CLASSIFICATION METRICS GUIDE

Accuracy vs. precision vs. recall in machine learning: what's the difference?

Last updated: October 1, 2024

A classification model aims to assign a pre-defined label to the objects in the input data. For example, you might want to predict if a user will stop using a certain software product. You will then create an ML model that classifies all users into "churner" or "non-churner" categories.

Once you know the actual labels (did the user churn or not?), you can measure the classification model quality metrics such as accuracy, precision, and recall.

Let's take a deep dive into each metric and explore them in detail!

What is accuracy?

Accuracy is a metric that measures how often a machine learning model correctly predicts the outcome. You can calculate accuracy by dividing the number of correct predictions by the total number of predictions.

> In other words, accuracy answers the question: how often the model is right?

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{All predictions}}$$

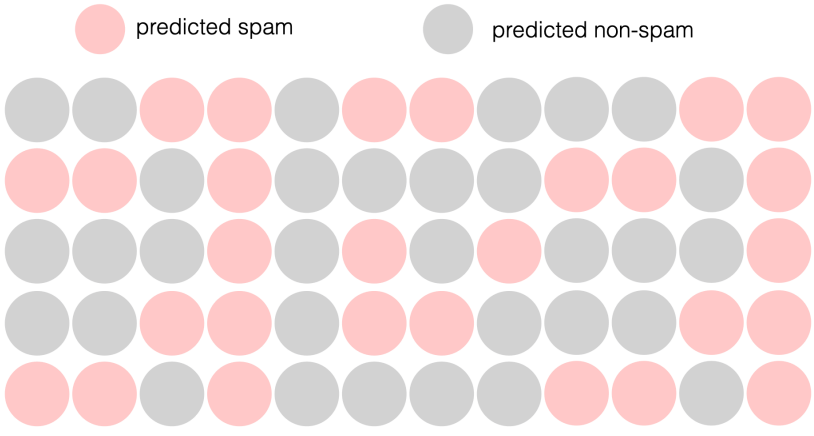
You can measure the accuracy on a scale of 0 to 1 or as a percentage. The higher the accuracy, the better. You can achieve a perfect accuracy of 1.0 when every prediction the model makes is correct.

This metric is simple to calculate and understand. Almost everyone has an intuitive perception of accuracy: a reflection of the model's ability to correctly classify data points.

Visual example

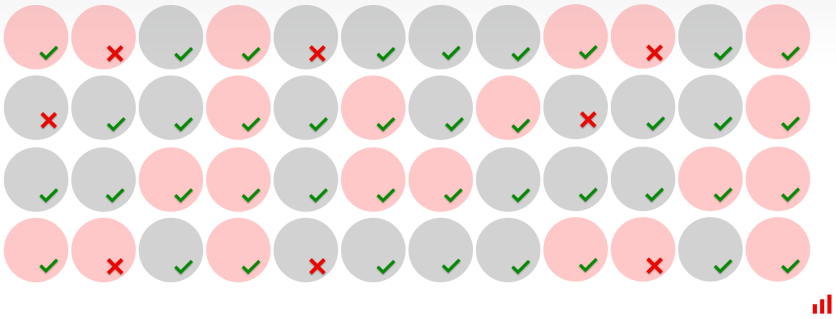
Let's say we have a machine learning model that performs spam detection.

For every email in the dataset, it produced a prediction and assigned one of the two classes: "spam" or "not spam." Here is how we can visualize the **predicted** labels:



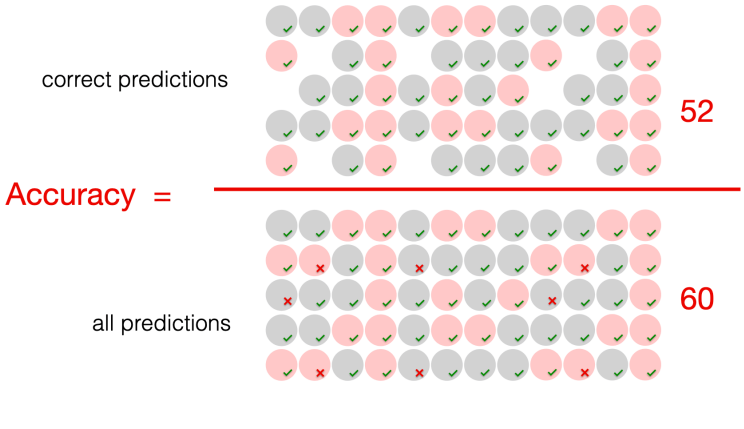
After we get the **actual** labels (learn which emails are spam and which are not), we can evaluate if the model predictions are correct.

