Classifier calibration

The why, when and how of model calibration for classification tasks.

Dimitris Poulopoulos Follow
Mar 9 · 6 min read ★

Photo by Christophe Hautier on Unsplash

. . .

When dealing with a classification problem, collecting only the predictions on a test set is hardly enough; more often than not we would like to compliment them with some level of confidence. To that end, we make use of the associated probability, meaning the likelihood calculated by the classifier, which specifies the class for each sample. But does this always reflect reality? And if not, how can we tell?

Introduction

Imagine we have two binary classifiers; model `A` and model `B`. Model `A` is 85% accurate and 0.86 confident for each prediction it makes. On the other hand, model `B` is also 85% accurate but 0.99 confident for each of its predictions. Which model do you think is better?

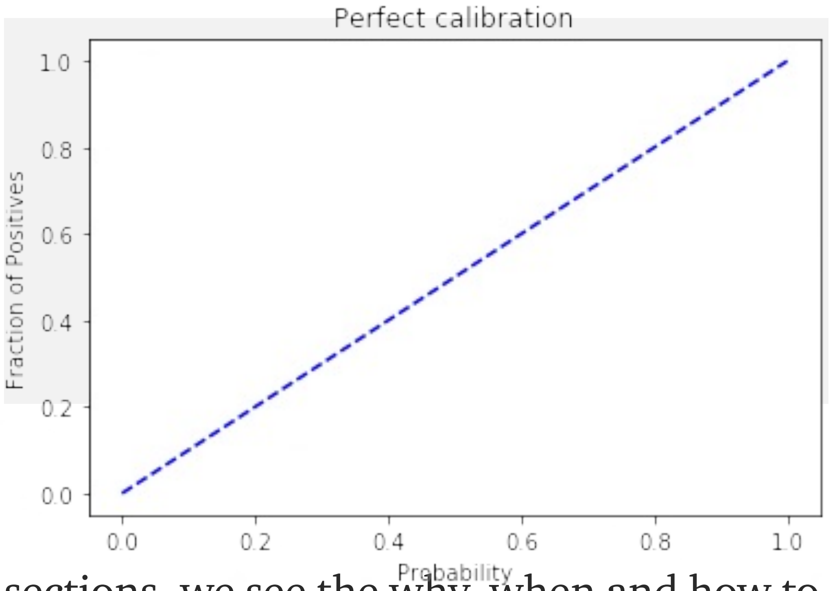
↳ overconfident

In this story, I will try to convince you that model `A` is better. Model `A` considers itself precise 86% of the time and indeed, that's almost the case. To the contrary, model `B` is overconfident about its predictions. This toy example demonstrates the intuition behind probability and model calibration.

Model calibration refers to the process where we take a model that is already trained and apply a post-processing operation, which improves its probability estimation. **Thus, if we were to inspect the samples that were estimated to be positive with a probability of 0.85, we would expect that 85% of them are in fact positive.**

> Formally, a model is perfectly calibrated if, for any probability value `p`, a prediction of a class with confidence `p` is correct `100*p` per cent of the time.

Now, because an image is worth a thousand words if you try to visualize it and plot every value of `p`, in the interval from `0` to `1`, we would expect to get a perfect linear relationship between the computed probability and the fraction of positives.



In the following sections, we see the why, when and how to calibrate your classifiers.

Why is model calibration important?

Model calibration is important only if you care about the probabilities your model computes. For instance, let us say that you are building a recommender engine, that ranks products according to user preferences. If your model estimates that user `u` will buy product `a` with probability 0.9, and item `b` with probability 0.7, you can go ahead and serve product `a` first. No need to calibrate that model.

