

Get started

Open in app



Follow

594K Followers



GETTING STARTED

# Handling imbalanced datasets in machine learning

What should and should not be done when facing an imbalanced classes problem?



Baptiste Rocca Jan 28, 2019 · 15 min read



*This post was co-written with Joseph Rocca.*

## Introduction

Suppose that you are working in a given company and you are asked to create a model that, based on various measurements at your disposal, predicts whether a product is defective or not. You decide to use your favourite classifier, train it on the data and



weeks later he enters your office and underlines the uselessness of your model. Indeed, the model you created has not found any defective product from the time it has been used in production.

After some investigations, you find out that there is only around 3.8% of the product made by your company that are defective and your model just always answers “not defective”, leading to a 96.2% accuracy. The kind of “naive” results you obtained is due to the imbalanced dataset you are working with. The goal of this article is to review the different methods that can be used to tackle classification problems with imbalanced classes.

## Outline

First we will give an overview of different evaluation metrics that can help to detect “naive behaviours”. We will then discuss a whole bunch of methods consisting in reworking the dataset and show that these methods can be misleading. Finally, we will show that reworking the problem is, most of the time, the best way to proceed.

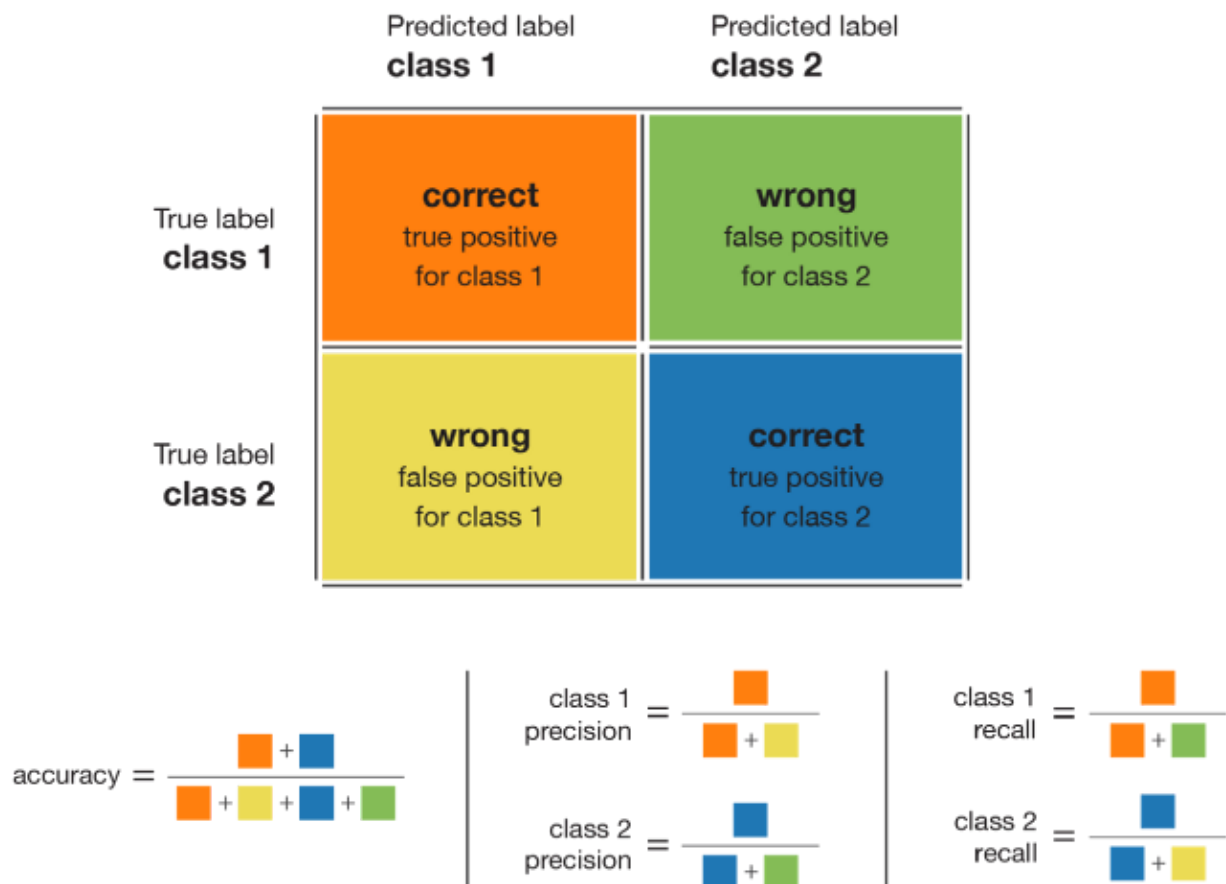
Some subsections indicated by a ( $\infty$ ) symbol contains more mathematical details and can be skipped without impacting the overall comprehension of this post. Notice also that in most of what follows, we will consider two classes classification problems but reasoning can easily be extended to multi-classes cases.

## Detect a “naive behaviour”

In this first section, we would like to remind different ways to evaluate a trained classifier in order to be sure to detect any kind of “naive behaviour”. As we saw in the introduction’s example, accuracy, while being an important and unavoidable metric, can be misleading and therefore should be used cautiously and alongside other metrics. Let’s see what other tools can then be used.