We can visually tag each correct and wrong prediction.



Now, you can simply count the number of times the model was right and divide it by the total number of predictions.

In our case, 52 out of 60 predictions (labeled with a green "tick" sign) were correct. Meaning the model accuracy is 87%.



On the surface, accuracy is a straightforward way to judge model quality. It shows the overall correctness of the model and is easy to communicate.

The truth is this measure is not always useful. Enter the accuracy paradox.

Accuracy paradox

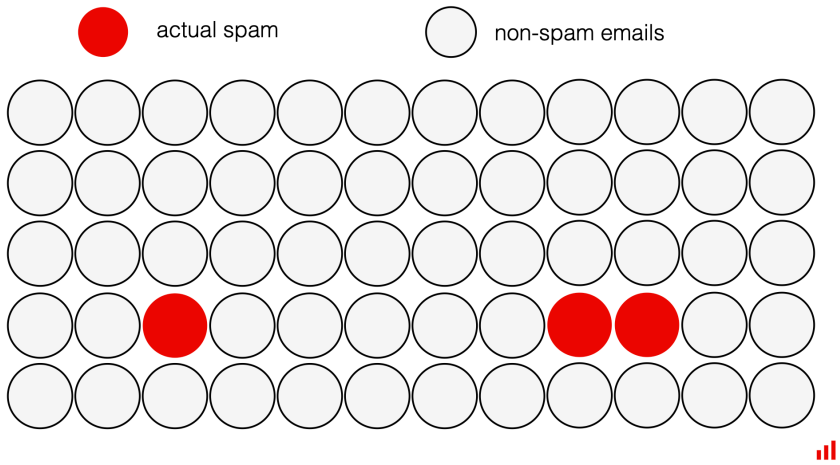
There is a downside to focusing on accuracy as a primary metric. The reason is that it treats all classes as equally important and looks at all correct predictions.

In the example above, there are two classes, and the dataset is reasonably balanced. We have multiple cases of both spam and non-spam emails. In these cases, accuracy can be a suitable and helpful metric. For example, this might happen when you classify the

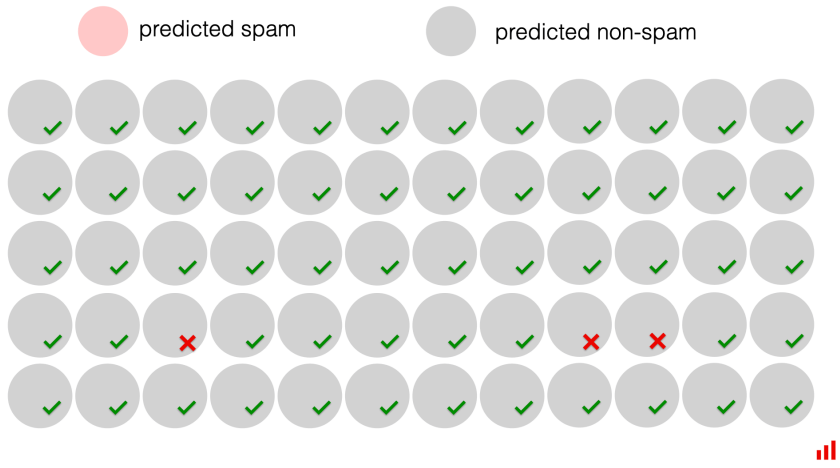
classes. These are the cases when one category has significantly more frequent occurrences than the other.

For example, this might happen when you are predicting payment fraud, equipment failures, users churn, or identifying illness on a set of X-ray images. In scenarios like this, you are typically interested in predicting the **events that rarely occur.** They might only happen in 1-10% of cases or even less frequently.

