# Plot Multiple ROC Curves in Python (With Example)

Authored by

**Mohammed looti**

October 30, 2025

In the expansive and critical domain of machine learning, the rigorous evaluation of predictive models is non-negotiable, particularly when dealing with classification models. A foundational and universally respected tool for this assessment is the **ROC curve**, which stands for the "receiver operating characteristic" curve. This graphical representation serves to illustrate the diagnostic capability of any binary classifier system across all possible discrimination thresholds, offering a nuanced view of performance that goes beyond simple accuracy metrics.

When data scientists embark on building predictive solutions, it is standard operational procedure to iterate through and test several different classification models on a given dataset. The challenge then becomes selecting the model that offers the most robust and reliable performance. Comparing the respective ROC curves of these candidates provides an intuitive, comprehensive, and standardized visual assessment. This powerful comparison allows analysts to swiftly identify models that strike the optimal balance between correctly identifying positive cases (True Positives) and minimizing incorrect positive identifications (False Positives).

This detailed, practical guide is designed to walk you through the precise process of simultaneously plotting multiple ROC curves using the powerful capabilities of Python. By visualizing the performance of diverse models side-by-side, you will gain the necessary skills to effectively compare their predictive capabilities. Our workflow will cover the entire process, from setting up the required environment and generating reproducible synthetic data to fitting varied models and generating the final comparative visualization.

## Understanding the Mechanics of ROC Curves and AUC

To fully appreciate the utility of the visual comparison, we must first solidify our understanding of the core concepts: the ROC curve itself and the scalar metric known as the Area Under the Curve (AUC). Fundamentally, an ROC curve plots two interdependent performance measures: the True Positive Rate (TPR) on the y-axis against the False Positive Rate (FPR) on the x-axis, as the classification threshold is systematically adjusted. The TPR, also frequently referred to as sensitivity or recall, quantifies the proportion of actual positive cases that the model correctly identified. Conversely, the FPR represents the proportion of actual negative cases that were incorrectly flagged as positive, calculated as 1 minus specificity.

The visual ideal for any ROC curve is to occupy the upper-left quadrant of the plot space. A curve that closely 'hugs' this corner signifies a highly effective classifier, as it indicates that the model is capable of achieving a maximal TPR while simultaneously maintaining a minimal FPR across a wide range of operational thresholds. In contrast, a diagonal line stretching from the bottom-left (0,0) to the top-right (1,1) represents a purely random classifier, which provides no predictive advantage over chance alone.

While the curve offers a rich visual insight, the AUC provides a single, easily digestible scalar value

that summarizes the classifier's overall performance. Conceptually, the AUC can be interpreted as the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. The possible range for the AUC is from 0.5 (random guessing) to 1.0 (perfect discrimination). Therefore, when comparing multiple models, the one whose AUC value is closest to 1.0 is considered the superior choice for the specific classification task.

## Step 1: Setting Up the Python Environment and Dependencies

To commence the practical implementation of plotting multiple ROC curves in Python, the first prerequisite is importing all necessary libraries. These specialized packages are essential as they provide the underlying framework for data manipulation, model training, performance evaluation, and sophisticated data visualization. Each library plays a critical and complementary role in ensuring the workflow executes smoothly and efficiently.

Our implementation relies heavily on three core packages. First, scikit-learn (sklearn) is indispensable, offering a comprehensive suite of tools for machine learning tasks, including standardized model fitting and evaluation metrics. Second, NumPy is required for high-performance numerical operations and array manipulation, which underlies much of the data processing. Finally, Matplotlib is the chosen visualization library, specifically its `pyplot` module, which will handle the crucial task of rendering the comparative ROC curves.

