How to Easily Plot a Precision-Recall Curve in Python

Authored by

stats writer

December 3, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Easily Plot a Precision-Recall Curve in Python*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=104518

ARABPSYCHOLOGY.COM

Introduction to Model Evaluation and the Precision-Recall Curve

In the field of machine learning, accurately assessing the performance of a classification model is paramount. While simple accuracy might suffice for balanced datasets, more sophisticated metrics are required when dealing with real-world scenarios, especially those involving class imbalance. For binary classification problems, the relationship between how many positive predictions were correct and how many actual positive cases were captured is often visualized using a specialized tool: the Precision-Recall Curve (PR Curve).

The Precision-Recall Curve plots precision against recall across various classification thresholds. This visualization provides a comprehensive view of the trade-off inherent in classification tasks. By adjusting the decision threshold, we can prioritize either minimizing false positives (improving precision) or minimizing false negatives (improving recall). The PR curve is particularly informative when one class significantly outweighs the other, offering a clearer picture of classifier performance on the minority class than the traditional Receiver Operating Characteristic (ROC) curve.

Generating this crucial curve in Python is straightforward, thanks to robust libraries like scikit-learn. The dedicated `precision_recall_curve()` function simplifies the process by accepting true class labels and the model's predicted probabilities, subsequently calculating the necessary precision and recall values across all possible decision boundaries. This guide walks through the detailed, step-by-step process of implementing and interpreting the Precision-Recall Curve using Python.

Understanding Core Metrics: Precision and Recall

Before diving into the code, it is essential to solidify the definitions of the two core metrics that form the basis of the curve: precision and recall. These metrics are derived from the outcomes summarized in the confusion matrix (True Positives, False Positives, True Negatives, and False Negatives), allowing us to understand the quality of the model's predictions relative to the actual outcomes. Understanding these definitions is fundamental to interpreting model performance, especially in contexts like fraud detection or medical diagnosis where the cost of different error types varies significantly.

Precision (also known as Positive Predictive Value) measures the accuracy of the positive predictions made by the model. It answers the question: "Of all the instances the model predicted as positive, how many were actually positive?" High precision indicates a low rate of False Positives. If a system must be highly reliable when asserting a positive result (e.g., flagging a safe transaction), high precision is critical.

Precision = True Positives / (True Positives + False Positives)

ARABPSYCHOLOGY.COM

Recall (also known as Sensitivity or True Positive Rate) measures the model's ability to find all the actual positive samples. It answers the question: "Of all the instances that were actually positive, how many did the model correctly identify?" High recall indicates a low rate of False Negatives. If missing an actual positive instance is extremely costly (e.g., missing a cancerous tumor), high recall is prioritized.

Recall = True Positives / (True Positives + False Negatives)

The inherent conflict between these two metrics is what necessitates the Precision-Recall Curve. Often, increasing the system's sensitivity to capture more true positives (increasing recall) simultaneously leads to misclassifying more negative samples as positive (decreasing precision). The PR Curve allows data scientists to visualize this exact trade-off across the entire spectrum of classification decisions determined by the probability threshold.

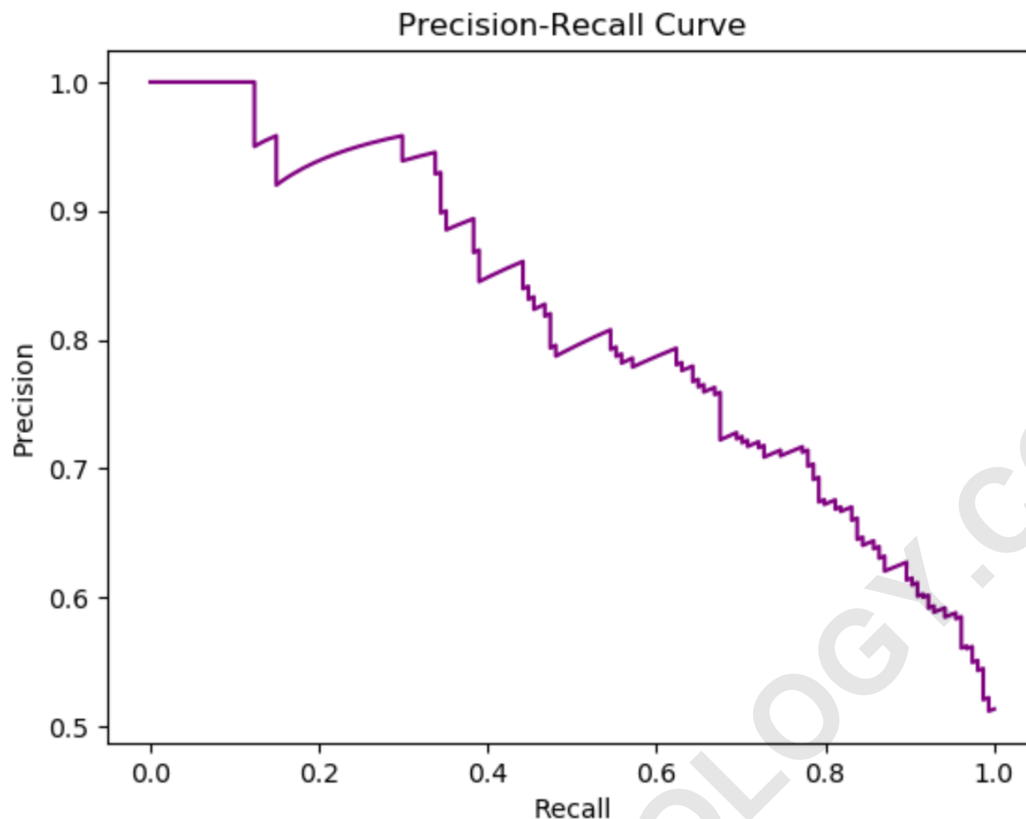
The Importance of the PR Curve in Classification Evaluation