## Confusion matrix, precision, recall and F1

A good and yet simple metric that should always be used when dealing with classification problem is the confusion matrix. This metric gives an interesting overview of how well a model is doing. Thus, it is a great starting point for any



The confusion matrix and the metrics that can be derived from it.

Let us give a short description of these metrics. The accuracy of the model is basically the total number of correct predictions divided by total number of predictions. The precision of a class define how trustable is the result when the model answer that a point belongs to that class. The recall of a class expresses how well the model is able to detect that class. The F1 score of a class is given by the harmonic mean of precision and recall ( $2 \times \text{precision} \times \text{recall} / (\text{precision} + \text{recall})$ ), it combines precision and recall of a class in one metric.

For a given class, the different combinations of recall and precision have the following meanings :

- high recall + high precision : the class is perfectly handled by the model



- high recall + low precision : the class is well detected but the model also include points of other classes in it
- low recall + low precision : the class is poorly handled by the model

In our introductory example, we have the following confusion matrix for 10000 products.



The confusion matrix of our introductory example. Notice that the “defective” precision can’t be computed.

The accuracy is 96.2% as said earlier. The non defective class precision is 96.2% and the defective class precision is not computable. The recall of the non defective class is 1.0 which is perfect (all the non defective products have been labelled as such). But the recall of the defective class is 0.0 which is the worse case (no defective products were detected). Thus, we can conclude our model is not doing well **for this class**. The F1 score is not computable for the defective products and is 0.981 for the non defective products. In this example, looking at the confusion matrix could have led to re-think



## ROC and AUROC

Another interesting metric is the ROC curve (standing for Receiver Operating Characteristic), defined with respect to a given class (that we will denote  $C$  in the following).

Suppose that for a given point  $x$ , we have a model that outputs the probability that this point belongs to  $C$ :  $P(C | x)$ . Based on this probability, we can define a decision rule that consists in saying that  $x$  belongs to class  $C$  if and only if  $P(C | x) \geq T$ , where  $T$  is a given threshold defining our decision rule. If  $T=1$ , a point is labelled as belonging to  $C$  only if the model is 100% confident it does. If  $T=0$ , every points are labelled as belonging to  $C$ .

Each value of the threshold  $T$  generates a point (false positive, true positive) and, then, the ROC curve is the curve described by the ensemble of points generated when  $T$  varies from 1 to 0. This curve starts at point  $(0,0)$ , ends at point  $(1,1)$  and is increasing. A good model will have a curve that increases quickly from 0 to 1 (meaning that only a little precision has to be sacrificed to get a high recall).

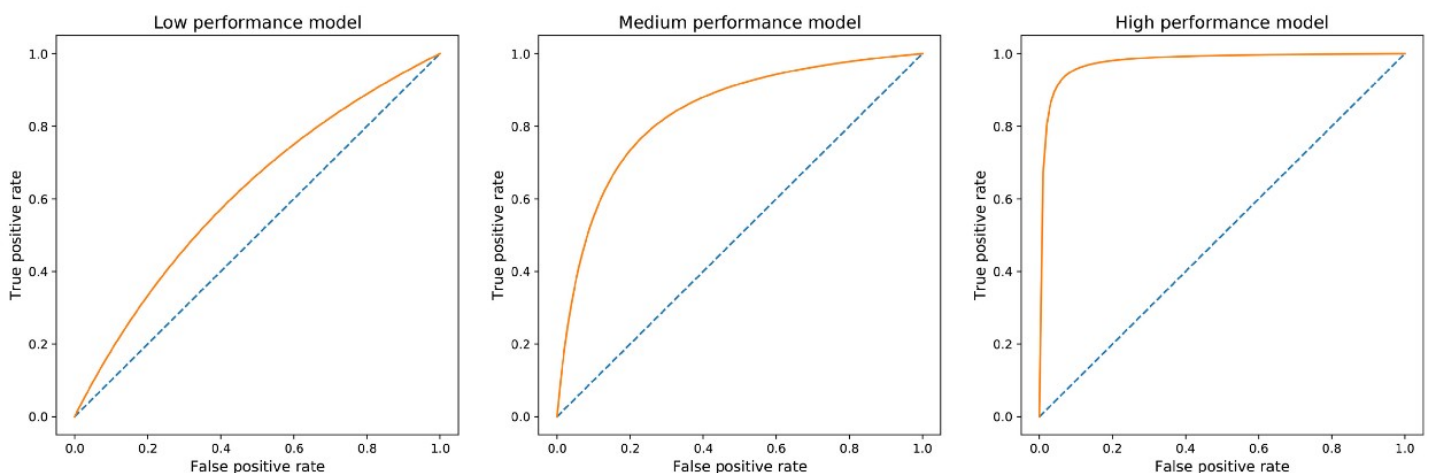


Illustration of possible ROC curves depending on the effectiveness of the model. On the left, the model has to sacrifice a lot of precision to get a high recall. On the right, the model is highly effective: it can reach a high recall while keeping a high precision.

Based on the ROC curve, we can build another metric, easier to use, to evaluate the model: the AUROC which is the Area Under the ROC curve. AUROC acts a little bit as a scalar value that summarises the entire ROC curve. As it can be seen, the AUROC tend towards 1.0 for the best case and towards 0.5 for the worst case.



class).

## What really is the problem?

Before trying to tackle the problem, let's try to better understand it. For this, we are going to consider a very simple example that will allow us both to review quickly some basic aspects of a two classes classification and to better grasp the fundamental problem of imbalanced dataset. This example will also be used in the following sections.

