Get started Open in app

Follow 574K Followers

You have **2** free member-only stories left this month. Sign up for Medium and get an extra one

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

Boaz Shmueli Jul 2, 2019 · 7 min read ★

Performance measures 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 classifier performance is by using the so-called "confusion matrix". For binary classification, a confusion matrix has two rows and two columns, and shows how many Positive samples were predicted as Positive or Negative (the first column), and how many Negative photos were predicted as Positive or Negative (the second column). Thus, it has a total of 4 cells. Every time our classifier makes a prediction, one of the cells in the table is incremented by one. By

Get started Open in app

Here is a simple example. Let's assume we have 10 photos, and exactly 7 of them have dogs. If our classifier was perfect, the confusion matrix would look like this:

		True/Actual	
		Positive ( )	Negative
Predicted	Positive ( )	7	0
	Negative	0	3

Our perfect classifier did not make any errors. All positive photos were classified as Positive, and all negative photos were classified as Negative.

In the real world, however, classifiers make errors. A binary classifier makes two kind of errors: some Positive samples are classified as Negative; and some Negative samples are classified as Positive. Let's look at a confusion matrix from a more realistic classifier:

		True/Actual	
		Positive ( )	Negative
Predicted	Positive ( )	5	1
	Negative	2	2

In this example, 2 photos with dogs were classified as Negative (no dog!), and 1 photo without a dog was classified as Positive (dog!).

When a Positive sample is *falsely* classified as Negative, we call this a False Negative (FN). And similarly, when a Negative sample is *falsely* classified as a Positive, it is called

Get started Open in app

		True/Actual	
		Positive ( )	Negative
Predicted	Positive ( )	5 (TP)	1 (FP)
	Negative	2 (FN)	2 (TN)

Now that we have gotten a handle on the confusion matrix and the various numbers, we can start looking at performance metrics: How good is our classifier? (In the back of our mind we always need to remember that "good" can mean different things, depending on the actual real-world problem that we need to solve.)

Let's start with *precision*, which answers the following question: what proportion of **predicted Positives** is truly Positive? We need to look at the total number of predicted Positives (the True Positives plus the False Positives, TP+FP), and see how many of them are True Positive (TP). In our case, 5+1=6 photos were predicted as Positive, but only 5 of them are True Positives. The precision in our case is thus 5/(5+1)=**83.3%**. In general, precision is TP/(TP+FP). Note that TP+FP is the sum of the **first row**.

Another very useful measure is *recall*, which answers a different question: what proportion of **actual Positives** is correctly classified? Looking at the table, we see that the number of actual Positives is 2+5=7 (TP+FN). Out of these 7 photos, 5 were predicted as Positive. The recall is thus 5/7 = **71.4%**. In general, recall is TP/(TP+FN). Note that TP+FN is the sum of the **first column**.

One might also be interested in *accuracy*: what proportion of photos — both Positive and Negative — were correctly classified? In our case, 5+2=7 of the photos were correctly classified out of a total of 10. Thus, the accuracy is 70.0%. Generally speaking, there are a total of TP+TN correctly classified photos out of TP+TN+FP+FN photos, and so the general formula for accuracy is (TP+TN)/(TP+TN+FP+FN).

Get started Open in app

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

Get started Open in app

actually cats!

		True/Actual		
		Cat	Fish	Hen
Predicted	Cat	4	6	3
	Fish	1	2	0
	Hen	1	2	6

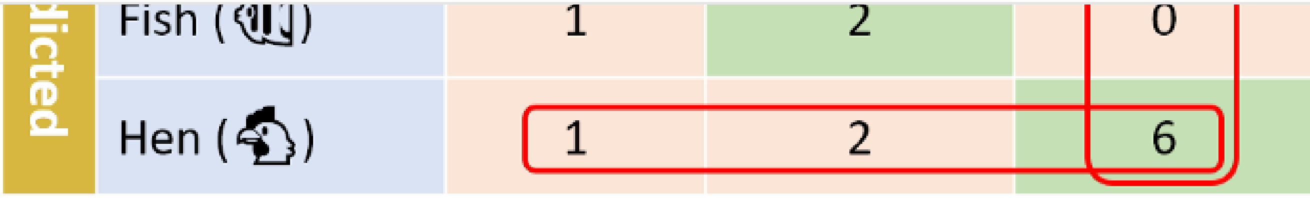
On the other hand, the *recall* for Cat is the number of correctly predicted Cat photos (4) out of the number of actual Cat photos (4+1+1=6), which is 4/6=66.7%. This means that our classifier classified 2/3 of the cat photos as Cat.

In a similar way, we can calculate the precision and recall for the other two classes: Fish and Hen. For Fish the numbers are 66.7% and 20.0% respectively. For Hen the number for both precision and recall is 66.7%. Go ahead and verify these results. You can use the two images below to help you.

		True/Actual		
		Cat	Fish	Hen
Predicted	Cat	4	6	3
	Fish	1	2	0
	Hen	1	2	6

		True/Actual		
		Cat	Fish	Hen

Get started Open in app



Fish	1	2	0
Hen	1	2	6

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,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
17 print(metrics.classification_report(y_true, y_pred, digits=3))
```

mc_metrics_part1.py hosted with ♡ by GitHub view raw

Get started Open in app

class (6 for Cat, 10 for Fish, etc).

```
[[4 1 1]
 [6 2 2]
 [3 0 6]]

              precision    recall  f1-score   support

         Cat      0.308     0.667     0.421         6
        Fish      0.667     0.200     0.308        10
         Hen      0.667     0.667     0.667         9

    accuracy                          0.480        25
   macro avg      0.547     0.511     0.465        25
weighted avg      0.581     0.480     0.464        25
```

The **classification_report** also reports other metrics (for example, F1-score). In an upcoming post, I'll explain **F1-score** for the multi-class case, and why you SHOULDN'T use it :)

Hope you found this post useful and easy to understand!

Continue to Part II: the F1-Score

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

Get this newsletter

Machine Learning Measurement Python Sklearn Classification

Get started

Open in app

Get the Medium app

Download on the App Store

GET IT ON Google Play