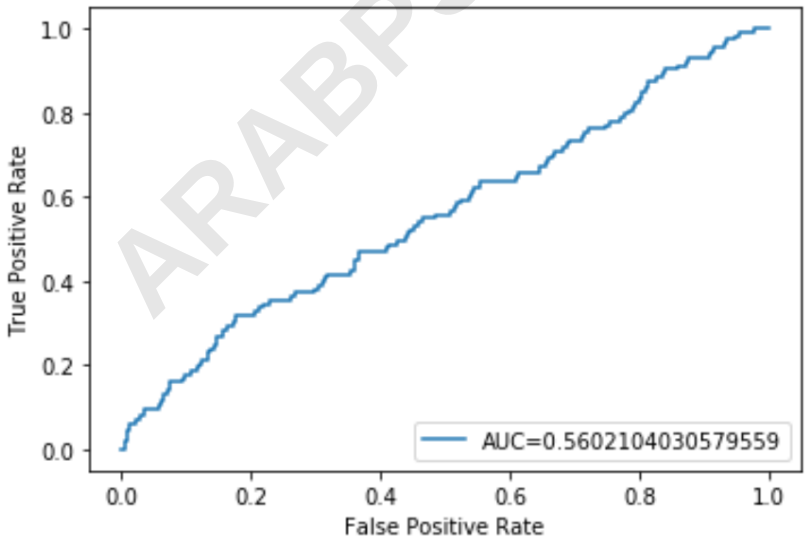
Step 4: Calculate the AUC

To calculate the AUC score, we use the `roc_auc_score()` function, also provided by `sklearn.metrics`. This score can then be integrated into the visualization to provide immediate context regarding the model's quality. The following revised code snippet performs the AUC calculation and updates the Matplotlib plot to display this score within the chart legend.

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

Executing this script generates the final plot, complete with the calculated performance metric in the legend.



Interpreting the AUC Results

The resulting AUC score for this Logistic Regression model is determined to be **0.5602**. Given that an AUC of 0.5 represents a model with no discriminatory power (equivalent to flipping a coin), an AUC of 0.5602 confirms the visual observation that this model performs only marginally better than random chance.

In practice, a strong classifier typically exhibits an AUC score significantly higher than 0.80. Scores near the 0.5 level suggest that the features used (student status, balance, income) are insufficient predictors of the default outcome, or that the linear assumptions inherent in Logistic Regression are not suitable for this specific prediction task. This diagnostic information is invaluable for subsequent steps in the machine learning workflow, guiding efforts toward feature engineering or selecting more complex non-linear models.

For more advanced analysis, it is often necessary to compare the performance of multiple classification models simultaneously. Learn How to Plot Multiple ROC Curves in Python to facilitate comparative model evaluation.