**Understanding the ROC Curve and AUC**

These binary classification performance measures go hand-in-hand — let’s explore.

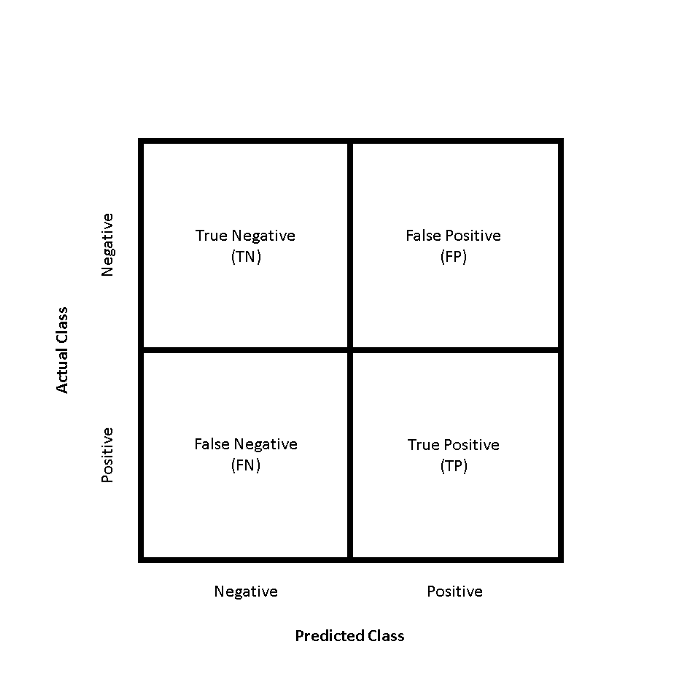


Photo by [Madeline Pere](https://unsplash.com/@mpere?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

**The ROC Curve**

The **receiver operating characteristic (ROC) curve** is frequently used for evaluating the performance of binary classification algorithms. It provides a graphical representation of a classifier’s performance, rather than a single value like most other metrics.

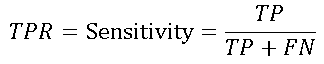
First, let’s establish that in binary classification, there are four possible outcomes for a test prediction: **true positive**, **false positive**, **true negative**, and **false negative**.



Confusion matrix structure for binary classification problems

The ROC curve is produced by calculating and plotting the **true positive rate**against the **false positive rate**for a single classifier at a variety of **thresholds**. For example, in 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 be positive at > 0.5 probability, it is labeled as positive. However, we could really choose any threshold between 0 and 1 (0.1, 0.3, 0.6, 0.99, etc.) — and ROC curves help us visualize how these choices affect classifier performance.

The **true positive rate**, or **sensitivity**, can be represented as:



where **TP** is the number of **true positives** and **FN** is the number of **false negatives**. The true positive rate is a measure of the probability that an *actual* positive instance will be classified as positive.

The **false positive rate**, or 1 —**specificity**, can be written as:

https://miro.medium.com/max/382/1*vhGYw-cuJjo-C8rl9pM_nA.png

where **FP** is the number of **false positives** and **TN** is the number of **true negatives**. The false positive rate is essentially a measure of how often a “false alarm” will occur — or, how often an *actual* negative instance will be classified as positive.

Figure 1 demonstrates how some theoretical classifiers would plot on an ROC curve. The gray dotted line represents a classifier that is no better than random guessing — this will plot as a diagonal line. The purple line represents a perfect classifier — one with a true positive rate of 100% and a false positive rate of 0%. Nearly all real-world examples will fall somewhere between these two lines — not perfect, but providing more predictive power than random guessing. Typically, what we’re looking for is a classifier that maintains a high true positive rate while also having a low false positive rate — this ideal classifier would “hug” the upper left corner of Figure 1, much like the purple line.