The following code snippet demonstrates the proper initiation and import of all required classes and functions from these libraries, setting the stage for the subsequent data generation and modeling steps:

```
from sklearn import metrics
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
import matplotlib.pyplot as plt
```

## Step 2: Preparing Synthetic Data for Binary Classification

For the purpose of this clear demonstration, we will proceed by generating a synthetic dataset. This approach utilizes scikit-learn's highly convenient make_classification function, which enables us to create a perfectly controlled data environment. By controlling the characteristics of the data, we can focus entirely on illustrating the ROC curve plotting process without needing to grapple with extensive real-world data cleaning or preprocessing complexities. We configure the function to

produce 1,000 observations, utilizing four predictor variables (features), ultimately resulting in a single binary response variable that is suitable for testing binary classification models.

The configuration parameters passed to make_classification--specifically `n_samples`, `n_features`, `n_informative`, and `n_redundant`--allow meticulous control over the complexity and dimensionality of the resulting dataset. Crucially, the inclusion of the `random_state=0` parameter guarantees the reproducibility of our experiment. This means that every time the code is executed, the exact same synthetic data will be generated, which is paramount for consistent comparison and verification of model performance.

Following the data generation, the essential step of splitting the dataset into training and testing sets must be performed. This technique is a cornerstone of robust machine learning practice, serving primarily to mitigate the risk of overfitting--a condition where a model performs excellently on seen data but poorly on novel data. We utilize train_test_split to reserve 30% of the data exclusively for testing, ensuring that our final evaluation of the ROC curves is unbiased and reflects true generalization capability.

**#create fake dataset**
**X, y = datasets.make_classification(n_samples=1000,**
**n_features=4,**
**n_informative=3,**
**n_redundant=1,**
**random_state=0)**

#split dataset into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3,random_state=0)

## Step 3: Training Diverse Classification Models

With the training and testing data partitioned, we now move to the core modeling phase. To facilitate a meaningful comparison via the ROC curve, we will train two fundamentally different classification models: a logistic regression model and a gradient boosted model. The logistic regression classifier represents a linear modeling approach, prized for its simplicity and interpretability of coefficients. Conversely, the gradient boosted model is a sophisticated ensemble method that sequentially builds decision trees to correct the errors of previous trees, typically yielding higher predictive accuracy but often sacrificing some interpretability.

For each model, the workflow remains consistent: first, we instantiate the model object; second, we fit the model to the training subset (`X_train` and `y_train`). Crucially, unlike simply predicting class labels (0 or 1), ROC curves require the predicted likelihood of belonging to the positive class.

Therefore, we must use the dedicated predict_proba method on the test data (`X_test`). This method returns the estimated probabilities, which are the essential inputs for calculating the varying TPR and FPR values needed for the curve generation.

## Step 4: Calculating Metrics and Visualizing the Comparison

Once the predicted probabilities are obtained for both models, we proceed to calculate the necessary metrics for visualization. The scikit-learn library provides the highly efficient roc_curve function, which takes the true class labels (`y_test`) and the predicted probabilities as input, returning the computed False Positive Rate (FPR) and True Positive Rate (TPR) arrays across all thresholds. Simultaneously, we use the roc_auc_score function to derive the corresponding AUC value, quantifying the overall discriminatory performance.

The Matplotlib library is then employed to generate the graphical output. We initiate a clean plot using `plt.figure(0).clf()` to prevent overlaying previous plots. For each model, the calculated FPR and TPR arrays are plotted using `plt.plot()`. A crucial detail for comparative visualization is incorporating the calculated AUC score directly into the plot legend (`label="Model Name, AUC="+str(auc)`). This method creates a clear, labeled visualization where the curves and their single-score performance metrics are presented together, making the model comparison immediate and comprehensive.

