**Why PR/ROC AUC should NOT be your first choice metrics for classification tasks**

[[Anatoly Alekseev](https://medium.com/@fingoldo?source=post_page-----8ed516cec1ee--------------------------------)](https://medium.com/@fingoldo?source=post_page-----8ed516cec1ee--------------------------------)

[Anatoly Alekseev](https://medium.com/@fingoldo?source=post_page-----8ed516cec1ee--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F33768f06b3b4&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40fingoldo%2Fwhy-rp-roc-auc-should-not-be-your-first-choice-metrics-for-classification-tasks-8ed516cec1ee&user=Anatoly+Alekseev&userId=33768f06b3b4&source=post_page-33768f06b3b4----8ed516cec1ee---------------------post_header-----------)

15 min read

·

Aug 8

17

In this article I’d like to share with you my findings about inconsistency of traditional ML metrics when it comes to imbalanced binary classification, and propose a superior metric from calibration domain that works well and possesses high discriminative power not only in cases of class imbalance, but also when the predictability itself is fundamentally weak (i.e., when factors most predictive to the target are unknown or not reachable, and everything we can hope is to explain at least *some*part of target’s variation with help of modelling).

Why is this topic important, in the first place? Because the goal of supervised machine learning process is to produce high quality models that generalize well to unseen data, and the quality is usually judged by special metrics aggregated over cross-validation folds. Choice of metrics is crucial as it guides construction and development of all blocks of a ML pipeline, be it preprocessing, feature selection, or hyperparameter tuning. Metrics chosen by a data scientist for a particular task must, therefore, be able to clearly distinguish between good and useless models, to be able to continue moving towards even better models.

Why is it worth writing an article about? Because there are, as it seems, two layers of public misconceptions circulating in the DS gang regarding the choice of classification metrics. First one hides in the fact that default scoring for classification tasks in a lot of ML packages is accuracy, that is a totally inappropriate metric for cases where minority class is of the most interest to researcher. Once the first layer is surpassed by an aspiring data scientist, they read on internet and in books that quantities such as area under ROC and especially PR curves handle class imbalance much better and are the recommended choices, and hit the second layer.

My experience, however, tells that there is even better class of metrics able of evaluating classification models, namely, calibration metrics. I learned their advantages working on a task of prediction goals from the corners in a football match, where minority class (scoring from a corner) frequency was only around 4%, and with the default threshold of 0.5 no model was NEVER predicting a goal. Ever. How should we even evaluate models on such tasks, when the row in classification report belonging to the positive (minority) class contains all zeros, except the support column?

Does it mean that ML is useless for such tasks? Not at all. Enter the world of [probabilistic forecasts evaluation](https://en.wikipedia.org/wiki/Forecast_skill). The key is that we have to walk away from binary predictions in favor of probabilistic ones. Forget about the .predict() method and stick with the .predict\_proba()! Once done that, we’ll have to compare issued predictions with actual outcomes. One way to do that is to compute a venerable [Brier score](https://en.wikipedia.org/wiki/Brier_score) (BR). [properscoring](https://github.com/properscoring/properscoring/tree/master" \t "_blank) library offers another interesting contender, a continuous ranked probability score ([CRPS](https://www.lokad.com/continuous-ranked-probability-score)). Idea of both is essentially similar to that of model’s [calibration checking](https://scikit-learn.org/stable/modules/calibration.html): after collecting, for example, 100 predictions in the bin of, say, from 0.7 to 0.75, why don’t we check what percentage of them actually resulted in positive outcomes? A value close to the bin’s center, i.e., 0.725, would signify good calibration and therefore high quality of the model in respective region. BR and CRPS compute the same essence in slightly different manners, and based on different premises.

Our goal for this article would be to test them, among other, more “standard” classification metrics, in their ability to discriminate between good and bad models, and see if we can come up with a custom competing metric from calibration domain. We’ll do so by conducting a series of controlled randomized experiments with large number of repetitions to see the limits of what’s possible to meet in real tasks and gain confidence in our evaluations.

What we ultimately want to know, is what metric, if any, we should prefer for classification tasks. Ideally, it should be

1. robust to data skew: we want it to work well under different classes ratios, balanced or unbalanced;
2. accurate: it should not denote worthless models as good, and vice versa
3. fast to compute.