While metrics like accuracy and F1-score provide single-point summaries of model effectiveness, they often obscure the nuances of performance, particularly regarding probability calibration. The PR curve offers a comprehensive graphical evaluation. It is particularly superior to the ROC curve when evaluating models on highly imbalanced datasets. In a scenario where the positive class is rare (e.g., 1% of the data), a high True Negative Rate (specificity, which feeds into the ROC curve) can artificially inflate the perceived quality of the model, even if it performs poorly on the minority positive class. The PR curve, focusing only on the positive class outcomes (precision and recall), avoids this bias.

The shape of the curve itself is highly informative. A model that perfectly separates the classes will exhibit a curve that follows the top right corner of the plot, maintaining high precision even as recall approaches 1.0. Conversely, a poor model will have a curve that closely tracks the baseline precision level (the proportion of positive samples in the dataset). The Area Under the Precision-Recall Curve (AUPRC) serves as a single metric summarizing the overall performance across all possible thresholds, providing a robust measure of model effectiveness on the positive class.

By visualizing the curve, practitioners can make informed decisions about setting the optimal decision threshold for deployment. Depending on the business objective--whether minimizing risk (high precision) or maximizing detection (high recall)--the appropriate operating point can be selected directly from the plot. For instance, in a sensitive medical screening test, one might accept a lower precision rate if it guarantees a very high recall rate, ensuring fewer actual cases are missed.

ARABPSYCHOLOGY.COM



ARABPSYCHOLOGY.COM

Prerequisites and Methodology Overview

To successfully generate the precision-recall curve, we will leverage the powerful data science ecosystem available in Python. The primary tools required are the numerical computing library NumPy (implicitly used by scikit-learn), the visualization library Matplotlib, and, most importantly, the scikit-learn library for model training and metric calculation. We will use a standard binary classification algorithm, Logistic Regression, as the example model, though the method applies universally to any classifier capable of outputting probability scores (e.g., Random Forests, Support Vector Machines with probability calibration, or Neural Networks).

The methodology follows four distinct steps. First, we import the required components. Second, we generate or load a dataset, split it into training and testing subsets, and fit the chosen classifier, ensuring we capture the probability scores for the positive class. Third, we utilize the scikit-learn function `precision_recall_curve` to calculate the coordinates (precision, recall, and corresponding thresholds). Finally, we use Matplotlib to plot these coordinates, creating the visual representation of the model's trade-off profile.

It is crucial to note that the input to the `precision_recall_curve` function requires the predicted probabilities, not the final hard class predictions (0 or 1). This is because the curve generation process involves testing every potential classification threshold derived from these continuous

probabilities. By using the probabilities, scikit-learn can automatically determine the precision and recall values that would result from classifying at each potential boundary, thereby mapping out the complete curve.

Step 1: Importing Necessary Libraries

The first step in any robust Python data science project is to import the necessary packages. We require modules for data generation, model handling, metric calculation, and visualization. Specifically, we import `datasets` and `model_selection` for preparing the input data, `LogisticRegression` for our classification task, `precision_recall_curve` from the `metrics` module to calculate the curve values, and `matplotlib.pyplot` for plotting the final result.