Let's illustrate the idea. Imagine that the actual balance of spam and non-spam emails looks like this. Out of 60 emails, only 3 (or 5%) are truly spam.



It is easy to "game" the accuracy metric when making predictions for a dataset like this. To do that, you simply need to predict that nothing will happen and **label every email as non-spam**. The model predicting the majority (non-spam) class all the time will mostly be right, leading to very high accuracy.



In this specific example, the accuracy is 95%: yes, the model missed every spam email, but it was still right in 57 cases out of 60.

Pros and cons

Let's sum up the accuracy metric!

Pros:

- Accuracy is a helpful metric when you deal with **balanced classes** and care about the overall model "correctness" and not the ability to predict a specific class.
- Accuracy is **easy to explain** and communicate.

Cons:

- If you have **imbalanced classes**, accuracy is less useful since it gives equal weight to the model's ability to predict all categories.
- Communicating accuracy in such cases **can be misleading** and disguise low performance on the target class.

When you see an imbalanced example like the spam example above, it is very intuitive to suggest a different approach to model evaluation that overcomes the limitation of accuracy: we do not need the "overall" correctness, we want to find spam emails after all! Can we focus on how well we find and detect them specifically?

Precision and **recall** are the two metrics that help with that.

Before we explain the two metrics, let's refresh the idea of a confusion matrix and the different types of errors a classification model can make.

> **Accuracy, precision, and recall for multi-class classification.** In this chapter, we focus on binary classification examples. You can read about how to calculate accuracy, precision, and recall for multi-class classification in a separate article.

Confusion matrix

In binary classification, there are two possible target classes, which are typically labeled as "positive" and "negative" or "1" and "0". In our spam example above, the target (positive class) is "spam," and the negative class is "not spam."

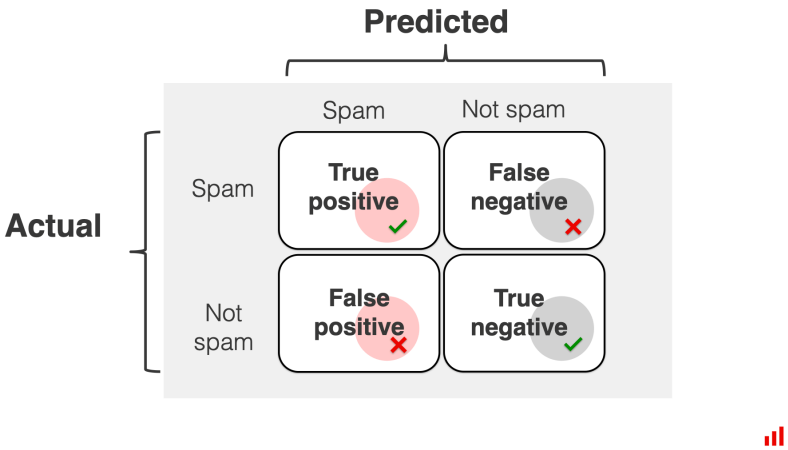
Correct predictions include so-called true positives and true negatives. This is how it unpacks for our spam use case example:

- **True negative (TN):** An email that is actually not spam and is correctly classified by the model as not spam.

Model errors include so-called false positives and false negatives. In our example:

- **False Positive (FP):** An email that is actually not spam but is incorrectly classified by the model as spam (a "false alarm").
- **False Negative (FN):** An email that is actually spam but is incorrectly classified by the model as not spam (a "missed spam").

Using the **confusion matrix**, you can visualize all 4 different outcomes in a single table.



To better understand how the Confusion Matrix works, you can explore a detailed guide in a separate chapter.

Considering these different ways of being right and wrong, we can now extend the accuracy formula. Correct predictions in the numerator include both true positives and negatives. All predictions in the denominator include all true and false predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

With a naive model that predicts “not spam” all the time, the model can identify True Negatives (thus inflating the overall Accuracy metric value) – but not the True Positives.

