**Statistics for Machine Learning**

**Stop Using 0.5 as the Threshold for Your Binary Classifier**

**Learn how to set the optimal threshold for your Machine Learning model**

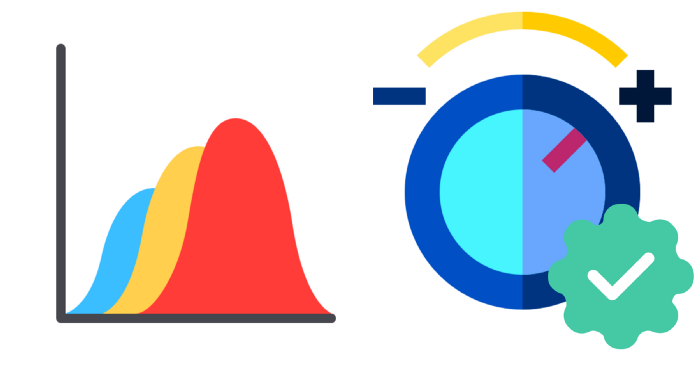


Image by author, using image files from flaticon.com

To produce a binary response, classifiers output a real-valued score that is thresholded. For example, logistic regression outputs a probability (a value between 0.0 and 1.0); and observations with a score equal to or higher than 0.5 produce a positive binary output (many other models use the 0.5 threshold by default).

However, using the default 0.5 threshold is suboptimal. In this blog post, I'll show you how you can choose the best threshold from your binary classifier. We'll be using [Ploomber](https://github.com/ploomber/ploomber?utm_source=medium&utm_medium=blog&utm_campaign=threshold) to execute our experiments in parallel and [sklearn-evaluation](https://github.com/ploomber/sklearn-evaluation?utm_source=medium&utm_medium=blog&utm_campaign=threshold) to generate the plots.

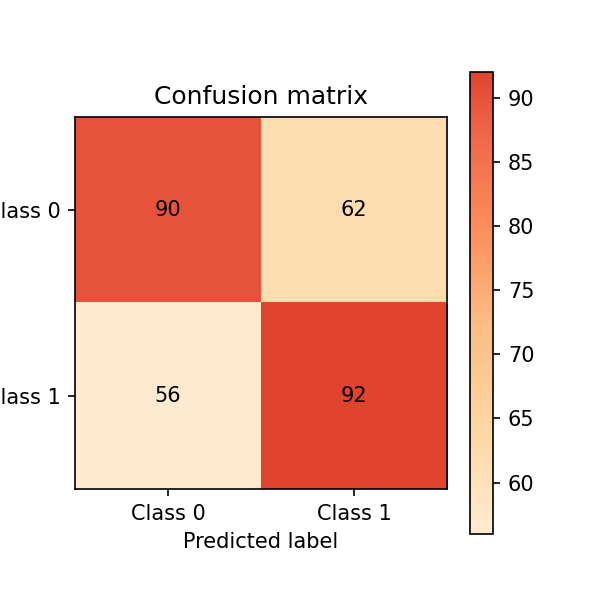
Hi! My name is Eduardo, and I like writing about all things data science. If you want to keep up-to-date with my content. Follow me on [Medium](https://medium.com/@edublancas?utm_source=medium&utm_medium=blog&utm_campaign=threshold) or [Twitter](https://twitter.com/edublancas?utm_source=medium&utm_medium=blog&utm_campaign=threshold). Thanks for reading!

Let’s continue with the example of training a logistic regression. Let’s imagine we’re working on a content moderation system, and our model should flag posts (images, videos, etc.) that contain harmful content; then, a human will take a look and decide whether the content is taken down.

The following snippet trains our classifier:

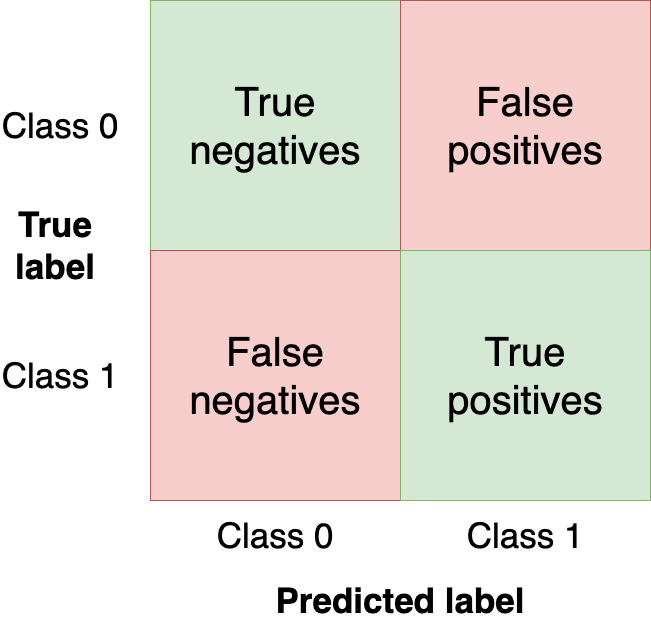
Let’s now make predictions on the test set and evaluate performance via a confusion matrix:

**Console output (1/1):**



Model’s confusion matrix. Image by author.

A confusion matrix summarizes the performance of our model in four regions:



Confusion matrix regions. Image by author.

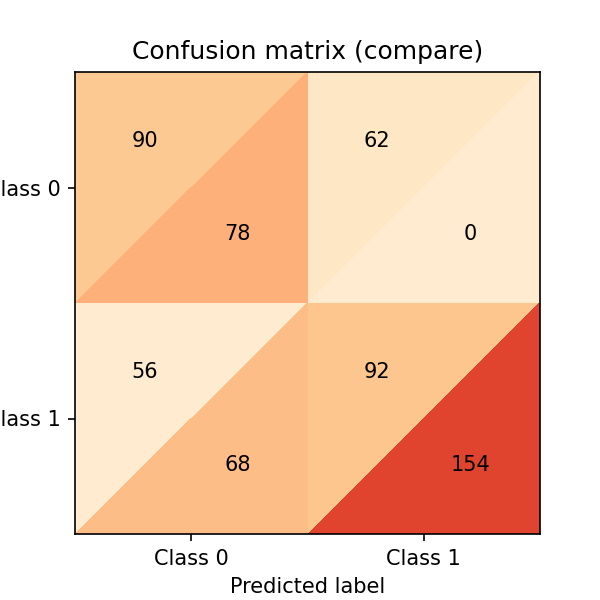
We want to get as many observations as we can (from the test set) in the upper-left and bottom-right quadrants since those are observations that our model got right. The other quadrants are model mistakes.

Changing the threshold of our model will change the values in the confusion matrix. In the previous example, we used the clf.predict function, which returns a binary response (i.e., uses 0.5 as threshold); however, we can use the clf.predict\_proba function to get the raw probability and use a custom threshold:

Let’s now make our classifier a bit more *aggressive* by setting a lower threshold (i.e., flag more posts as harmful) and create a new confusion matrix:

Let’s compare both matrices. The [sklearn-evaluation](https://github.com/ploomber/sklearn-evaluation?utm_source=medium&utm_medium=blog&utm_campaign=threshold) library allows us to do that easily:

**Console output (1/1):**



Combined confusion matrix. Image by author.

The upper triangles are from our 0.5 threshold, and the lower ones are from the 0.4 threshold. A few things to notice:

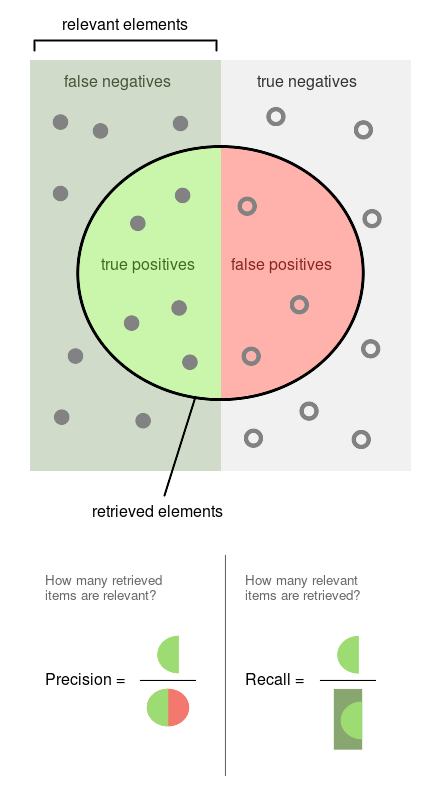
* Both models are predicting 0 for the same number of observations (this is a coincidence). 0.5 threshold: (90 + 56 = 146). 0.4 threshold: (78 + 68 = 146)
* Lowering the threshold is causing a *few more false positives* (68 from 56)
* Lowering the threshold is *increasing true positives a lot* (154 from 92)

As you can see, the tiny threshold changes hugely affected the confusion matrix. However, we’ve only analyzed two threshold values. Let’s analyze model performance across all values to understand the threshold dynamics better. But before that, let’s define new metrics we’ll use for model evaluation.