### An imbalanced example

Let's suppose that we have two classes: C0 and C1. Points from the class C0 follow a one dimensional Gaussian distribution of mean 0 and variance 4. Points from the class C1 follow a one dimensional Gaussian distribution of mean 2 and variance 1. Suppose also that in our problem the class C0 represent 90% of the dataset (and, so, the class C1 represent the remaining 10%). In the following picture, we have depicted a representative dataset containing 50 points along with the theoretical distributions of both classes in the right proportions

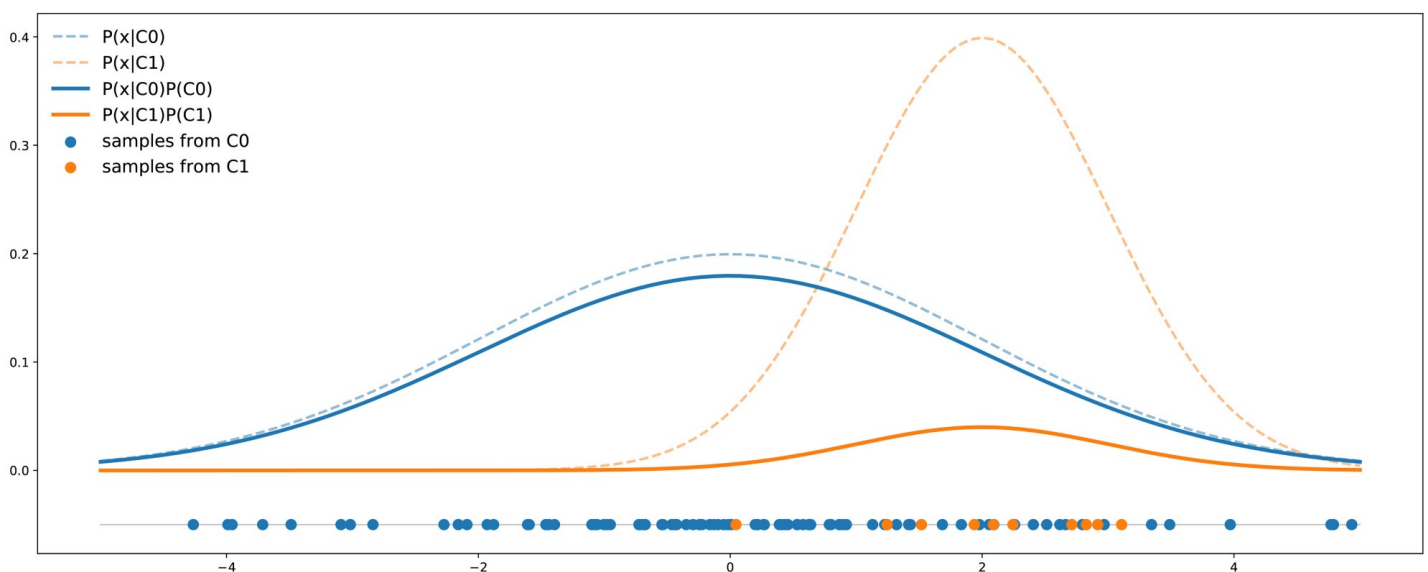


Illustration of our imbalanced example. Dotted lines represent the probability densities of each class independently. Solid lines also take into account the proportions.

In this example we can see that the curve of the C0 class is always above the curve of



Mathematically, using basic Bayes rule, we can write

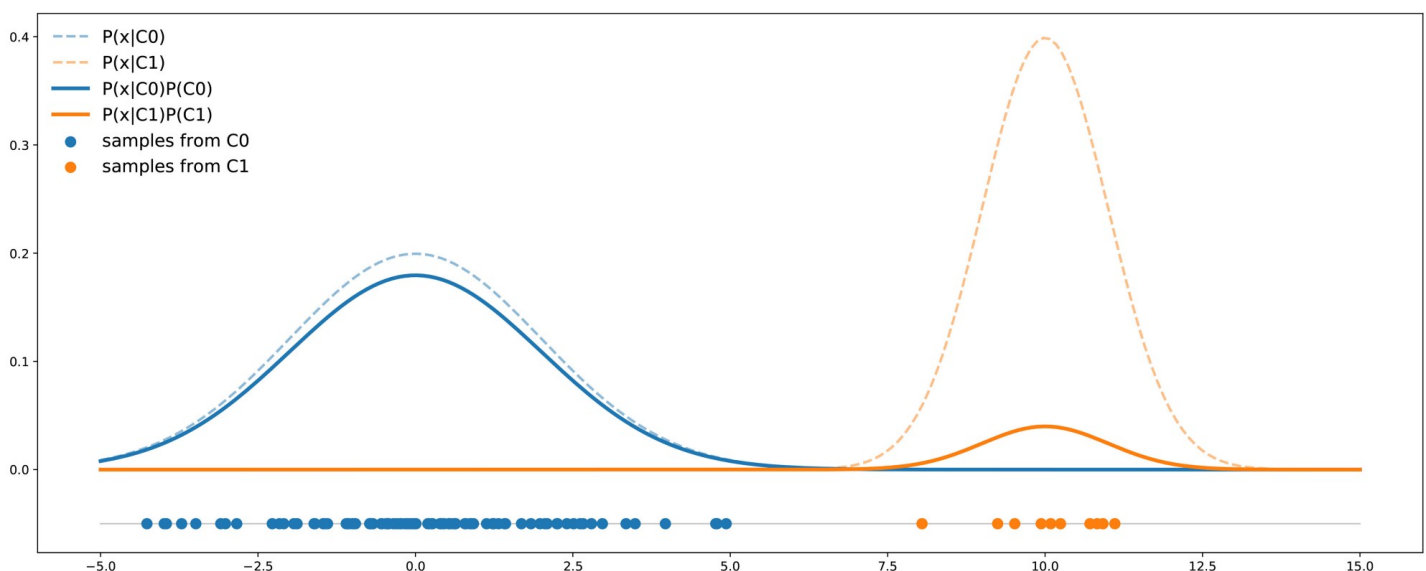
$$\mathbb{P}(C0|x) = \frac{\mathbb{P}(x|C0)\mathbb{P}(C0)}{\mathbb{P}(x)} > \frac{\mathbb{P}(x|C1)\mathbb{P}(C1)}{\mathbb{P}(x)} = \mathbb{P}(C1|x)$$