Because of how it is constructed, accuracy ignores the specific types of errors the model makes. It focuses on “being right overall.” To evaluate how well the model deals with identifying and predicting True Positives, we should measure **precision** and **recall** instead.

What is precision?

Precision is a metric that measures how often a machine learning model correctly predicts the positive class. You can calculate precision by dividing the number of correct positive predictions (true positives) by the total number of instances the model predicted as positive (both true and false positives).

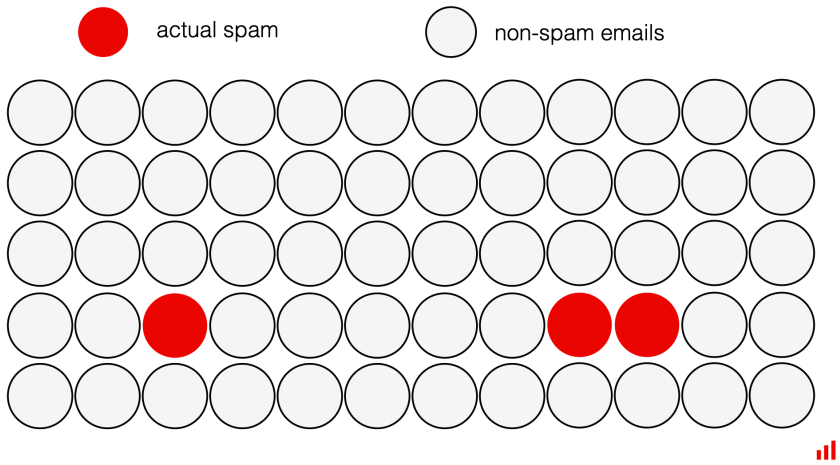
In other words, precision answers the question: how often the positive predictions are correct?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

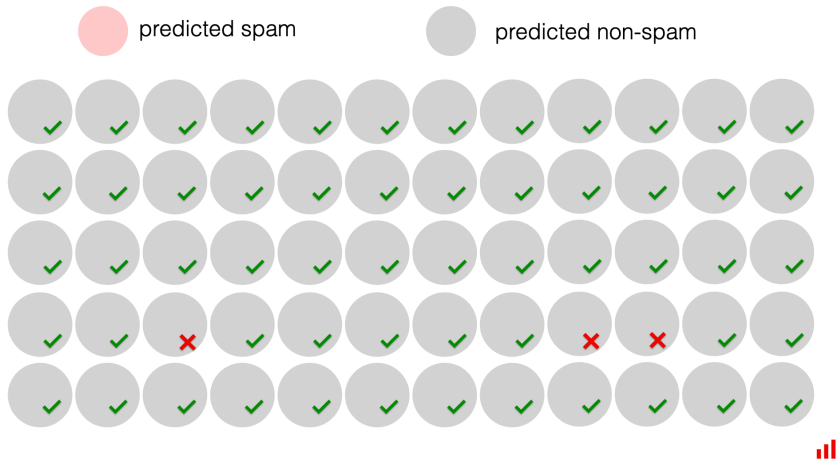
Visual example

Let's go back to our fictional example with an imbalanced problem, such as spam occurring only in 5% of all emails.

Here is the refresher on how the true labels (the actual split between spam and non-spam emails) look in this case:



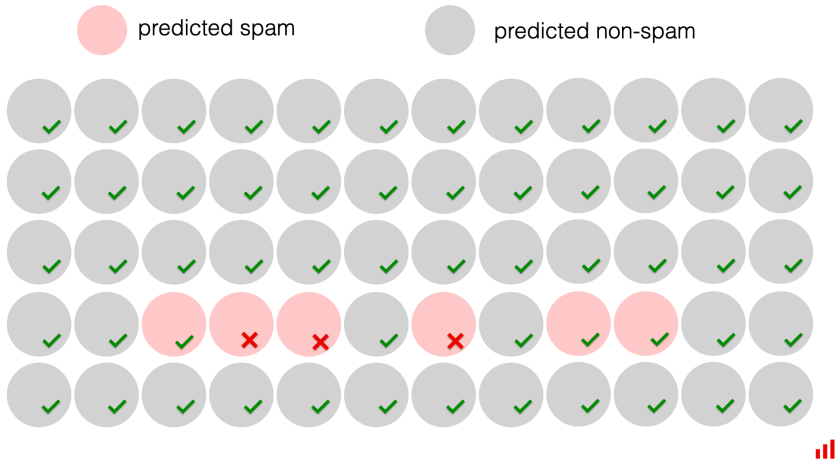
Let's first try the same trick that we did with accuracy. We simply predict that all emails are non-spam.