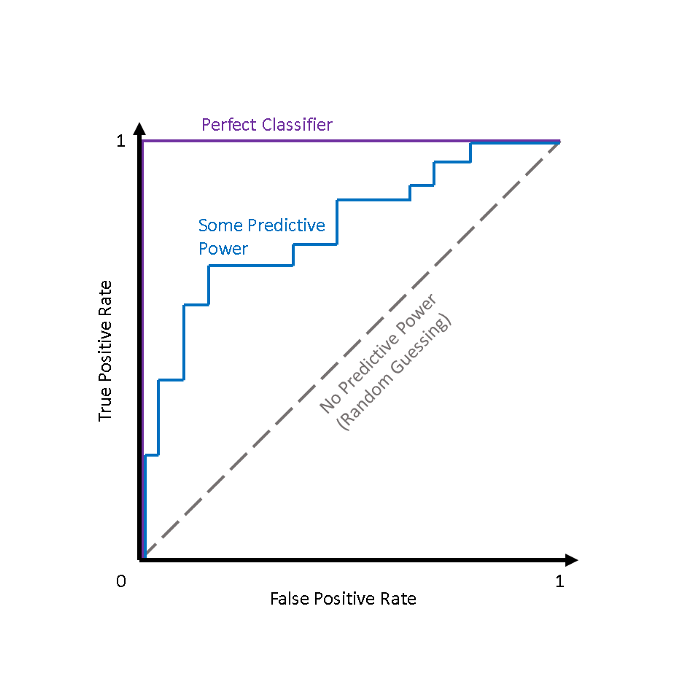


Fig. 1 — Some theoretical ROC curves

**AUC**

While it is useful to visualize a classifier’s ROC curve, in many cases we can boil this information down to a single metric — theAUC.

**AUC**stands for **area under the (ROC) curve.**Generally, the higher the AUC score, the better a classifier performs for the given task.

Figure 2 shows that for a classifier with no predictive power (i.e., random guessing), AUC = 0.5, and for a perfect classifier, AUC = 1.0. Most classifiers will fall between 0.5 and 1.0, with the rare exception being a classifier performs *worse*than random guessing (AUC < 0.5).

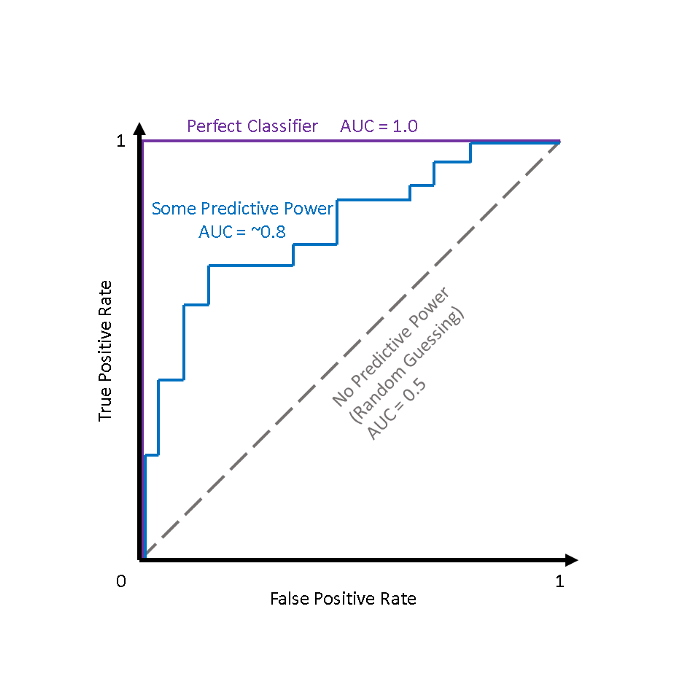


Fig. 2 — Theoretical ROC curves with AUC scores

**Why use ROC Curves?**

One advantage presented by ROC curves is that they aid us in finding a classification threshold that suits our specific problem.

For example, if we were evaluating an email spam classifier, we would want the false positive rate to be really, really low. We wouldn’t want someone to lose an important email to the spam filter just because our algorithm was too aggressive. We would probably even allow a fair amount of actual spam emails (true positives) through the filter just to make sure that no important emails were lost.