where we can clearly see the effect of the priors and how it can lead to a situation where a class is always more likely than the other.

All this implies that even from a perfect theoretical point of view we know that if we had to train a classifier on these data, **the accuracy of the classifier would be maximal when always answering C0**. So, if the goal is to train a classifier to get the best possible accuracy then it should not be seen as a problem but just as a fact: with these features, the best we can do (in terms of accuracy) is to always answer C0. We have to accept it.

## About the separability

In the given example, we can observe that the two classes are not well separable (they are not far apart from each other). However, we can notice that facing an imbalanced dataset doesn't necessarily mean that the two classes are not well separable and, so, that the classifier can't do a pretty good job on the minority class. For example, consider that we still have two classes C0 (90%) and C1 (10%). Data in C0 follow a one dimensional Gaussian distribution of mean 0 and variance 4 whereas data in C1 follow a one dimensional Gaussian distribution of mean 10 and variance 1. If we plot the data as before, we then have







Here we see that contrarily to the previous case the  $C_0$  curve is not always above the  $C_1$  curve and, so, there are points that are more likely to be drawn from class  $C_1$  than from class  $C_0$ . In this case, the two classes are separated enough to compensate the imbalance: a classifier will not necessarily answer  $C_0$  all the time.

## Theoretical minimal error probability ( $\infty$ )

Finally, we should keep in mind that a classifier has a theoretical minimal error probability. For a classifier of this kind (one feature, two classes), we can mention that, graphically, the theoretical minimal error probability is given by the area under the minimum of the two curves.