The already know that the **accuracy** is a whopping 95% (the model is right in 57 cases out of 60), but the **precision** is 0!

To calculate the precision, we must divide the number of **correctly predicted spam emails** by their total number. However, the number of correctly identified spam emails is 0. There were 3 spam emails in the dataset, and the model missed them all. All the correct predictions were about non-spam emails.

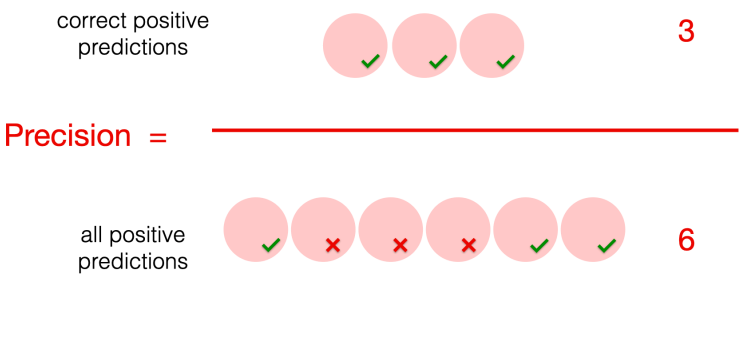
Let's now imagine we **actually trained a reasonable ML model.** It does identify some of the emails as spam. We then compare the model predictions against the true labels and get the following results:



Now, let's look at the model quality once again.

Model accuracy is 95%. The model correctly labeled 57 out of 60 emails (and made 3 errors). This is exactly the same accuracy level as we got with the model that simply labeled every email as non-spam. However, we already learned that accuracy is not a helpful metric in this case. The fact that our new (actually useful) model has the same accuracy as the model that predicts nothing valuable illustrates that once again.

Now, how do we calculate the precision? We must divide correctly predicted spam emails (3) by the total number of positive predictions (6).



The precision is 50%. The model labeled 6 emails as spam and was right half the time. 3 out of 6 emails labeled as spam were, in fact,

Pros and cons

When is precision a suitable metric to focus on?

Here are the pros of the precision metric:

- It works well for problems with **imbalanced classes** since it shows the model correctness in identifying the target class.
- Precision is useful when the **cost of a false positive** is high. In this case, you typically want to be **confident in identifying the target class,** even if you miss out on some (or many) instances.

To illustrate it, let's continue with the spam detection example.

Say, as a product manager of the spam detection feature, you decide that cost of a false positive error is high. You can interpret the error cost as a negative user experience due to misprediction. You want to ensure that the user never misses an important email because it is incorrectly labeled as spam. As a result, you want to **minimize false positive errors.**

In this case, **precision** is a good metric to evaluate and optimize for. A higher precision score indicates that the model makes fewer false positive predictions. It is more likely to be correct whenever it predicts a positive outcome.

Like every optimization, it comes at a cost. In this case, some spam emails might be left undetected and will make their way into the users' inboxes. However, you might decide the cost of this type of error (false negative) is more tolerable. Users can still flag these emails as spam manually. However, whenever you automatically put emails in the spam folder and hide them from the user, they better be genuinely spam.

Of course, the precision has its downsides too.

Here is the main **con of precision:**

- Precision does not consider **false negatives.** Meaning: it does not account for the cases when we miss our target event!

Just like shown in the example above, precision disregards the fact that we might miss out on some spam emails.

This is why you need one more metric to balance precision: recall.

What is recall?

instances. The latter includes true positives (successfully identified cases) and false negative results (missed cases).

In other words, recall answers the question: can an ML model find all instances of the positive class?