How can we design such an experiment, given that there is a plethora of target-predictor relationship powers, from *no relation*to *fully defined*, and a variety of model classes of different learning capability, from highly nonlinear boosted trees down to a linear regression or Naive Bayes classfier? My suggestion is that we skip modelling part entirely and instead operate on predicted probabilities directly. Also we will concentrate on two edge cases: when our predictions extract from the input data maximum possible knowledge (be it definitive to the target or only partial), and no knowledge at all.

Upon sampling “real” outcomes from our probabilities (known in advance) over these edge cases, and computing ML metrics to summarize the quality of such two “models”, we’ll hopefully see how beneficial each ML metric is as a criterion for telling good and bad models apart, in terms of robustness, accuracy and speed. Let’s start with a data generation procedure:

import numpy as np, pandas as pd  
  
def generate\_data(  
 size=10\_000,  
 baseline\_prob: float = None,  
 impact\_span: float = None,  
 fuzzy\_influence: bool = False,  
 fuzzy\_baseline\_prob: float = None,  
 fuzzy\_impact\_span: float = None,  
 clip\_lim: float = 1e-2,  
) -> tuple:  
 """Generates known probabilities and then samples binary outcomes (realizations) from them.  
 Known probabilities can be created uniformly (with default prameters) or with density focused in certain area, specified by baseline\_prob and impact\_span parameters.  
 The latter is useful to simulate class imbalance problems, when occurence of minority class is rare and traditional metrics perform poorly.  
 Sampling can be done  
 1) directly with fuzzy\_influence = False. Generated probs become ral underlying probs of the generated realizations.  
 2) indirectly/fuzzily with fuzzy\_influence = True. In this case, simulating only partial knowledge of influencing factors, bulk of true underlying probabilities  
 is defined by unknown values drawn from separately created uniform distribution, but with our known probs added. This is to model a situation where our ML tools  
 are able to generalize from partial information contained at least in observed data: model would use overall classes balance as a base figure, plus add its  
 ML knowledge distilled from supplied features. The ratio of known/unknown influence can be regulated by fuzzy\_baseline\_prob and fuzzy\_impact\_span params.  
 """  
 known\_probs = np.random.uniform(size=size)  
 if baseline\_prob and impact\_span:  
 known\_probs = np.clip(baseline\_prob + (known\_probs - 0.5) \* impact\_span, clip\_lim, 1 - clip\_lim)  
  
 if fuzzy\_influence:  
 fuzzy\_probs = np.random.uniform(size=size)  
 if fuzzy\_baseline\_prob and fuzzy\_impact\_span:  
 fuzzy\_probs = np.clip(fuzzy\_baseline\_prob + (fuzzy\_probs - 0.5) \* fuzzy\_impact\_span, clip\_lim, 1 - clip\_lim)  
 probs = np.clip(known\_probs + fuzzy\_probs, clip\_lim, 1 - clip\_lim)  
 known\_probs = np.clip(known\_probs + fuzzy\_probs.mean(), clip\_lim, 1 - clip\_lim)  
 else:  
 probs = known\_probs  
 realizations = (np.random.uniform(size=size) < probs).astype(np.int8)  
  
 return known\_probs, realizations

We’ll be evaluating standard classification report, PR and ROC AUC from *scikit-learn*, Brier and continuous ranked probability scores from *properscoring,*geometric mean score from *imbalanced-learn*. As our own contender we’ll be using CALIB, a mean absolute error over differences between predicted and real positive outcomes percentage. Consider it as a slight modification of widely known [calibration curve](https://en.wikipedia.org/wiki/Calibration_curve): entire predictions range is binned from min\_prob to max\_prob into N consecutive intervals, occurrence frequency is tallied for each bin and compared to theoretical middle of that bin. I prefer MAE to MSE here ’cause I want it to be more robust to possible outliers arising due to imminent bin data scarcity. Exact code for CALIB you will find in the accompanying notebook.

