

[Open in app](#)

Doug Steen

78 Followers

[About](#)[Follow](#)

Precision-Recall Curves

Sometimes a curve is worth a thousand words - how to calculate and interpret precision-recall curves in Python.



[Doug Steen](#) Sep 19, 2020 · 7 min read



Photo by [Lance Anderson](#) on [Unsplash](#)

In a previous post, I covered ROC curves and AUC — how to calculate them, and how to interpret them. Today, I'm going to run through another exercise for a similar binary classification performance measure — **precision-recall curves**.

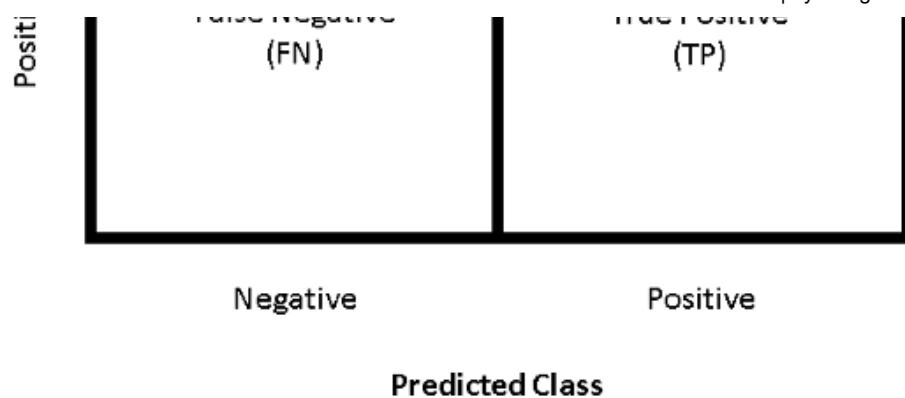
Precision-Recall Curves

Much like the **ROC curve**, The **precision-recall curve** is used for evaluating the performance of binary classification algorithms. It is often used in situations where classes are heavily imbalanced. Also like ROC curves, precision-recall curves provide a graphical representation of a classifier's performance across many thresholds, rather than a single value (e.g., accuracy, f-1 score, etc.).

Precision and Recall

To understand precision and recall, let's quickly refresh our memory on the possible outcomes in a binary classification problem. Any prediction relative to labeled data can be a **true positive**, **false positive**, **true negative**, or **false negative**.

Actual Class	Negative	True Negative (TN)	False Positive (FP)
	Positive	False Negative	True Positive



Confusion matrix structure for binary classification problems

The precision-recall curve is constructed by calculating and plotting the **precision** against the **recall** for a single classifier at a variety of **thresholds**. For example, if we use logistic regression, the threshold would be the predicted probability of an observation belonging to the positive class. Normally in logistic regression, if an observation is predicted to belong to the positive class at probability > 0.5 , it is labeled as positive. However, we could really choose any probability threshold between 0 and 1. A precision-recall curve helps to visualize how the choice of threshold affects classifier performance, and can even help us select the best threshold for a specific problem.

Precision (also known as **positive predictive value**) can be represented as:

$$precision = PPV = \frac{TP}{TP + FP}$$

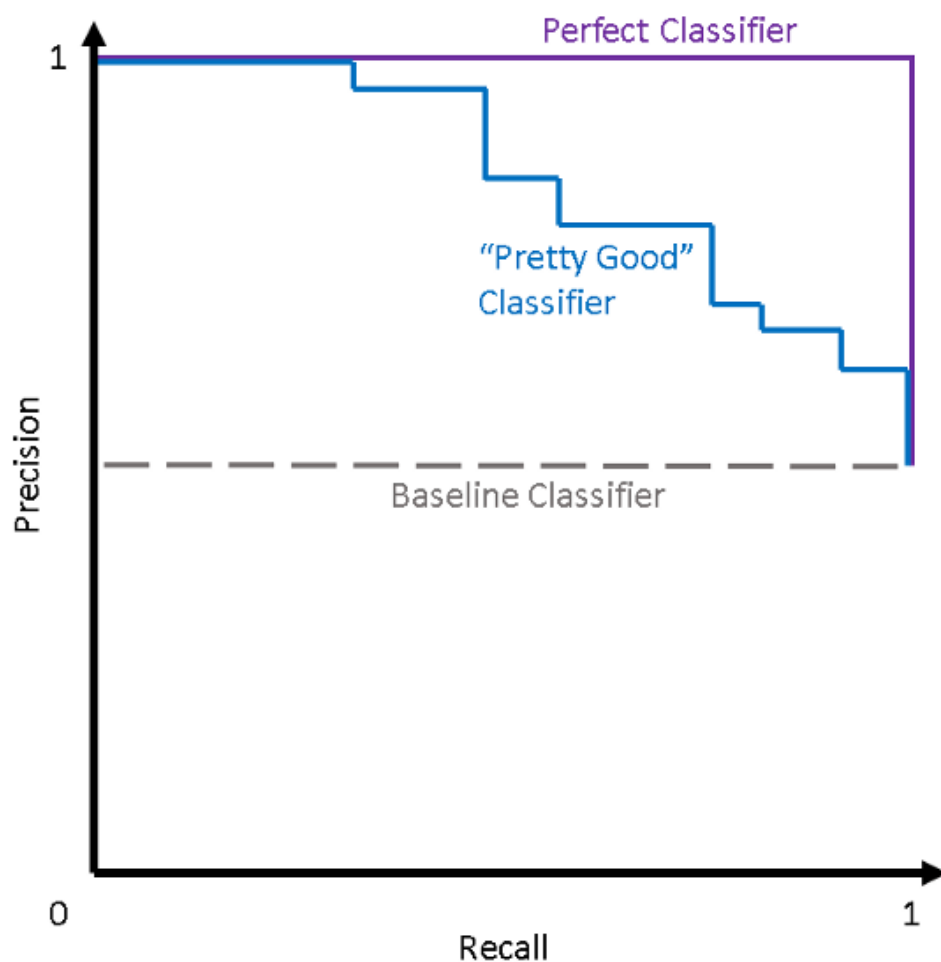
where **TP** is the number of **true positives** and **FP** is the number of **false positives**. Precision can be thought of as the fraction of positive predictions that *actually* belong to the positive class.

Recall (also known as **sensitivity**) can be represented as:

$$recall = sensitivity = \frac{TP}{TP + FN}$$

where **TP** is the number of **true positives** and **FN** is the number of **false negatives**. Recall can be thought of as the fraction of positive predictions out of all positive instances in the data set.

The figure below demonstrates how some theoretical classifiers would plot on a precision-recall curve. The gray dotted line represents a “baseline” classifier — this classifier would simply predict that all instances belong to the positive class. The purple line represents an ideal classifier — one with perfect precision and recall at all thresholds. Nearly all real-world examples will fall somewhere between these two lines — not perfect, but providing better predictions than the “baseline”. A good classifier will maintain both a high precision and high recall across the graph, and will “hug” the upper right corner in the figure below.



Some theoretical precision-recall curves

AUC-PR and AP

Much like ROC curves, we can summarize the information in a precision-recall curve with a single value. This summary metric is the **AUC-PR**. AUC-PR stands for **area under the (precision-recall) curve**. Generally, the higher the AUC-PR score, the better a classifier performs for the given task.

One way to calculate AUC-PR is to find the **AP**, or average precision. The documentation for `sklearn.metrics.average_precision_score` states, “AP summarizes a precision-recall curve as the weighted mean of precision achieved at each threshold, with the increase in recall from the previous threshold used as the weight.” So, we can think of AP as a kind of weighted-average precision across all thresholds.

In a perfect classifier, $\text{AUC-PR} = 1$. In a “baseline” classifier, the AUC-PR will depend on the fraction of observations belonging to the positive class. For example, in a balanced binary classification data set, the “baseline” classifier will have $\text{AUC-PR} = 0.5$. A classifier that provides some predictive value will fall between the “baseline” and perfect classifiers.

Example: Heart Disease Prediction

I’ve always found it a valuable exercise to calculate metrics like the precision-recall curve from scratch — so that’s what I’m going to do with the [Heart Disease UCI](#) data set in Python. The data set has 14 attributes, 303 observations, and is typically used to predict whether a patient has heart disease based on the other 13 attributes, which include age, sex, cholesterol level, and other measurements.

Imports & Load Data

```
1  # Imports
2  import pandas as pd
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  # Load data
```

```
7 df = pd.read_csv('heart.csv')
8 df.head()
```

pr_imports.py hosted with ❤ by GitHub

[view raw](#)

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Train-Test Split

For this analysis, I'll use a standard 75% — 25% train-test split.