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

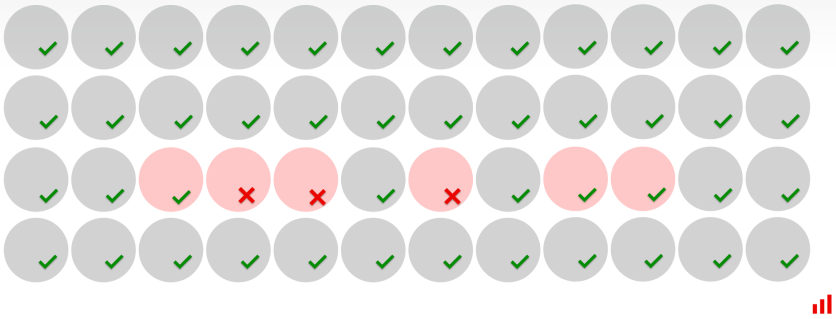
You can measure the recall on a scale of 0 to 1 or as a percentage. The higher the recall, the better. You can achieve a perfect recall of 1.0 when the model can find all instances of the target class in the dataset.

Recall can also be called **sensitivity** or **true positive rate**. The term "sensitivity" is more commonly used in medical and biological research rather than machine learning. For example, you can refer to the sensitivity of a diagnostic medical test to explain its ability to expose the majority of true positive cases correctly. The concept is the same, but “recall” is a more common term in machine learning.

Visual example

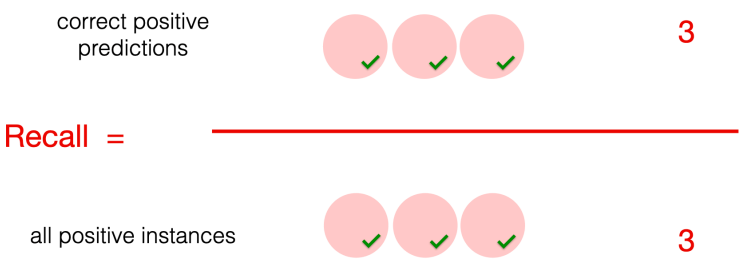
Let’s stick with the example we used to explain precision.

We are predicting spam emails. The problem is unbalanced, and only 3 real spam emails are in the dataset. We trained a model, applied it to our validation dataset, and compared the predictions against the true labels with the following results:



We already calculated the **model accuracy** as 95% (the model correctly labeled 57 out of 60 emails) and **model precision** as 50% (the model correctly labeled 3 out of 6 spam emails).

What about the **recall**? To calculate the recall, we must divide the number of discovered spam emails by the total number of spam emails in the dataset.



The recall is 100%. There were 3 spam emails in the dataset, and the model found all of them! We calculate it as 3/(3+0). There were no false negatives since the model did not miss spam.

This way, recall shows yet another dimension of the model quality. All in all, this fictional model has 95% accuracy, 50% precision, and 100% recall. Depending on your goals, this might be a great (or not-so-great) outcome! When should you optimize for recall?

Pros and cons

Here are the pros of the recall metric:

- It works well for problems with **imbalanced classes** since it is focused on the model's ability to find objects of the target class.

For example, if an ML model points to possible medical conditions, detects dangerous objects in security screening, or alarms to potentially expensive fraud, missing out might be very expensive. In this scenario, you might prefer to be overly cautious and manually review more instances the model flags as suspicious.

In other words, you would treat **false negative errors** as more costly than false positives. If that is the case, you can **optimize for recall** and consider it the primary metric.

Here is the **downside of recall:**

- Recall is that it does not account for the cost of these **false positives**.

The costs of false positives might still be substantial. In extreme cases, they can make the model useless if you have to review too many decisions and the precision is low.

The corner case for the recall is the opposite of the accuracy paradox: if you want to get "total recall," you can flag every single object as suspicious. For example, you can flag every email as spam. This way, you achieve 100% recall (you will detect all objects of the target class), but an overwhelming majority of predictions will be false positives, making the model useless.