However, **if you are building a mission-critical application, which computes the probability of a person being sick, the actual probability value is significant. If, for example, your model is not that confident for a specific patient, a human doctor should certainly know about it and act accordingly.**

There are also other cases where model calibration is useful:

- *Debugging:* we want to know when our model is wrong with high confidence or assigns a low probability to the correct class
- *Ensembles:* if we want to combine many probability models, having accurate predictions makes a difference

How to check your model

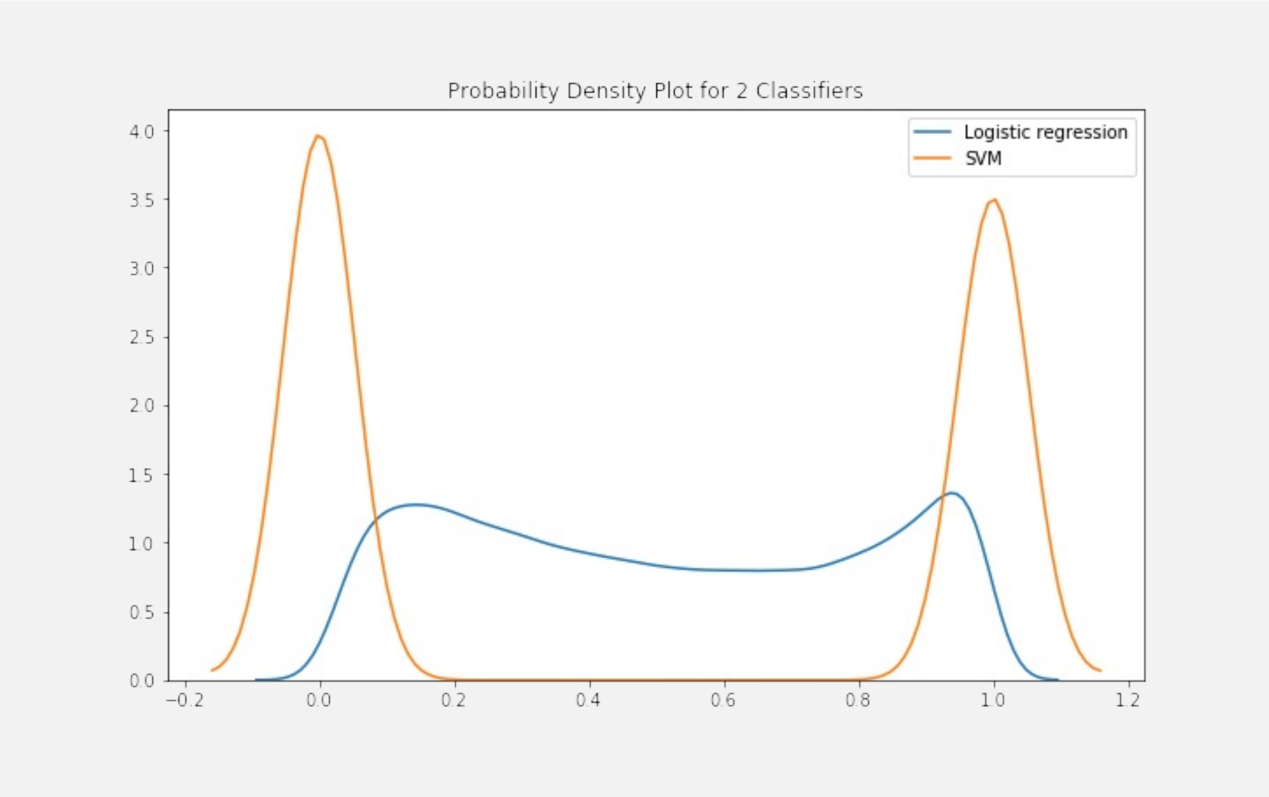
So far we have seen what is model calibration and why it is important in some cases. But how can we check if our classifier is calibrated?

The best way is to see it with our own eyes. For this experiment, we create a random dataset for classification, using the `make_classification` helper method that scikit-learn provides.

Next, we instantiate two classifiers to compare; a simple logistic regression model and an implementation of Support Vector Machines provided by scikit-learn.

