Machine Learning Evalutation Metrics

Lucas Paiolla Forastiere

12 de fevereiro de 2021

Sumário

1 Simple evaluation metrics

In **classification**, the most common metric is the **accuracy**:

$$E = \frac{\text{Right predictions}}{\text{Total predictions}}$$

In **regression**, the most common metric is the R^2 **score**:

$$r^2 = 1 - \frac{\Sigma(y_i - \hat{y}_i)^2}{\Sigma(y_i - \overline{y})^2}$$

However, there are many better evaluation metrics, and we're going to discuss some of them here.

There could be many evaluation methods, but at the end, it depends on what you want to do with the model you're creating.

Before proceding, let's understand why accuracy might not be the best metric.

One example is when we have **imbalanced classes**, which is when one of the classes has lots more samples when the other classes. For instance, if we're trying to predict if a credit card transaction is fraudulent, we may have many more non fraudulent examples than fraudulent examples.

In that scenario, let's supose we got an accuracy of 99.9%. That seens to be a amazingly good classifier, but perhaps that 0.1% os failness happens whenever the transaction is fraudulent. We could have, for instance, a dummy model that always predicts *non-fraudulent* without even looking at the features! That ridiculous model would have a very high accuracy, because most examples indeed are *non-fraudulent*, but it will fail every single that we have a *fraudulent* example, and that awful!

2 Confusion Matrices

When we have *binary classifiers*, confusion matrices are very useful because they allow us to visualize better how the model is making mistakes. It consists in dividing the prediction classes and actual classes into a 2×2 matrix where the first row indicates when the model predicted the positive class and the second the negative one, and the first column is all the actual positive class and the second is the nevative one.

		Actual classes	
		1	0
Predicted classes	1	True positive	False positive
	0	False negative	True negative

By doing so, we divide the dataset into the cases when the model predicted positive and that was right (*true positive*); when the model predicted negative and that was right (*true negative*); when the model predicted positive and that was wrong (*false positive*); and when the model predicted negative and that was wrong (*false negative*).

Depending on our application, we might want to minimize false negatives and don't care mush about false positives or vice-versa.

2.1 Basic Evaluation Metrics

We can also notice that the accuracy value is a summary of the table:

$$E = \frac{TN + TP}{TN + TP + FN + FP}$$

Some other evaluation metrics we can define are **recall** and **precision**:

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

The better the recall, the lower the number of *false negatives* and the better the precision, the lower the number of *false positives*.

Therefore, we can use these values when avoiding whether *false negatives* or *false positives*.

Recall is also known as *True Positive Rate* (TPR) and there's also a *False Positive Rate* (FPR):

$$FPR = \frac{FP}{TN + FP}$$

And we want that number to be the closest to zero as possible.

Having two (Recall and Precision) or even three (FPR) numbers is not ideal when comparing two machine learning models. Sometimes, I'd better to have a simple number such tal we could use to point at a model and say "that's the better".

The **F1-score** is a combination of precision and recall:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} == \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

This score is a specify case of a more general precision metric called the **F-score**, which adds a parameter β:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta \cdot FN + FP}$$

where we can set β to emphasis recall or precision:

- $0 < \beta < 1$ emphasizes precision;
- $1 < \beta$ emphasizes recall;

3 Multi-class evalutations

Many multi-class evaluation metrics are just a generalization of the binary ones. For instance, we can generalize the concept of the confusion matrix easy: the columns will still be the actual classes and the rows, the predicted classes, but now it will be a $n \times n$ matrix, where n is the number of classes. The correct predictions will still be the main diagonal and the wrong ones will be off that diagonal.

Confusion matrices are still very good, because they enable us to visualize where the model is making more mistakes. Suppose we are creating a NN to predict written digits. Then we can visualize what digits the NN is missing the more (for instance, it might confuse the number eight with the number zero).

We can also compute precision and recall, but we have two ways to do this.

The first one is called **macro-average**, and it's when we give each class the same weight. And the second one is called **micro-average**, and it's when we give each instance the same weight.

To compute *macro-average* for recall (and same thing for precision), we compute the recall value of each class individualy and then average these values. For instance, supose we had 5 oranges in our test set and only 1 of them was predicted correctly. Therefore, we have a recall of 0.2, because we have just one true possitive and four false negative. Supose we also have a lemon and apple class with respectively recall of 0.5 and 1.0. Therefore, to compute the *macro-average recall*, we average those three values obtaining 0.57.

Suppose now we want the *micro-average*. Then, we just look at each single instance. Suppose we had 2 lemons and 2 apples in our test set (and there, 1 of the lemons and 2 of the apples were predicted right). Therefore, we have 1 orange + 1 lemon + 2 apples right predicted out of $5 + 2 + 2 = 9$ fruits. Therefore, our *micro-average recall* is equal to $4/9 = 0.44$.

If the classes have about the same number of instances, macro and micro average will be about the same, but if some classes are much larger than others, we can use micro or macro depending on what we need. If we want to weight our metric toward the largest ones, we use micro-averaging, but if we want to weight our matric toward the smallest ones, we use macro-averaging.

If the micro-average is much lower than the macro-average, then we should examine the larger classes to look for poor performance. And if the macro-average is much lower, then we do the opposite: look at the smaller classes.

4 Regression Evaluation

In regression, we can't simply *count* the number of errors, because there are no classes, so basicly our model is *always* getting wrong. What we can, however, is compute some metrics like the distance from the target point and the predicted point (like the R^2 score).

We could also differentiate the errors in types like what we do in classification, but it turns out this isn't important on those problems.

For many cases, the typical R^2 score is quite good to analyse the model, but some times you might want another evaluation metric like `mean_absolute_error` (absolute difference of target and predicted values), `mean_squared_error` (squared difference of target and predicted values) or `median_absolute_error` (which is robust to outliners).