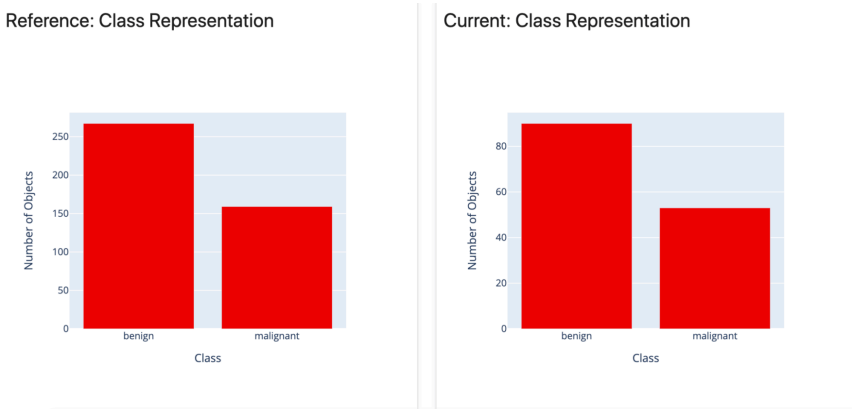
No single metric is perfect. Because of this, it makes sense to look at multiple metrics simultaneously and define the right balance between precision and recall.

> **Want a more detailed example of evaluating classification models?** Head to a code tutorial that shows how to compare the performance of two classification models that look similar on the surface but have different performance characteristics.

Practical considerations

Accuracy, precision, and recall are all important metrics to evaluate the performance of an ML model. Since none reflects the "absolute best" way to measure the model quality, you would typically need to look at them jointly, or consciously choose the one more suitable for your specific scenario.

recall, and accuracy, it makes sense to evaluate the proportion of classes and remember how each metric behaves when dealing with imbalanced classes. Some metrics (like accuracy) can look misleadingly good and disguise the performance of important minority classes.



Example class representation plot generated using Evidently Python library.

Mind the cost of error. You typically can balance precision and recall depending on the specific goals of your project.

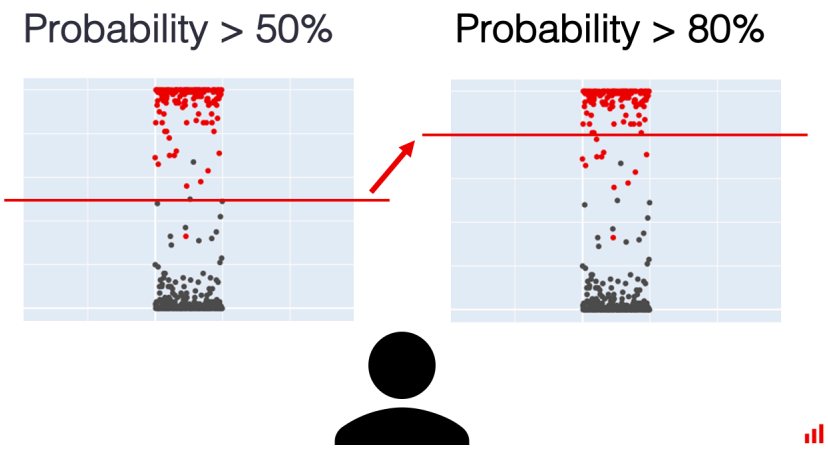
> Precision is a suitable metric when you care more about "being right" when assigning the positive class than "detecting them all." The recall metric is the opposite.

To understand which metric to prioritize, you can assign a specific cost to each type of error. Sometimes, these costs can be directly calculated or modeled.

For example, in churn prediction, you can measure the cost of false negatives (i.e., failing to identify a customer who is likely to churn) as the lost revenue from this customer. You can measure the cost of false positives (i.e., incorrectly identifying a customer as likely to churn when they are not) as the cost of marketing incentives, such as discounts to retain the customer.

Similarly, you can come up with cost estimations for each type of error in other applications. For example, in financial fraud detection, you can weigh the potential financial and reputation losses against the cost of investigation and customer dissatisfaction. In manufacturing quality control, you can evaluate the downstream costs of missing a defective product against the cost of manual inspection, and so on.

Mind the decision threshold. Another way to navigate the right balance between precision and recall is by manually setting a different decision threshold for probabilistic classification.