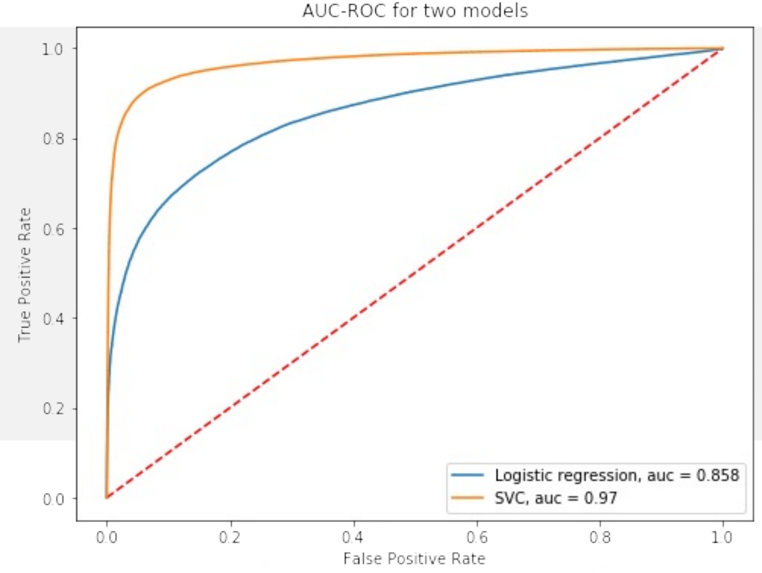
Finally, we fit the classifiers to our training data and compute our predictions on the test data set. Specifically for the SVM, to get the probabilities for the positive class we need to know how the decision function separates the test samples, and normalize the results to be between `0` and `1`.

Let us now plot the `Kernel Density Estimation` for the two classifiers.



Logistic regression Vs SVM KDE plot

As we expected, the results we get from logistic regression is spread from `0` to `1`, while the SVM predictions are exactly `0` or `1`. Following, let us check the AUC-ROC curve for the two binary classifiers, but this time using the probabilities we calculated for the SVM.



- acc but not calibrated
- similar to random but perfectly calibrated

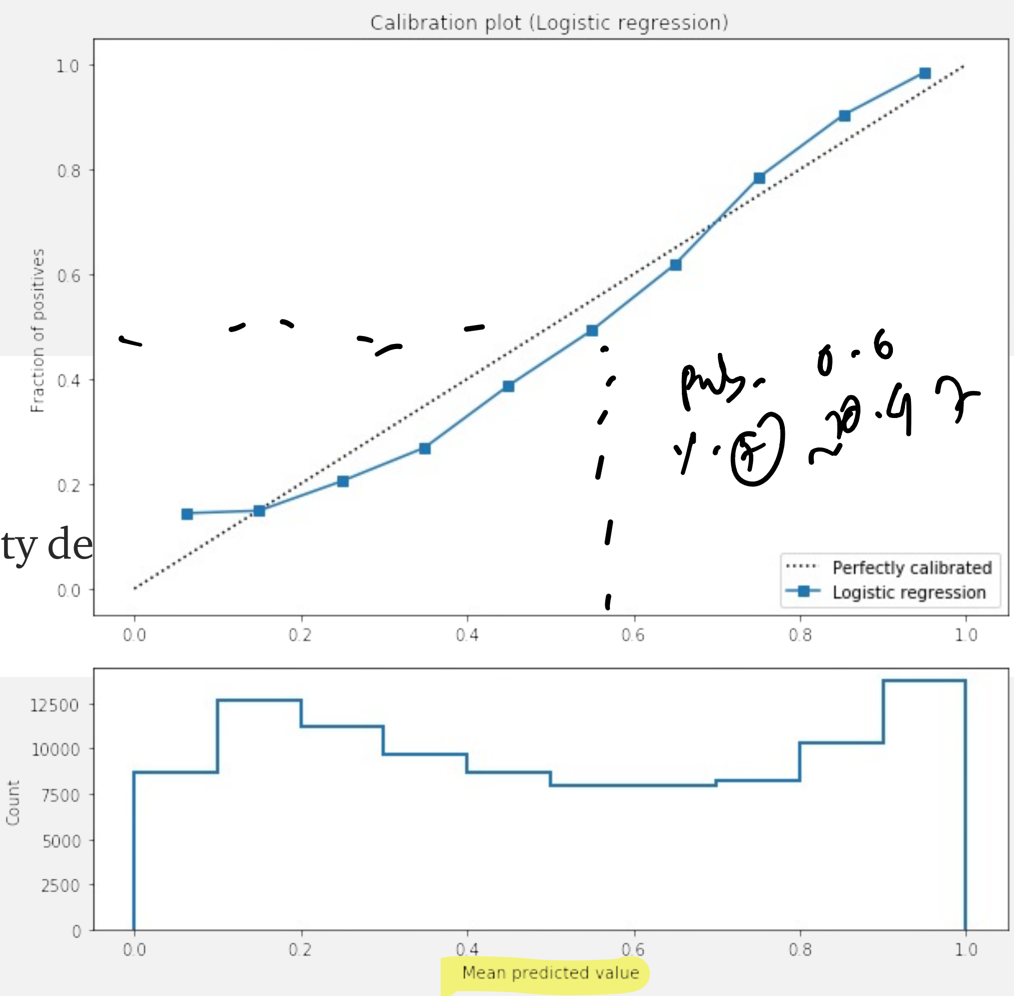
We can see that the SVM is almost perfect on this data set. But the accuracy is a totally different conversation than calibration. We can have a perfectly accurate model that is not calibrated at all and, on the other hand, a model that is no better than random, which is perfectly calibrated nonetheless. So, how can we check?