Ensuring all imports are properly configured guarantees a smooth execution of the subsequent steps, allowing us to seamlessly integrate data preparation, model training, performance evaluation, and graphical output within a single script. This foundational step establishes the environment necessary for reproducible machine learning evaluation.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
```

Step 2: Preparing Data and Fitting the Logistic Regression Model

Next, we prepare the data and train our chosen model. For demonstration purposes, we utilize scikit-learn's synthetic data generator, `make_classification`, to quickly create a balanced binary classification dataset with 1000 samples and four features. This synthetic data provides a clean environment to demonstrate the calculation of the PR curve without external data dependencies.

Following data generation, the dataset is segmented into training and testing sets using `train_test_split`. This separation is standard practice to ensure that model performance is evaluated on unseen data, preventing optimistic bias in our evaluation metrics. A 70/30 split (70% for training, 30% for testing) is employed here. The Logistic Regression classifier is then fitted exclusively on the training data (`X_train` and `y_train`).

Crucially, after fitting the model, we use the trained classifier to generate probability predictions on the held-out test set (`X_test`). The `predict_proba` method returns the probability of an instance belonging to each class. For binary classification, we are interested in the probability of the positive class (often indexed as 1), which is extracted by slicing the results using . This vector of probability

scores (`y_score`) is the necessary input for calculating the precision-recall values in the next step.

```
#create dataset with 5 predictor variables
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)

#fit logistic regression model to dataset
classifier = LogisticRegression()
classifier.fit(X_train, y_train)

#use logistic regression model to make predictions
y_score = classifier.predict_proba(X_test)
```

ARABPSYCHOLOGY.COM

Step 3: Generating the Precision and Recall Values

With the true labels (`y_test`) and the predicted positive class probabilities (`y_score`) available, we can now compute the data required for the Precision-Recall Curve. This is achieved using scikit-learn's `precision_recall_curve` function. This function intelligently iterates through all unique probability scores present in `y_score`, treating each as a potential classification threshold.

The function returns three arrays: `precision`, `recall`, and `thresholds`. The `precision` and `recall` arrays contain the corresponding precision and recall metrics calculated at each specific threshold value. It is important to note that the `thresholds` array will be one element shorter than the precision and recall arrays, as the highest possible threshold (which yields 100% precision and 0% recall) is usually implied rather than explicitly stored.

These arrays form the backbone of our visualization. Each (recall, precision) pair represents an operating point on the curve, dictating the model's behavior at a specific level of certainty. By obtaining these numerical results, we transition from model training to detailed performance analysis, setting the stage for graphical interpretation.