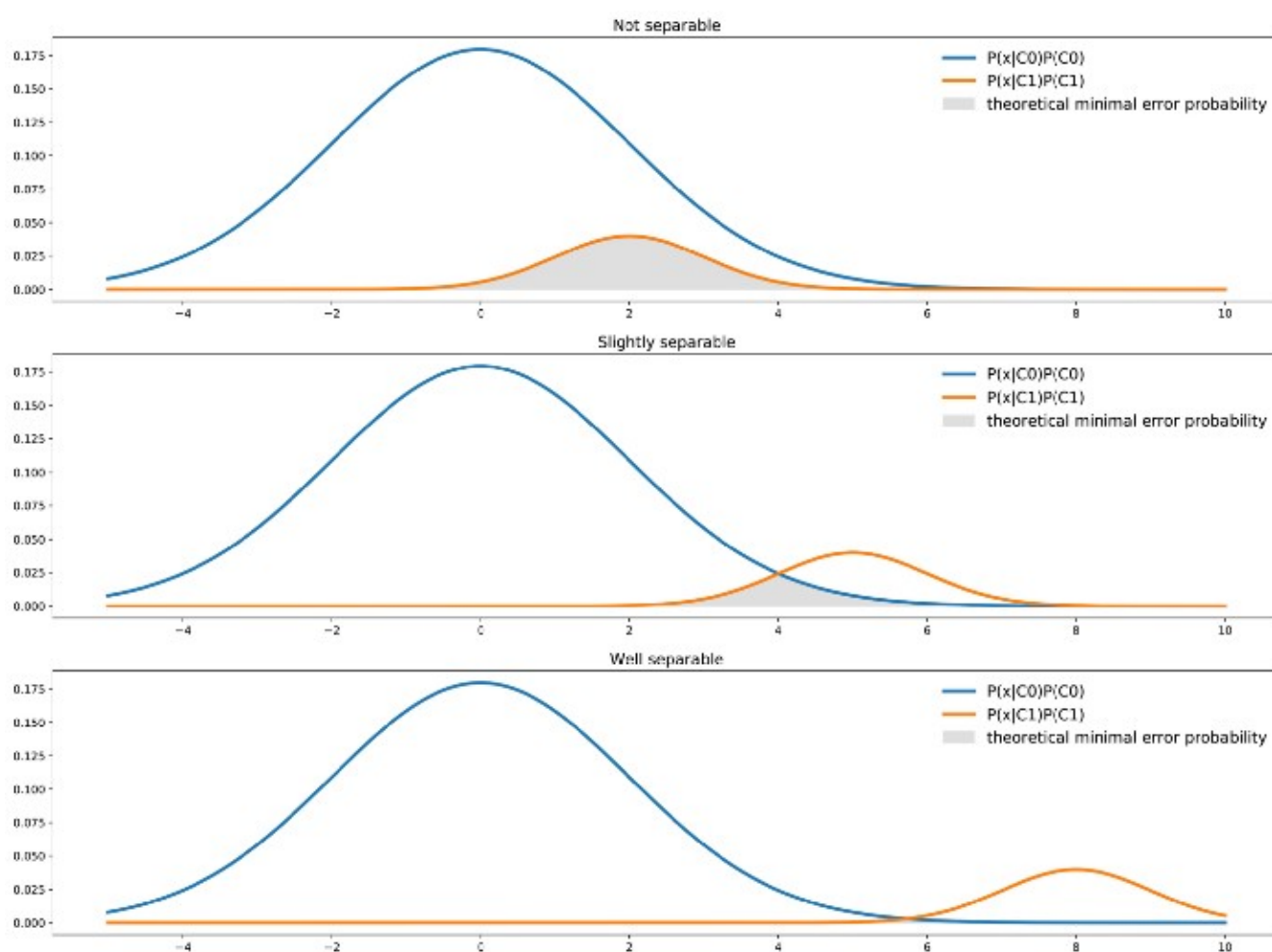


Illustration of the theoretical minimal error for different degree of separability of two classes.

We can recover this intuition mathematically. Indeed, from a theoretical point of view, the best possible classifier will choose for each point  $x$  the most likely of the two classes. It naturally implies that for a given point  $x$ , the best theoretical error probability is given by the less likely of these two classes





Then we can express the overall error probability

$$\mathbb{P}(\text{wrong}) = \int_{\mathbb{R}} \mathbb{P}(\text{wrong}|x)\mathbb{P}(x)dx = \int_{\mathbb{R}} \min(\mathbb{P}(x|C0)\mathbb{P}(C0), \mathbb{P}(x|C1)\mathbb{P}(C1))dx$$

Which is the area under the min of the two curves represented above.

## Reworking the dataset is not always a solution

To begin, the very first possible reaction when facing an imbalanced dataset is to consider that data are not representative of the reality: if so, we assume that real data are almost balanced but that there is a proportions bias (due to the gathering method, for example) in the collected data. In this case, it is almost mandatory to try collecting more representative data.

Let's see, now, what can be done when the dataset is imbalanced because reality is so. In the the next two subsections we present some methods that are often mentioned to tackle imbalanced classes and that deal with the dataset itself. In particular, we discuss the risks related to undersampling, oversampling and generating synthetic data as well as the benefits of getting more features.

## Undersampling, oversampling and generating synthetic data

These methods are often presented as great ways to balance the dataset before fitting a classifier on it. In a few words, these methods act on the dataset as follows:

- undersampling consists in sampling from the majority class in order to keep only a part of these points
- oversampling consists in replicating some points from the minority class in order to increase its cardinality
- generating synthetic data consists in creating new synthetic points from the minority class (see SMOTE method for example) to increase its cardinality

All these approaches aim at rebalancing (partially or fully) the dataset. But should we rebalance the dataset to have as much data of both classes ? Or should the majority



Illustration of the effect that different degrees of majority class undersampling have on the model decisions.

When using a resampling method (for example to get as much data from C0 than from C1), we show the wrong proportions of the two classes to the classifier during the training. The classifier learned this way will then have a lower accuracy on the future real test data than the classifier trained on the unchanged dataset. Indeed, the true proportions of classes are important to know for classifying a new point and that information has been lost when resampling the dataset.

So, if these methods have not to be completely rejected, they should be used cautiously: it can lead to a relevant approach if new proportions are chosen with purpose (we will see that in the next section), but it can also be a nonsense to just