The first step is to take all predictions and group them into bins. We are going to group them by the probability estimation that the model made. Next, we calculate the fraction of positives per bin and finally the average confidence per bin, which is just the average of the probability estimates of the samples that belong to that bin. If we plot that average against the fraction of positives per bin we get the reliability plot. We want that plot to resemble the linear plot we saw at the beginning.

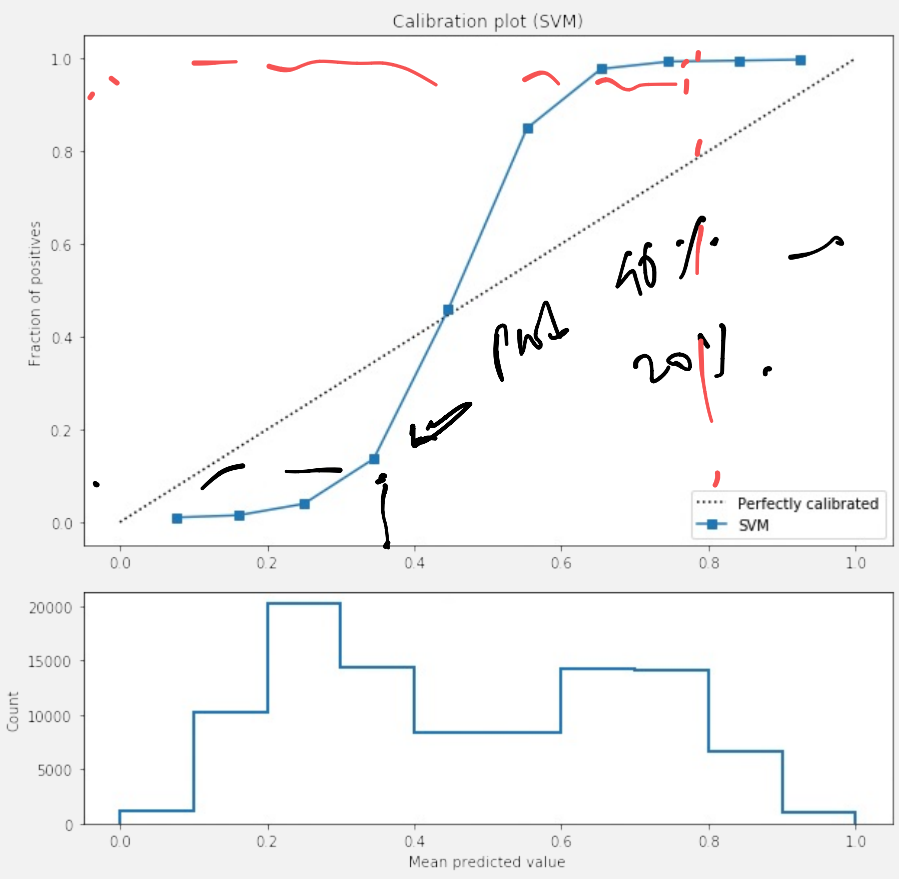
To plot the calibration curve of each classifier we define a utility function like the one below.