Logistic Regression Classifier

Before I define a precision and recall function, I'll fit a vanilla Logistic Regression classifier on the training data, and make predictions on the test set.

```
1 # Fit a vanilla Logistic Regression classifier and make predictions
2 from sklearn.linear_model import LogisticRegression
3
4 clf = LogisticRegression(max_iter=1000)
5 clf.fit(X_train, y_train)
6 y_pred_test = clf.predict(X_test)
```

pr_logreg.py hosted with ❤ by GitHub

[view raw](#)

Calculating Precision and Recall

With these test predictions, I can write and test a function to calculate the precision and recall. I'll need to count how many predictions are true positives, false positives, and false negatives. After that, it's just a little simple math using the precision and recall formulas.

```
1 # Function to calculate Precision and Recall
2
3 def calc_precision_recall(y_true, y_pred):
4
5     # Convert predictions to series with index matching y_true
6     y_pred = pd.Series(y_pred, index=y_true.index)
7
8     # Instantiate counters
9     TP = 0
10    FP = 0
11    FN = 0
12
13    # Determine whether each prediction is TP, FP, TN, or FN
14    for i in y_true.index:
15        if y_true[i]==y_pred[i]==1:
```

```
16         TP += 1
17     if y_pred[i]==1 and y_true[i]!=y_pred[i]:
18         FP += 1
19     if y_pred[i]==0 and y_test[i]!=y_pred[i]:
20         FN += 1
21
22     # Calculate true positive rate and false positive rate
23     # Use try-except statements to avoid problem of dividing by 0
24     try:
25         precision = TP / (TP + FP)
26     except:
27         precision = 1
28
29     try:
30         recall = TP / (TP + FN)
31     except:
32         recall = 1
33
34     return precision, recall
35
36 # Test function
37
38 calc_precision_recall(y_test, y_pred_test)
```

pr_function.py hosted with ❤ by GitHub

[view raw](#)

```
(0.7941176470588235, 0.6923076923076923)
```

The initial logistic regulation classifier has a precision of 0.79 and recall of 0.69 — not bad! Now let's get the full picture using precision-recall curves.

Varying the Probability Threshold

To calculate the precision-recall curve, I need to vary the **probability threshold** that the logistic regression classifier uses to predict whether a patient has heart disease (target=1) or doesn't (target=0). Remember, while logistic regression is used to assign a class label, what it's *actually* doing is determining the *probability* that an observation belongs to a specific class. In a typical binary classification problem, an observation

must have a probability of > 0.5 to be assigned to the positive class. However, in this case, I will vary that threshold probability value incrementally from 0 to 1.

In the code blocks below, I obtain the precision and recall scores across a range of threshold probability values. For comparison, I use logistic regression with (1) no regularization and (2) L2 regularization.

Plotting the Precision-Recall Curve



These curves give the shape we would expect — at thresholds with low recall, the precision is correspondingly high, and at very high recall, the precision begins to drop. The two versions of the classifier have similar performance, but it looks like the l2-regularized version slightly edges out the non-regularized one.

Additionally, both classifiers can achieve a precision score of about 0.8 while only sacrificing minimal recall!

Calculating AUC-PR

The AUC-PR score can be calculated using one of two useful functions in `sklearn.metrics`. **`auc()`** and **`average_precision_score()`** will both do the job. The only difference is the inputs required for each function: **`auc()`** takes the precision and recall scores themselves, and **`average_precision_score()`** takes the test data labels and the classifier's test prediction probabilities.

```
LR (No reg.) AUC-PR: 0.91  
LR (L2 reg.) AUC-PR: 0.92
```

```
LR (No reg.) Avg. Prec.: 0.91  
LR (L2 reg.) Avg. Prec.: 0.92
```

As expected, the two functions for calculating AUC-PR return the same results! The values also confirm what we can see visually on the graph — the l2-regularized classifier performs slightly better than the non-regularized classifier.

Precision-Recall Curves using sklearn

In addition to providing functions to calculate AUC-PR, sklearn also provides a function to efficiently plot a precision-recall curve —

`sklearn.metrics.plot_precision_recall_curve()`. This function requires only a classifier (fit on training data) and the test data as inputs.



As you can see, the output plots include the AP score, which we now know also represents the AUC-PR score.

References

Precision-Recall - scikit-learn 0.23.2 documentation

Example of Precision-Recall metric to evaluate classifier output

quality. Precision-Recall is a useful measure of...

scikit-learn.org

sklearn.metrics.auc - scikit-learn 0.23.2 documentation

scikit-learn: machine learning in Python

scikit-learn.org

sklearn.metrics.average_precision_score - scikit-learn 0.23.2 documentation

Compute average precision (AP) from prediction scores AP summarizes a precision-recall curve as the weighted mean of...

scikit-learn.org

sklearn.metrics.plot_precision_recall_curve - scikit-learn 0.23.2 documentation

Plot Precision Recall Curve for binary classifiers. Extra keyword arguments will be passed to matplotlib's . Read more...

scikit-learn.org

ROC Curves and Precision-Recall Curves for Imbalanced Classification - Machine Learning Mastery

Most imbalanced classification problems involve two classes: a negative case with the majority of examples and a...

machinelearningmastery.com

Machine Learning

Precision Recall

Data Science

Python

Classification

About Write Help Legal

Get the Medium app