So far, we’ve evaluated our models with absolute numbers. To ease comparison and evaluation, we’ll now define two normalized metrics (they take values between 0.0 and 1.0).

Precision is the proportion of flagged observations that are events (i.e., posts that our model thinks are harmful, and they are). On the other hand, recall is the proportion of actual events that our model retrieves (i.e., from all the harmful posts, which proportion of them we’re able to detect).

You can see both definitions graphically in the following diagram (Source: [Wikipedia](https://upload.wikimedia.org/wikipedia/commons/2/26/Precisionrecall.svg))



Precision and recall diagram. Source: Wikipedia.

Since both precision and recall are proportions, they are on the same zero-to-one scale. So let’s now proceed to run the experiments.

**Running experiments**

We’ll obtain the precision, recall, and other statistics along several threshold values to better understand how the threshold affects them. We’ll also repeat the experiment multiple times to measure the variability.

*Note: the commands in this section are bash commands. Execute them in a terminal or add the %%sh magic if using Jupyter.*

To efficiently scale our work, we’ll run our experiments using [Ploomber Cloud](https://www.cloud.ploomber.io/signin.html?utm_source=medium&utm_medium=blog&utm_campaign=threshold). It allows us to run experiments in parallel and retrieve the results quickly.

We [created a notebook](https://github.com/ploomber/posts/blob/master/threshold/fit.ipynb?utm_source=medium&utm_medium=blog&utm_campaign=threshold) that fits one model and computes the statistics for several threshold values. We’ll execute the same notebook 20 times in parallel. First, let’s download the notebook:

**Console output (1/1):**

Let’s execute the notebook (the configuration in the notebook file tells Ploomber Cloud to run it 20 times in parallel):

**Console output (1/1):**

After a few minutes, we’ll see that our 20 experiments are finished:

**Console output (1/1):**

Let’s download the results from each experiment. The results are stored in .csv files:

**Console output (1/1):**

**Visualizing experiment results**

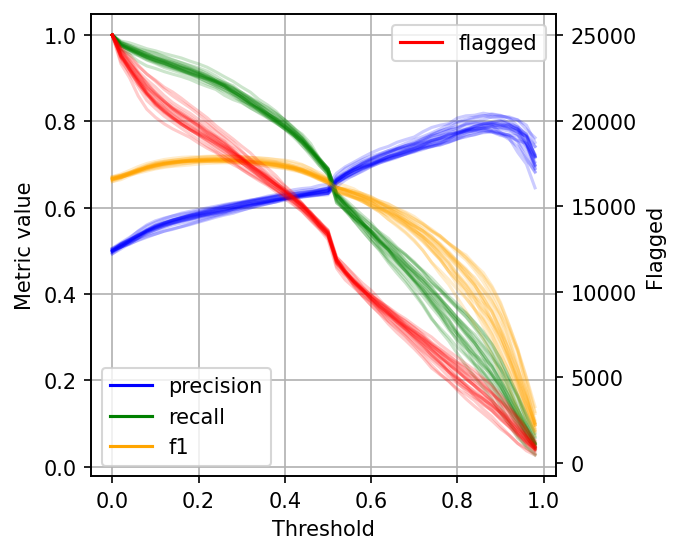
We’ll now load the results of all experiments and plot them all at once.

The left scale (zero to one) measures our three metrics: precision, recall, and F1. The F1 score is the harmonic mean between precision and recall, the best value of the F1 score is 1.0 and the worst is 0.0; F1 values both precision and recall equally so you can see it as balancing between the two. If you are working on a use case where both precision and recall are important, maximizing F1 is an approach that can help you optimize your classifier's threshold.