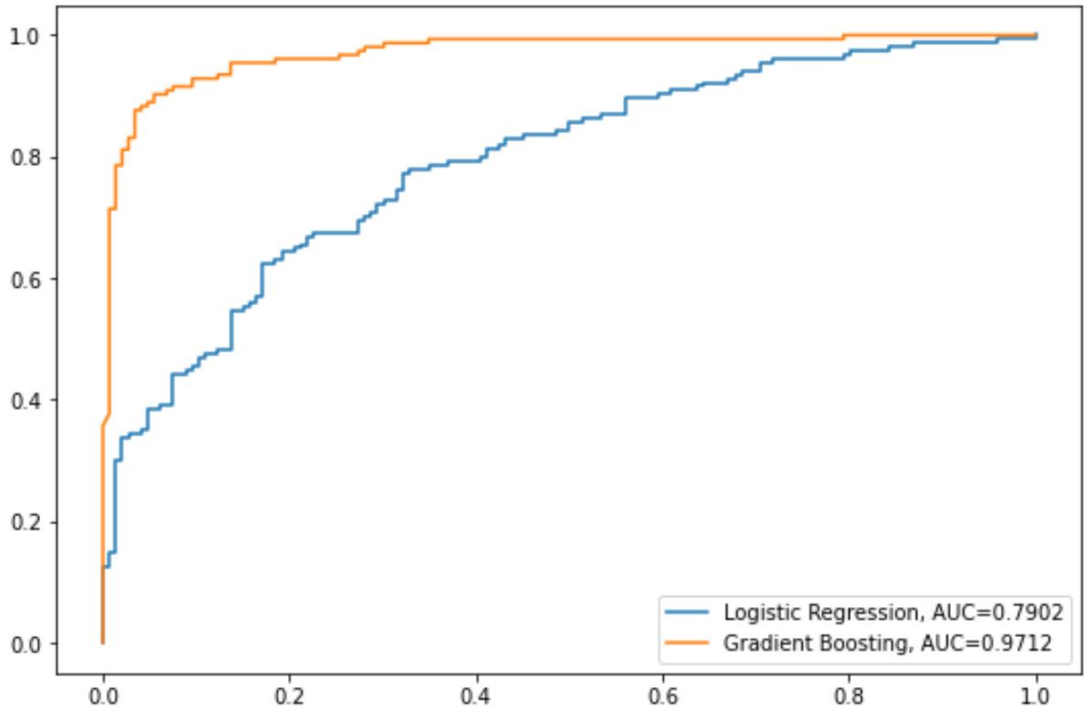
**#set up plotting area**
**plt.figure(0).clf()**

#fit logistic regression model and plot ROC curve
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Logistic Regression, AUC="+str(auc))

#fit gradient boosted model and plot ROC curve
model = GradientBoostingClassifier()
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Gradient Boosting, AUC="+str(auc))

#add legend

plt.legend()




## Step 5: Interpreting Results and Model Selection

The resulting visualization clearly contrasts the performance of the two trained models. In the generated plot, the blue line represents the ROC curve for the logistic regression model, while the orange line illustrates the performance characteristics of the gradient boosted model. This visual comparison immediately informs the data scientist about the trade-offs in True Positive Rate versus False Positive Rate achieved by each model across varying decision thresholds.

As previously established, a superior model exhibits a curve that is positioned closer to the top-left corner (0, 1), indicating high sensitivity and high specificity simultaneously. By visual inspection, the orange curve, representing the gradient boosted model, is noticeably closer to this ideal corner than the blue curve. This visual evidence suggests that the boosted model is significantly better at separating the positive and negative classes in the test data.

To move beyond visual interpretation and provide a firm, quantitative justification for model selection, we rely on the AUC scores, which are conveniently included in the plot legend. The higher the AUC value (closer to 1.0), the better the model's overall discriminatory power. Examining the calculated values confirms the visual assessment:

AUC of logistic regression model: **0.7902**

AUC of gradient boosted model: **0.9712**

The disparity between these results is significant. An AUC of 0.9712 for the gradient boosted model signifies near-excellent performance, demonstrating robust capability in classifying the data into its correct categories. The logistic regression model, while performing decently at 0.7902, is clearly inferior in this scenario. This example perfectly illustrates the critical importance and practical utility of plotting multiple ROC curves and quantifying their performance via the AUC metric for informed and evidence-based model selection in any machine learning task.

## Additional Resources for Further Study

To support continued learning and help deepen your expertise in evaluating classification models and mastering ROC curves, the following high-quality references and documentation are highly recommended:

These resources offer valuable technical insights, theoretical foundations, and additional practical examples that will assist you in mastering the nuances of model evaluation and selection within the field of machine learning.