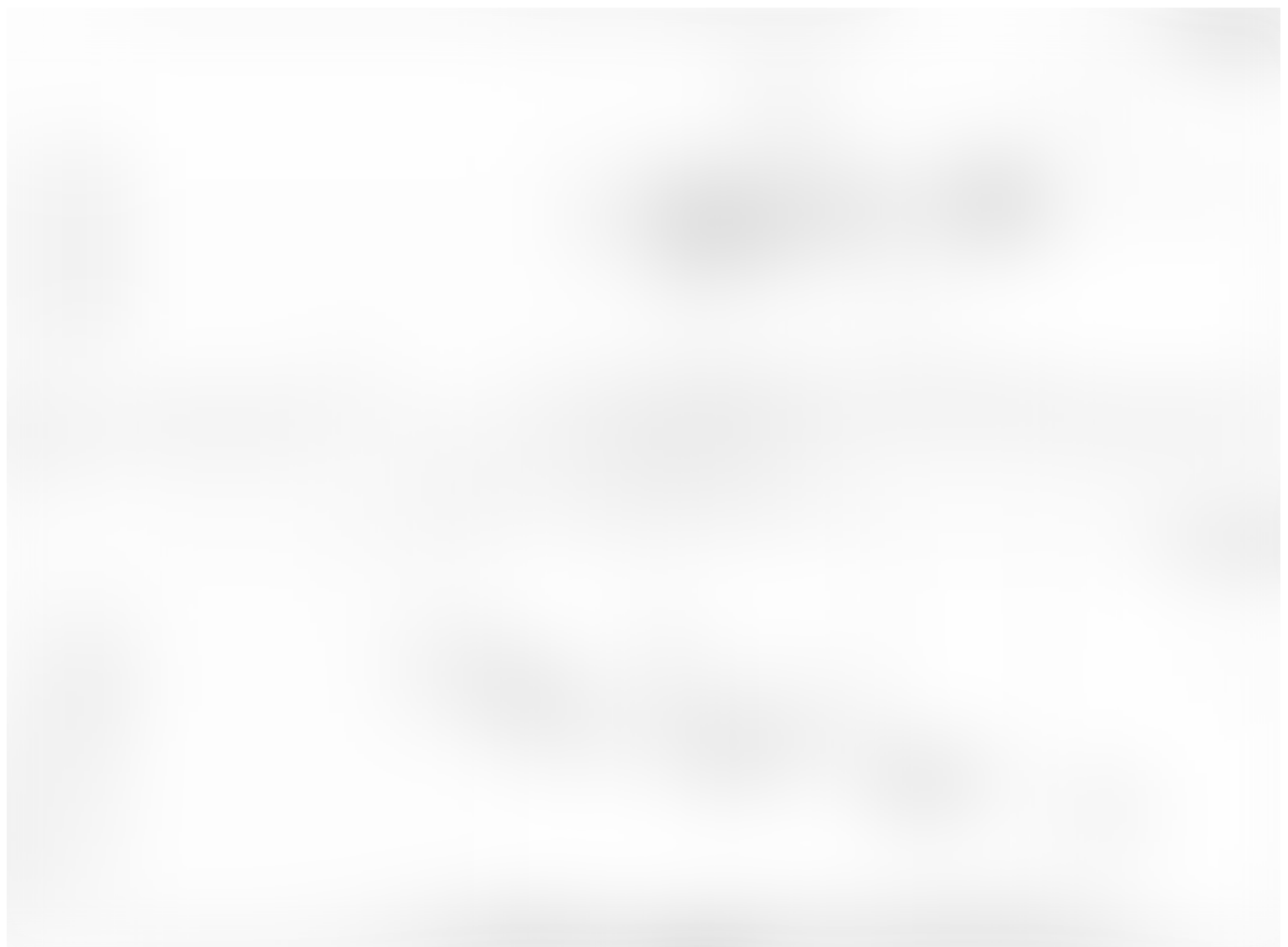


**changing the reality, so it requires to be careful** and to have in mind what it means for the outputted results of our classifier.

## Getting additional features

We discussed in the previous subsection the fact that resampling the training dataset (modifying the classes proportions) can be or not a good idea depending on the real purpose of the classifier. We saw in particular that if the two classes are imbalanced, not well separable and that we target a classifier with the best possible accuracy, then getting a classifier that always answer the same class is not necessarily a problem but just a fact: there is nothing better to do with these variables.

However, it remains possible to obtain better results in terms of accuracy by enriching the dataset with an additional feature (or more). Let's go back to our first example where classes were not well separable: maybe can we find a new additional feature that can help distinguish between the two classes and, so, improve the classifier accuracy.



Looking for additional features can help separate two classes that were not initially separable.



information from the reality is a far better idea when it is possible.

## Reworking the problem is better

Up to now the conclusion is pretty disappointing: if the dataset is representative of the true data, if we can't get any additional feature and if we target a classifier with the best possible accuracy, then a "naive behaviour" (answering always the same class) is not necessarily a problem and should just be accepted as a fact (if the naive behaviour is not due to the limited capacity of the chosen classifier, of course).

So what if we are still unhappy with these results? In this case, it means that, in one way or another, **our problem is not well stated** (otherwise we should accept results as they are) and that we should rework it in order to get more satisfying results. Let's see an example.

## Cost-based classification

The feeling that obtained results are not good can come from the fact that the objective function was not well defined. Up to now, we have assumed that we target a classifier with high accuracy, assuming at the same time that both kinds of errors ("false positive" and "false negative") have the same cost. In our example it means we assumed that predicting C0 when true label is C1 is as bad as predicting C1 when true label is C0. Errors are then symmetric.

Let's consider our introductory example with defective (C1) and not defective (C0) products. In this case, we can imagine that not detecting a defective product will cost more to the company (customer service costs, possible juridical costs if dangerous defects, ...) than wrongly labelling a not defective product as defective (production cost lost). Now, predicting C0 when true label is C1 is far worse than predicting C1 when true label is C0. Errors are no longer symmetric.

Consider then more particularly that we have the following costs:

- predicting C0 when true label is C1 costs P01
- predicting C1 when true label is C0 costs P10 (with  $0 < P10 << P01$ )



## Theoretical minimal cost ( $\infty$ )

From a theoretical point of view, we don't want to minimise the error probability defined above but the expected prediction cost given by

$$\begin{aligned}
 \mathbb{E}(\text{cost}) &= \mathbb{P}(\text{predicted\_C0}, \text{true\_C1})P_{01} + \mathbb{P}(\text{predicted\_C1}, \text{true\_C0})P_{10} \\
 &= \int_{\mathbb{R}} [\mathbb{P}(\text{predicted\_C0}, \text{true\_C1}, x)P_{01} + \mathbb{P}(\text{predicted\_C1}, \text{true\_C0}, x)P_{10}]dx \\
 &= \int_{\mathbb{R}} [\mathbb{P}(\text{predicted\_C0}|\text{true\_C1}, x)\mathbb{P}(\text{true\_C1}, x)P_{01} + \mathbb{P}(\text{predicted\_C1}|\text{true\_C0}, x)\mathbb{P}(\text{true\_C0}, x)P_{10}]dx \\
 &= \int_{\mathbb{R}} [\mathbb{P}(\text{predicted\_C0}|x)\mathbb{P}(\text{true\_C1}, x)P_{01} + \mathbb{P}(\text{predicted\_C1}|x)\mathbb{P}(\text{true\_C0}, x)P_{10}]dx \\
 &= \int_{\mathbb{R}} [\mathbb{1}_{\{C(x)=C_0\}}\mathbb{P}(\text{true\_C1}, x)P_{01} + \mathbb{1}_{\{C(x)=C_1\}}\mathbb{P}(\text{true\_C0}, x)P_{10}]dx
 \end{aligned}$$