```
#calculate precision and recall
precision, recall, thresholds = precision_recall_curve(y_test, y_score)
```

Step 4: Visualizing the Precision-Recall Tradeoff

The final step involves using Matplotlib to plot the calculated precision and recall values. We create a figure and an axes object, and then plot recall values on the x-axis and precision values on the y-axis. This graphical representation immediately illustrates the crucial trade-off inherent in the model's performance: the extent to which we must sacrifice precision to achieve higher recall, or vice versa.

Proper visualization requires clear axis labels and a title to ensure the plot is immediately understandable. We title the plot "Precision-Recall Curve" and label the axes accordingly. The curve itself is plotted using `ax.plot(recall, precision)`, connecting all the calculated operating points derived from the varying thresholds. Displaying the plot with `plt.show()` renders the final result, providing the primary artifact for performance analysis.

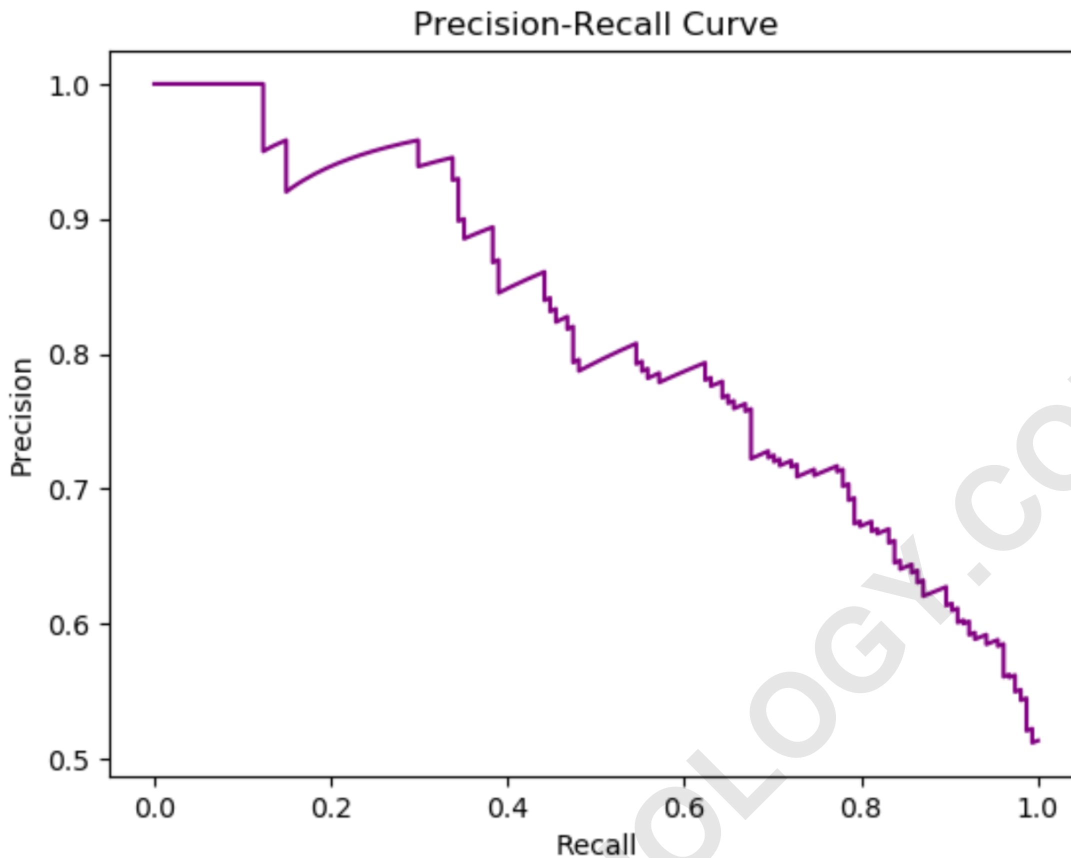
This visualization allows for quick comparison between different models or different hyperparameter settings for the same model. A model that performs better will generally have its curve closer to the upper-right corner (high precision across high recall values), signifying superior positive class prediction capabilities.

```
#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()
```

ARABPSYCHOLOGY.COM



ARABPSYCHOLOGY.COM

Interpreting the Precision-Recall Curve

The resulting plot clearly maps the relationship between recall (x-axis) and precision (y-axis) for our Logistic Regression model. As observed on the graph, the curve typically slopes downward. This downward trend is a graphical confirmation of the trade-off principle: as we strive to increase the model's ability to find all positive instances (increase recall), we necessarily loosen the classification criteria (lower the probability threshold), which in turn increases the number of false positives, thus reducing precision.

The ideal scenario for a classification model is to achieve high precision and high recall simultaneously. While a perfect model would yield a rectangular curve hitting the coordinates (1.0, 1.0), real-world models inevitably demonstrate this curvature. The goal is often to find the "elbow" of the curve, representing the optimal balance point where a small decrease in precision yields a substantial increase in recall, or vice versa. This operating point corresponds directly to a specific decision threshold that can be read from the calculated `thresholds` array generated in Step 3.

For deployment, the interpretation of the PR curve must align with the operational costs. If the application prioritizes avoiding false alarms (e.g., in quality control where incorrectly stopping the line is expensive), we would choose a high-precision operating point, accepting lower recall.

Conversely, if the application demands minimizing missed opportunities (e.g., detecting critical infrastructure failures), we prioritize high recall, even if it introduces more false positives (lower precision). The PR curve is the essential visual tool that empowers data scientists to make these critical, context-driven decisions.

Conclusion: Leveraging the PR Curve for Robust Model Selection

Generating the precision-recall curve in Python using scikit-learn is a highly effective method for robustly evaluating binary classification models, especially those operating on imbalanced data. The `precision_recall_curve()` function efficiently calculates the entire range of potential performance outcomes based on varying probability thresholds, requiring only the true labels and predicted probabilities as input.

By transforming these numerical arrays into a clear Matplotlib plot, data scientists gain insight into the fundamental trade-off between minimizing false positives and minimizing false negatives. This visualization is far more informative than single aggregated metrics like the F1-score because it reveals the behavior of the model across all possible operating points, enabling truly informed decision-making regarding model deployment and optimization. Mastering the creation and interpretation of the PR curve is a prerequisite for professional model evaluation.

The example provided, using Logistic Regression, demonstrates the complete pipeline from data preparation to final visualization. This methodology can be easily adapted to evaluate any other probabilistic classification algorithm, ensuring that model performance is assessed accurately against real-world operational constraints.

ARABPSYCHOLOGY.COM