Additionally, inspired by the words of Timothy Masters in his [book](https://www.amazon.com/Assessing-Improving-Prediction-Classification-Algorithms/dp/1484233352/) “Assessing and Improving Prediction and Classification”: ”*The importance of consistent performance is often ignored, with average performance being the focal point instead. However, in most cases, a model that performs fairly well across the majority of training cases will ultimately outperform a model that performs fabulously most of the time but occasionally fails catastrophically. Properly designed training sets and optimization criteria can take consistency into account.*”, I decided to add a simple measure of consistency, a standard deviation. For all metrics mentioned above we, where possible, will be estimating not only the means, but also the standard deviations across a range of predictions. As the decision criteria we for each metric will inspect:

1. its mean;
2. its std;
3. its mean+sdt/2 or mean-std/2 depending on if it’s better for a particular metric to increase or decrease. Division by 2 here is to give the mean a slightly higher weight. Actually, only BR, CRPS and CALIB metrics are natively not reduced to a single number, and allow computing std. All of them are in a sense score losses, so we will use the *mean+sdt/2*option for them as the 3rd option.

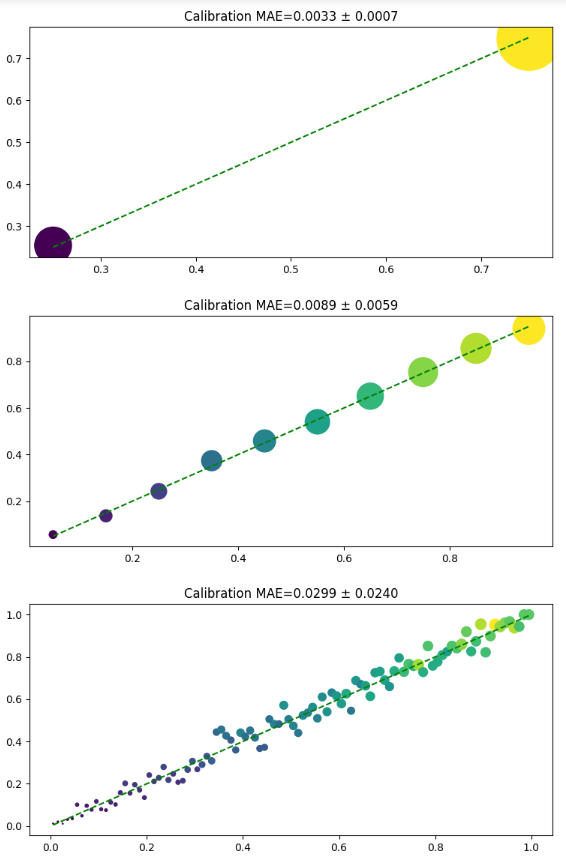
Experiments we’ll run for a balanced and unbalanced setups with approx. 50% and 4% minority class occurrence, respectively. In addition to printing classification reports and computing the metrics, I’ll be showing calibration plots for a set of 2, 10 and 100 bins. We’ll use 10 000 data points. Here’s the code of a quick estimation procedure that compares the “True” (all possible predictive knowledge extracted from data, predictions should be highly relevant, as much as ‘fuzzy\_influence’ parameter allows) and “Rand” (no knowledge is gained from data, predictions are absolutely random and not relevant to the outcomes) models:

from sklearn.metrics import average\_precision\_score, roc\_auc\_score, brier\_score\_loss, classification\_report  
from imblearn.metrics import classification\_report\_imbalanced  
from properscoring import crps\_ensemble, brier\_score  
from mlframe.metrics import fast\_calibration\_report  
from imblearn.metrics import geometric\_mean\_score  
  
ndigits = 4  
  
  
def compute\_metrics\_block(realizations: np.ndarray, probs: np.ndarray, print\_block\_name: str = None, ndigits: int = 4) -> tuple:  
 """Computes a set of metrics from a realizations/probs arrays pair."""  
 calibration\_mae, calibration\_std = fast\_calibration\_report(y\_true=realizations, y\_pred=probs, nbins=10, show\_plots=False)  
 brier\_vals = brier\_score(observations=realizations, forecasts=probs)  
 crps\_vals = crps\_ensemble(  
 observations=realizations,  
 forecasts=probs,  
 )  
 gms = geometric\_mean\_score(y\_true=realizations, y\_pred=probs > 0.5, average="weighted")  
 pr\_auc = average\_precision\_score(  
 y\_true=realizations,  
 y\_score=probs,  
 )  
 roc\_auc = roc\_auc\_score(  
 y\_true=realizations,  
 y\_score=probs,  
 )  
  