where  $C(\cdot)$  defines the classifier function. So, if we want to minimise the expected prediction cost, the theoretical best classifier  $C(\cdot)$  minimises

$$\mathbb{1}_{\{C(x)=C_0\}} \times \mathbb{P}(\text{true\_C1}, x) \times P_{01} + \mathbb{1}_{\{C(x)=C_1\}} \times \mathbb{P}(\text{true\_C0}, x) \times P_{10}$$

or equivalently, dividing by the density of  $x$ ,  $C(\cdot)$  minimises

$$\mathbb{1}_{\{C(x)=C_0\}} \times \mathbb{P}(\text{true\_C1}, x) \times P_{01} + \mathbb{1}_{\{C(x)=C_1\}} \times \mathbb{P}(\text{true\_C0}, x) \times P_{10}$$

So, with this objective function, the best classifier from a theoretical point of view will then be such that:

$$\mathbb{1}_{\{C(x)=C_0\}} \times \mathbb{P}(\text{true\_C1}, x) \times P_{01} + \mathbb{1}_{\{C(x)=C_1\}} \times \mathbb{P}(\text{true\_C0}, x) \times P_{10}$$

Notice that we recover the expression of the “classic” classifier (focus on accuracy) when costs are equal.

## Probability threshold

One first possible way to take into account the cost in our classifier is to do it after the training. The idea is, first, to train a classifier the basic way to output the following probabilities



without assuming any costs. Then, the predicted class will be C0 if

$$P(x|C0)P(C0) > P(x|C1)P(C1)$$

and C1 otherwise.

Here, it doesn't matter which classifier we are using as long as it outputs the probability of each class for a given point. In our main example, we can fit a Bayes classifier on our data and we can then reweight the obtained probabilities to adjust the classifier with the costs errors as described.

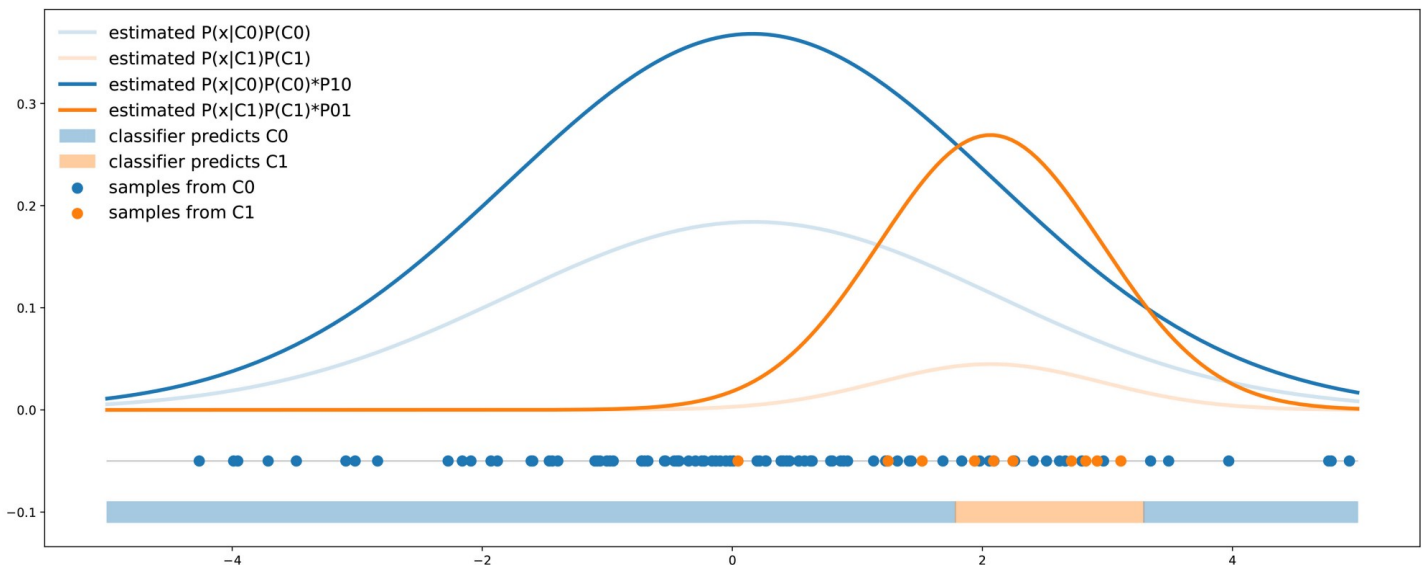


Illustration of the probability threshold approach: the outputted probabilities are reweighted such that costs are taken into account in the final decision rule.