We also included a red curve (scale on the right side) showing the number of cases that our model flagged as harmful content. This statistic is relevant because there’s a limit on the number of events we can intervene in many real-world use cases.

Following our content moderation example, we might have an X number of people looking at the posts flagged by our model, and there's a limit on how many they can review. Hence, considering the number of flagged cases can help us better choose a threshold: there's no benefit in finding 10,000 daily cases if we can only review 5,000. And it'd be wasteful for our model to only flag 100 daily cases if we have more capacity than that.

**Console output (1/1):**



Metrics for difference threshold values. Image by author.

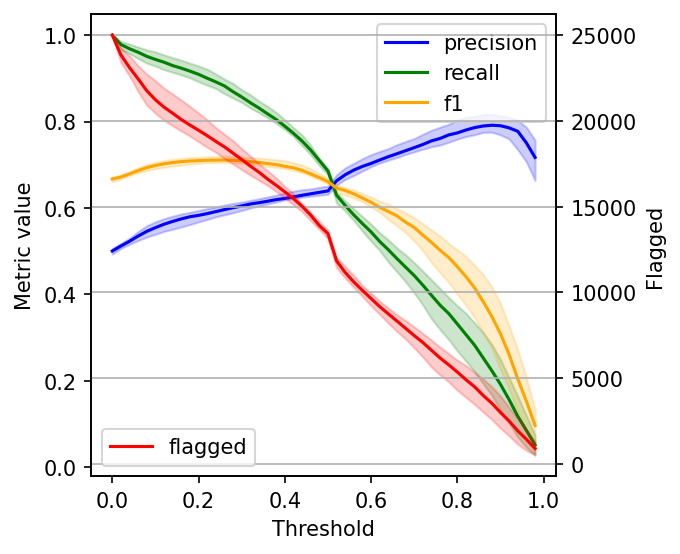
As you can see, when setting low thresholds, we have high recall (we retrieve a large proportion of actually harmful posts) but low precision (there are many non-harmful flagged posts). However, if we increase the threshold, the situation reverses: recall goes down (we missed many harmful posts), but precision is high (most flagged posts are harmful).

When choosing a threshold for our binary classifier, we must compromise on the precision or recall since no classifier is perfect. So let’s discuss how you can reason about choosing the suitable threshold.

**Plotting intervals**

The data gets noisy on the right side (larger threshold values). So, to clean it up a bit, we’ll re-create the plot, but this time, I’ll plot the 2.5%, 50%, and 97.5% percentiles instead of plotting all values.

**Console output (1/1):**



Metrics for difference threshold values. Image by author.

**Selecting the optimal threshold**

When choosing a threshold, we can ask ourselves: is it more important to retrieve as many harmful posts as possible (high recall)? Or is it more important to have high certainty that the ones we flag are harmful (high precision)?

If both are equally important, one common way to optimize under these conditions is to maximize the F-1 score:

**Console output (1/1):**

However, it’s hard to decide what to compromise in many situations, so incorporating some constraints will help.

Say we have ten people reviewing harmful posts, and they can check 5,000 posts together. Let’s see our metrics if we fix our threshold, so it flags approximately 5,000 posts:

**Console output (1/1):**

However, when presenting the results, we might want to show a few alternatives: the model performance under the current constraints (5,000 posts) and how better we could do if we increase the team (e.g., by doubling the size).

**Closing remarks**

The optimal threshold for your binary classifier is the one that optimizes for business outcomes and takes into account process limitations. With the processes described in this post, you’re better equipped to decide the optimal threshold for your use case.

If you have questions about this post, feel free to ask in our [Slack community](https://ploomber.io/community?utm_source=medium&utm_medium=blog&utm_campaign=threshold), which gathers hundreds of data scientists worldwide.

Also, remember to sign up for [Ploomber Cloud](https://www.cloud.ploomber.io/signin.html?utm_source=medium&utm_medium=blog&utm_campaign=threshold)! There’s a free tier! It’ll help you quickly scale up your analysis without dealing with complex cloud infrastructure.

**Packages used**

Here are the package versions we used when writing this post:

**Console output (1/1):**

*Originally published at* [*ploomber.io*](https://ploomber.io/blog/threshold/?utm_source=medium&utm_medium=blog&utm_campaign=threshold)*.*