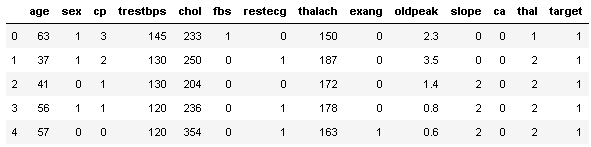
On the other hand, if our classifier is predicting whether someone has a terminal illness, we might be ok with a higher number of false positives (incorrectly diagnosing the illness), just to make sure that we don’t miss any true positives (people who actually have the illness).

Additionally, ROC curves and AUC scores also allow us to compare the performance of different classifiers for the same problem.

**Example: Heart Disease Prediction**

To demonstrate how the ROC curve is constructed in practice, I’m going to work with the [Heart Disease UCI](https://www.kaggle.com/ronitf/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 & Loading Data**



**Train-Test Split**

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

**Logistic Regression Classifier**

Before I write a function to calculate false positive and true positive rate, I’ll fit a vanilla Logistic Regression classifier on the training data, and make predictions on the test set.

**Calculating True Positive Rate and False Positive Rate**

Now that I have test predictions, I can write a function to calculate the true positive rate and false positive rate. This is a critical step, as these are the two variables needed to produce the ROC curve.

(0.6923076923076923, 0.1891891891891892)

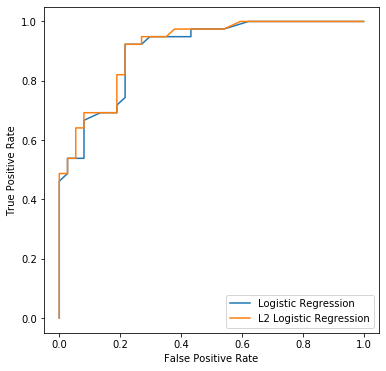
The test shows that the function appears to be working — a true positive rate of 69% and a false positive rate of 19% are perfectly reasonable results.

**Exploring varying thresholds**

To obtain the ROC curve, I need more than one pair of true positive/false positive rates. I need to vary the **threshold probability**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. This will result in the ranges of true positive rates and false positive rates that allow me to build the ROC curve.

In the code blocks below, I obtain these true positive rates and false positive rates across a range of threshold probability values. For comparison, I use logistic regression with (1) no regularization and (2) L2 regularization.

**Plotting the ROC Curves**



Both versions of the logistic regression classifier seem to do a pretty good job, but the L2 regularized version appears to perform slightly better.

**Calculating AUC scores**

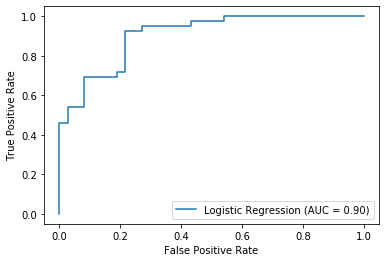
**sklearn** has an **auc()** function, which I’ll make use of here to calculate the AUC scores for both versions of the classifier. **auc()** takes in the true positive and false positive rates we previously calculated, and returns the AUC score.

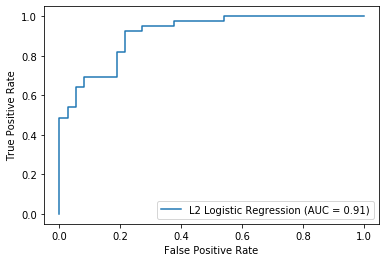
Logistic Regression (No reg.) AUC 0.902979902979903  
Logistic Regression (L2 reg.) AUC 0.9116424116424116

As expected, the classifiers both have similar AUC scores, with the L2 regularized version performing slightly better.

**ROC curves and AUC the easy way**

Now that we’ve had fun plotting these ROC curves from scratch, you’ll be relieved to know that there is a much, much easier way. **sklearn**’s **plot\_roc\_curve()**function can efficiently plot ROC curves using only a fitted classifier and test data as input. These plots conveniently include the AUC score as well.





**Closing**

If you’ve made it this far, thanks for reading! I found it a valuable exercise to inefficiently create my own ROC curves in Python, and I hope you gained something from following along.

**Some helpful references on ROC and AUC:**