 if print\_block\_name:  
 print(  
 f"{print\_block\_name} perf: GMS={gms:.{ndigits}f}, PR AUC={pr\_auc:.{ndigits}f}, ROC AUC={roc\_auc}, BR MEAN={brier\_vals.mean():.{ndigits}f}, BR STD={brier\_vals.std():.{ndigits}f}, CRPS MEAN={crps\_vals.mean():.{ndigits}f}, CRPS STD={crps\_vals.std():.{ndigits}f}, CALIB MAE={calibration\_mae:.{ndigits}f}, CALIB STD={calibration\_std:.{ndigits}f}"  
 )  
  
 return gms, pr\_auc, roc\_auc, calibration\_mae, calibration\_std, brier\_vals, crps\_vals  
  
  
def show\_ideal\_vs\_random(  
 probs: np.ndarray,  
 realizations: np.ndarray,  
 show\_bins=[2, 10, 100],  
 figsize: tuple = (9, 4),  
 nsummary\_runs: int = 0,  
 gen\_func: object = None,  
) -> None:  
 """Takes true probabilities and a snapshot of their outcomes, compares a range of classification ML metrics for  
 1) true probabilities  
 2) completely unrelated probabilities sampled randomly  
  
 Capable ML metrics should be able to separate true underlying probabilities from unrelated junk.  
  
 Draws calibration plots for a set of bins.  
 """  
  
 columns = "gms pr\_auc roc\_auc calib\_mae calib\_std calib\_mae\_std br\_mean br\_std br\_mean\_std crps\_mean crps\_std crps\_mean\_std".split()  
 misclassifications = np.zeros(len(columns), dtype=np.int64)  
 lines = []  
 for j in tqdm(range(nsummary\_runs + 1)):  
 if not nsummary\_runs:  
 print("True classification report:")  
 print(classification\_report(y\_true=realizations, y\_pred=probs > 0.5, digits=ndigits, zero\_division=0))  
 if gen\_func:  
 probs, realizations = gen\_func()  
 gms, pr\_auc, roc\_auc, calibration\_mae, calibration\_std, brier\_vals, crps\_vals = compute\_metrics\_block(  
 realizations=realizations, probs=probs, print\_block\_name="True" if not nsummary\_runs else None, ndigits=ndigits  
 )  
 if nsummary\_runs:  
 meaningful\_model\_metrics = [gms, pr\_auc, roc\_auc, calibration\_mae, calibration\_std, calibration\_mae + calibration\_std / 2]  
 for arr in (brier\_vals, crps\_vals):  
 arr\_mean, arr\_std = arr.mean(), arr.std()  
 meaningful\_model\_metrics.extend([arr\_mean, arr\_std, arr\_mean + arr\_std / 2])  
 tmp = np.random.uniform(size=len(probs))  
 gms, pr\_auc, roc\_auc, calibration\_mae, calibration\_std, brier\_vals, crps\_vals = compute\_metrics\_block(  
 realizations=realizations, probs=tmp, print\_block\_name="Rand" if not nsummary\_runs else None, ndigits=ndigits  
 )  
 if nsummary\_runs:  
 junk\_model\_metrics = [gms, pr\_auc, roc\_auc, calibration\_mae, calibration\_std, calibration\_mae + calibration\_std / 2]  
 for arr in (brier\_vals, crps\_vals):  
 arr\_mean, arr\_std = arr.mean(), arr.std()  
 junk\_model\_metrics.extend([arr\_mean, arr\_std, arr\_mean + arr\_std / 2])  
  
 line = []  
 for i in range(len(junk\_model\_metrics)):  
 if i < 3:  
 ratio = meaningful\_model\_metrics[i] / junk\_model\_metrics[i]  
 else:  
 ratio = junk\_model\_metrics[i] / meaningful\_model\_metrics[i]  
 if ratio < 1:  
 misclassifications[i] += 1  
 line.append(ratio)  
 lines.append(line)  
 else:  
 if show\_bins:  
 for nbins in show\_bins:  
 calibration\_mae, calibration\_std = fast\_calibration\_report(  
 y\_true=realizations,  
 y\_pred=probs,  
 nbins=nbins,  
 figsize=figsize,  
 )  
 if j >= nsummary\_runs - 1:  
 break  
 if nsummary\_runs:  
 ratios = pd.DataFrame(data=lines, columns=columns)  
 misclassifications = pd.Series(data=misclassifications / nsummary\_runs, index=columns)  
 return ratios, misclassifications

And now to the fun part! Let’s examine ideal vs random performance when the data is balanced:

*probs, realizations = generate\_data(size=10\_000, baseline\_prob=None, impact\_span=None)  
show\_ideal\_vs\_random(probs=probs, realizations=realizations)*

True classification report:  
 precision recall f1-score support  
  
 0 0.7460 0.7440 0.7450 4973  
 1 0.7474 0.7494 0.7484 5027  
  
 accuracy 0.7467 10000  
 macro avg 0.7467 0.7467 0.7467 10000  
weighted avg 0.7467 0.7467 0.7467 10000  
  
True perf: GMS=0.7467, PR AUC=0.8331, ROC AUC=0.8331, BR MEAN=0.1668, BR STD=0.1968, CRPS MEAN=0.3339, CRPS STD=0.2351, CALIB MAE=0.0089, CALIB STD=0.0059  
Rand perf: GMS=0.5062, PR AUC=0.4987, ROC AUC=0.5002, BR MEAN=0.3336, BR STD=0.2998, CRPS MEAN=0.4998, CRPS STD=0.2893, CALIB MAE=0.2437, CALIB STD=0.1517

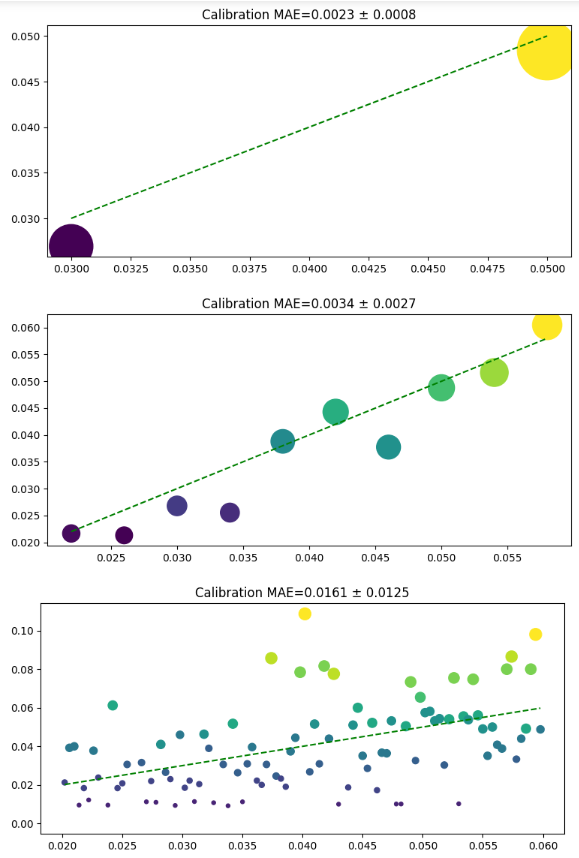


Balanced classes, perfect relationship

As we can see, all “layer 2” metrics plus our newly introduced probabilistic metrics do a good job of separating True from Rand. Let’s inspect imblance similar to what I met in the football project:

*probs, realizations = generate\_data(size=10\_000, baseline\_prob=0.04, impact\_span=0.04)  
show\_ideal\_vs\_random(probs=probs, realizations=realizations)*

True classification report:  
 precision recall f1-score support  
  
 0 0.9621 1.0000 0.9807 9621  
 1 0.0000 0.0000 0.0000 379  
  
 accuracy 0.9621 10000  
 macro avg 0.4810 0.5000 0.4903 10000  
weighted avg 0.9256 0.9621 0.9435 10000  
  
True perf: GMS=0.1910, PR AUC=0.0523, ROC AUC=0.5955, BR MEAN=0.0363, BR STD=0.1743, CRPS MEAN=0.0748, CRPS STD=0.1753, CALIB MAE=0.0034, CALIB STD=0.0027  
Rand perf: GMS=0.5297, PR AUC=0.0463, ROC AUC=0.5404, BR MEAN=0.3298, BR STD=0.2986, CRPS MEAN=0.4967, CRPS STD=0.2883, CALIB MAE=0.4621, CALIB STD=0.2826



96% class imbalance, perfect relationship

Here we spot a problem immediately: second line of a supposedly perfect classification report contains all zeros. Real positive occurence of class one, and, therefore, its probability is so small, that usual 50% threshold never leads to a positive forecast, even for the ideal model. What about “layer 2” metrics? Geometric mean score from imblearn starts misbehaving. ROC AUC of the ideal and random model become pretty close, 0.59 vs 0.54. And that is with only one sampling attempt. It feels like if we try generating data a few more times, we might see the random model beating the ideal one by ROC AUC (it’s indeed the case). PR AUC behave the same: 0.052 and 0.046 are barely distinguishable. Certainly it does not feel like a difference between the most capable and absolutely worthless models, does it?