We are now ready to plot the calibration curve for each model. Let us start with logistic regression.

It seems pretty de



06 — 0.47
negative
overconfident
for 0.6 —
0.47!
of the
predicted



overconfident.

prob. for ⑧ are
almost 100%.
underconfident

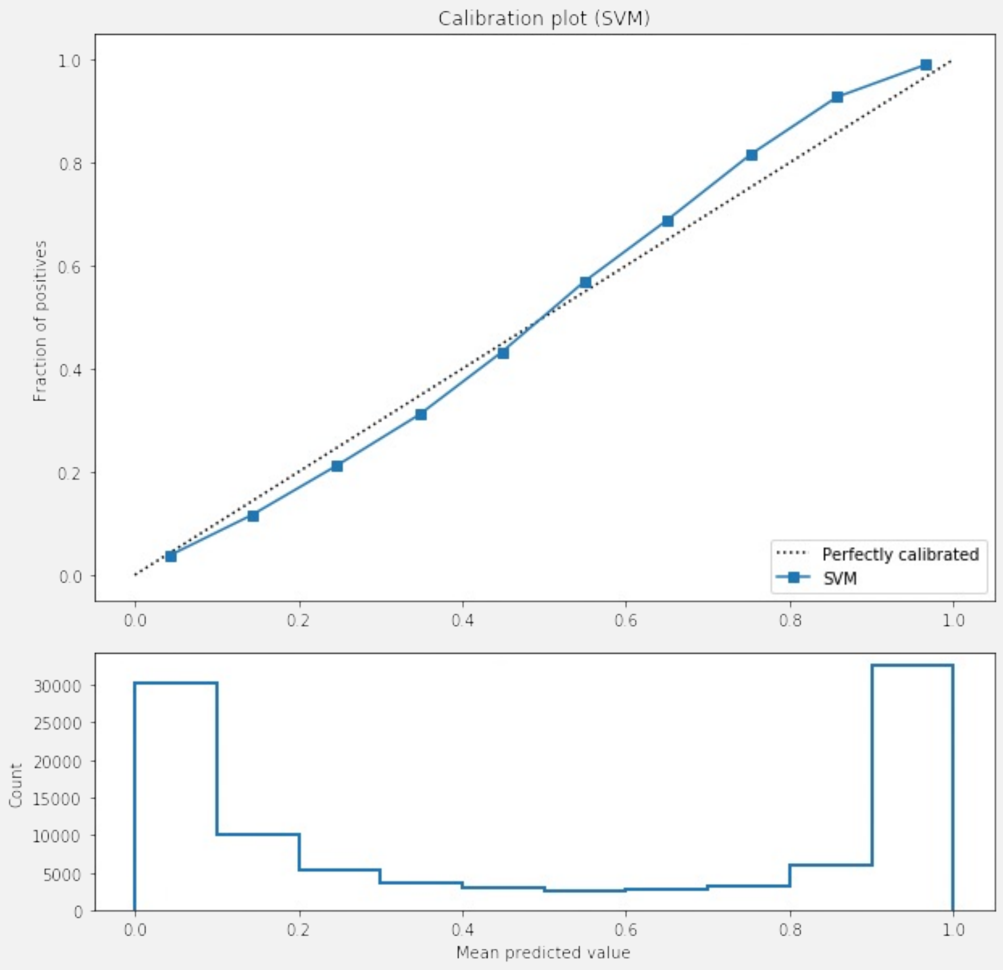
Calibration plot for SVM

It is evident that the SVM model is far from calibrated. We can say that is under-confident when it predicts that the sample does not belong to the positive class and overconfident otherwise. So, how can we fix this?

Calibrating the model

The two most popular methods of calibrating a machine learning model are the `isotonic` and `Platt's` method.

`Scikit-learn` provides a base estimator for calibrating models through the `CalibratedClassifierCV` class. For this example, we will use the `Platt's` method, which is equivalent to setting the `method` argument in the constructor of the class to `sigmoid`. If you want to use the `isotonic` method you can pass that instead.



↗ Calibrated.

Calibrated SVM classifier

The outcome for the SVM classifier is impressively different. Now we have a calibrated SVM classifier. Please note that if you call the `predict_proba` method on the SVM classifier, the results are already calibrated via `Platt's` method (see here). You can try it yourself.