## Classes reweight

The idea of class reweight is to take into account the asymmetry of cost errors directly during the classifier training. Doing so, the outputted probabilities for each class will already embed the cost error information and could then be used to define a classification rule with a simple 0.5 threshold.

For some models (for example Neural Network classifiers), taking the cost into account during the training can consist in adjusting the objective function. We still want our classifier to output

$$\mathbb{P}(\text{true\_C1}|x) \text{ and } \mathbb{P}(\text{true\_C0}|x)$$



For some other models (for example Bayes classifier), resampling methods can be used to bias the classes proportions such that to enter the cost error information inside the classes proportions. If we consider the costs  $P_{01}$  and  $P_{10}$  (such that  $P_{01} > P_{10}$ ), we can either:

- oversample the minority class by a factor  $P_{01}/P_{10}$  (cardinality of the minority class should be multiplied by  $P_{01}/P_{10}$ )
- undersample the majority class by a factor  $P_{10}/P_{01}$  (cardinality of the majority class should be multiplied by  $P_{10}/P_{01}$ )

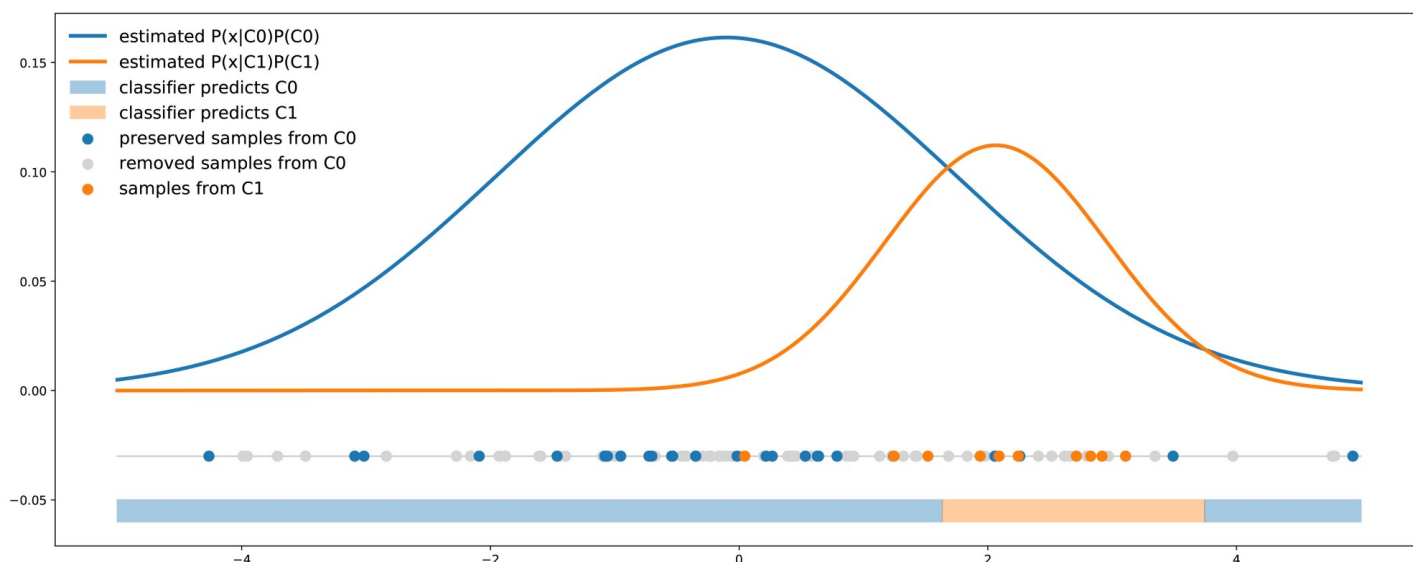


Illustration of the class reweight approach: the majority class is undersampled with a proportion that is chosen carefully to introduce the cost information directly inside the class proportions.

## Takeaways

The main takeaways of this article are:

- whenever using a machine learning algorithm, evaluation metrics for the model have to be chosen cautiously: we must use the metrics that gives us the best overview of how well our model is doing with regards to our goals





can be a “naive” one that always answer the majority class

- resampling methods can be used but have to be thought carefully: they should not be used as stand alone solutions but have to be coupled with a rework of the problem to serve a specific goal
- reworking the problem itself is often the best way to tackle an imbalanced classes problem: the classifier and the decision rule have to be set with respect to a well chosen goal that can be, for example, minimising a cost

We should notice that we have not discussed at all techniques like “stratified sampling” that can be useful when batch training a classifier. When facing an imbalanced classes problem, such techniques ensure more stability during the training (by removing the proportions variance inside batches).

Finally, let’s say that the main keyword of this article is “goal”. Knowing exactly what you want to obtain will help overcome imbalanced dataset problems and will ensure having the best possible results. Defining the goal perfectly should always be the first thing to do and is the starting point of any choice that have to be done in order to create a machine learning model.

Thanks for reading!

Last article written with Joseph Rocca:

### **Understanding Generative Adversarial Networks (GANs)**

Building, step by step, the reasoning that leads to GANs.

[towardsdatascience.com](https://towardsdatascience.com)

---

Get started

Open in app



---

and cutting-edge research to original features you don't want to miss. [Take a look.](#)

Get this newsletter

Machine Learning

Data Science

Artificial Intelligence

Data

Getting Started

[About](#) [Write](#) [Help](#) [Legal](#)

Get the Medium app

