Multi-Class Metrics Made Simple, Part I: Precision and Recall

Boaz Shmueli Follow
Jul 2, 2019 · 7 min read ★

Performance metrics for precision and recall in multi-class classification can be a little — or very — confusing, so in this post I'll explain how precision and recall are used and how they are calculated. It's actually quite simple! But first, let's start with a quick recap of precision and recall for binary classification. (There's also Part II: the F1-score, but I recommend you start with Part I).

In binary classification we usually have two classes, often called Positive and Negative, and we try to predict the class for each sample. Let's look at a simple example: our data is a set of images, some of which contain a dog. We are interested in detecting photos with dogs. In this case, our Positive class is the class of all photos of dogs and the Negative class includes all the other photos. In other words, if a sample photo contains a dog, it is a Positive. If it does not, it is a Negative. Our classifier predicts, for each photo, if it is Positive (P) or Negative (N): is there a dog in the photo?

Given a classifier, I find that the best way to think about

What is more important, precision or recall? This really depends on your specific classification problem. Imagine, for example, that your classifier needs to detect diabetes in human patients. “Positive” means the patient has diabetes. “Negative” means that the patient is healthy. (I know, it’s confusing. But that’s medical lingo!). In this case, you probably want to make sure that your classifier has high recall, so that as many diabetics as possible are correctly detected. Take another example — say you are building a video recommendation system, and your classifier predicts Positive for a relevant video and Negative for non-relevant video. You want to make sure that almost all of the recommended videos are relevant to the user, so you want high precision. Life is full of trade-offs, and that’s also true of classifiers. There’s usually a trade-off between good precision and good recall. You usually can’t have both.

. . .

Our dog example was a binary classification problem. Binary classification problems often focus on a Positive class which we want to detect. In contrast, in a typical multi-class classification problem, we need to categorize each sample into 1 of N different classes. Going back to our photo example, imagine now that we have a collection of photos. Each photo shows one animal: either a **cat**, a **fish**, or a **hen**. Our classifier needs to predict which animal is shown in each photo. This is a classification problem with N=3 classes.

Let's look at a sample confusion matrix that is produced after classifying 25 photos:

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Similar to our binary case, we can define precision and recall for each of the classes. For example, the *precision* for the Cat class is the number of correctly predicted Cat photos (4) out of all predicted Cat photos (4+3+6=13), which amounts to 4/13=30.8%. So only about a 1/3 of the photos that our predictor classifies as Cat are actually cats!

In Python's scikit-learn library (also known as **sklearn**), you can easily calculate the precision and recall for each class in a multi-class classifier. A convenient function to use here is **sklearn.metrics.classification_report**.

Here is some code that uses our Cat/Fish/Hen example. I first created a list with the true classes of the images (y_true), and the predicted classes (y_pred). Usually y_pred will be generated using the classifier — here I set its values manually to match the confusion matrix.

In line 14, the confusion matrix is printed, and then in line 17 the precision and recall is printed for the three classes.

```
1  from sklearn import metrics
2
3  # Constants
4  C="Cat"
5  F="Fish"
6  H="Hen"
7
8  # True values
9  y_true = [C,C,C,C,C,C, F,F,F,F,F,F,F,F,F, H,H,H,H,H,H,H,H,H]
10 # Predicted values
11 y_pred = [C,C,C,C,H,F, C,C,C,C,C,C,H,H,F,F, C,C,C,H,H,H,H,H,H]
12
13 # Print the confusion matrix
14 print(metrics.confusion_matrix(y_true, y_pred))
15
16 # Print the precision and recall, among other metrics
```

And here is the output. Note the confusion matrix is transposed