The decision threshold is the value above which input is classified as belonging to a particular class and below which it is classified as belonging to a different class. For example, you can assign predictions to a specific class when the predicted probability is 0.5 or move it to 0.8.

If the goal is to minimize false positives (maximize precision), then a higher decision threshold may be more appropriate. On the other hand, if the goal is to minimize false negatives (maximize recall), then a lower decision threshold may be more appropriate.

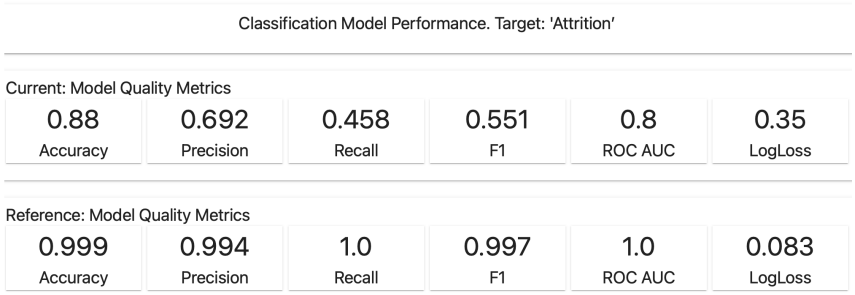
> You can read more about setting classification decision thresholds in a separate guide.

By considering accuracy, precision, recall, and the cost of errors, you can make more nuanced decisions about the performance of ML models on the specific application.

Ultimately, the choice of metric depends on the particular task and the trade-offs between different types of errors.

> **Want to see a real example with data and code?** Here is a tutorial on the employee churn prediction problem "What is your model hiding?". You will train two different classification

Accuracy, precision, and recall in ML monitoring



Example metrics summary generated using Evidently Python library.

Since accuracy, precision, and recall are numerical measurements, you can conveniently use them to **track the model quality over time**. The only major limitation is the need for **true labels** in production.

The feedback loop might be very fast in some use cases, like online personalization in e-commerce. For example, immediately after showing the promotional offer to the user during check-out, you will know if the user clicked on it and accepted the offer. In this case, you can compute quality metrics with a short delay.

In other cases, you might need to wait days, weeks, or even months to know if the model predictions were correct. In this case, you can only retroactively calculate accuracy, precision, or recall for the past period after you receive the new labels. You can also monitor **proxy metrics** like data drift to detect deviations in the input data which might affect model quality.

In practical applications, it is often advisable to compute the quality metrics for **specific segments**. For example, in cases like churn prediction, you might have multiple groups of customers based on geography, subscription type, usage level, etc. Based on your business priorities, it might make sense to evaluate the model precision and recall separately, for example, for the premium user segment. Focusing on a single overall quality metric might disguise low performance in an important segment.

Accuracy, precision, and recall in Python

You will need to prepare your dataset that includes predicted values for each class and true labels and pass it to the tool. You will instantly get an interactive report that includes a confusion matrix, accuracy, precision, recall metrics, ROC curve and other visualizations. You can also integrate these model quality checks into your production pipelines.

```
classification_performance_report = Report(metrics=[
    ClassificationPreset(probas_threshold=0.7),
])

classification_performance_report.run(current_data=bcancer_cur, reference_data=bcancer_ref)

classification_performance_report
```

Evidently allows calculating various additional Reports and Test Suites for model and data quality. Check out Evidently on GitHub and go through the Getting Started Tutorial.

Get started with AI observability

Try our open-source library with over 20 million downloads, or sign up to Evidently Cloud to run no-code checks and bring all the team to a single workspace to collaborate on AI quality.

Sign up free ⟶

Or try open source ⟶

Evidently AI Team

Read next

Multi-class Precision and Recall

There are different ways to calculate accuracy, precision, and recall for multi-class classification. You can calculate metrics by each class or use macro- or micro-averaging. This chapter explains the difference between the options and how they behave in important corner cases.

Classification Threshold

In probabilistic machine learning problems, the model output is not a label but a score. You must then set a decision threshold to assign a specific label to a prediction. This chapter explains how to choose an optimal classification threshold to balance precision and recall.