Also, keep in mind that the accuracy of the model might be lower after calibration. For example, the AUC-ROC curve is now 0.963. Thus, we can see that in some case we might have a trade-off between accuracy and calibration to consider. You can always check other metrics as well (e.g. precision, recall, F1 score etc.).

isotonic

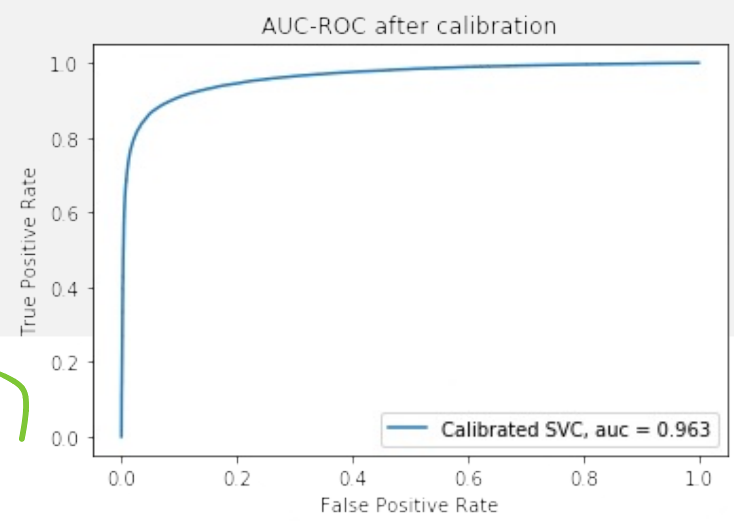
no sigmoid assumption.

prone to overfit

platt's scaling

sigmoid assumption

better for small data (compared to isotonic)



Conclusion

In this story, we examined what is model calibration, why and when to use it, how to check if your classifier is calibrated and how to potentially fix it if not.

Finally, I want to give a rule of thumb, as to whether to use *isotonic regression* of *Platt's scaling*. Isotonic regression does not make the sigmoid assumption, thus, it may be a better fit for a broad amount of cases. On the other hand, it is more prone to overfit. Thus, if we have a small data set, *Platt's scaling* might work better.

> ***My name is Dimitris Poulopoulos and I'm a machine learning researcher at BigDataStack and PhD(c) at the University of Piraeus, Greece. I have worked on designing and implementing AI and software solutions for major clients such as the European Commission, Eurostat, IMF, the European Central Bank, OECD, and IKEA. If you are interested in reading more posts about Machine Learning, Deep Learning and Data Science, follow me on Medium, LinkedIn or @james2pl on twitter.***

Machine Learning · Supervised Learning · Classification · Scikit Learn · Programming