On the other hand, probabilistic methods really shine. BR MEAN of 0.0363 compared to 0.3298 that’s an x10 difference, very clear and convincing. CRPS stays in the same ballpark with MEAN of 0.0748 to 0.4967, an x7 difference. Their STDs might serve as an additional proof, ratios being from 1.5 to 2.0 and confirming MEANs signals.

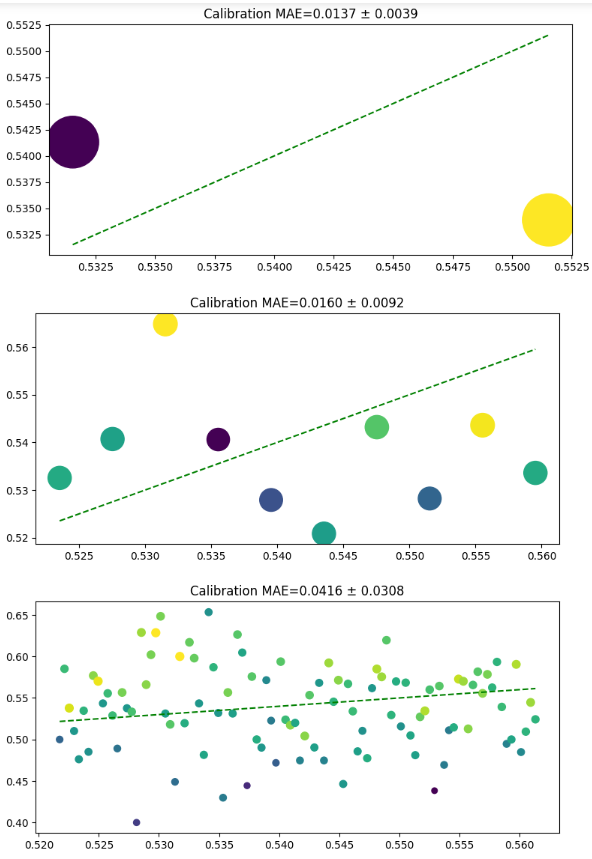
Our freshly invented CALIB metric deserves special attention. CALIB MAE from 0.0034 to 0.4621 it’s more than x100 ratio, unbeatable discriminative power. Same for its STD, 0.0027 vs 0.2826.

Now let’s dig further and deeper. So far, we considered cases with perfect (although probabilistic) relationship between target and hypothetical inputs. It’s because we were generating outcomes directly from known probabilities that later served as a replacement/abstraction for “some powerful model” with its predict\_proba method. Let’s make our task more complex and consider the cased when either our model is not fully capable of learning complex connections, or the supplied input data is missing the most influencing factors.

Our data generating procedure supports that via fuzzy\_influence=True parameter. With it, it knows only a part of true underlying probability used to generate actual outcome/label:

*probs, realizations = generate\_data(size=10\_000, baseline\_prob=0.04, impact\_span=0.04, fuzzy\_influence=True)  
show\_ideal\_vs\_random(probs=probs, realizations=realizations)*

True classification report:  
 precision recall f1-score support  
  
 0 0.0000 0.0000 0.0000 4624  
 1 0.5376 1.0000 0.6993 5376  
  
 accuracy 0.5376 10000  
 macro avg 0.2688 0.5000 0.3496 10000  
weighted avg 0.2890 0.5376 0.3759 10000  
  
True perf: GMS=0.4986, PR AUC=0.5357, ROC AUC=0.4976, BR MEAN=0.2488, BR STD=0.0431, CRPS MEAN=0.4969, CRPS STD=0.0431, CALIB MAE=0.0160, CALIB STD=0.0092  
Rand perf: GMS=0.4971, PR AUC=0.5395, ROC AUC=0.5002, BR MEAN=0.3319, BR STD=0.2942, CRPS MEAN=0.4999, CRPS STD=0.2863, CALIB MAE=0.2527, CALIB STD=0.1428



Balanced classes, weak relationship

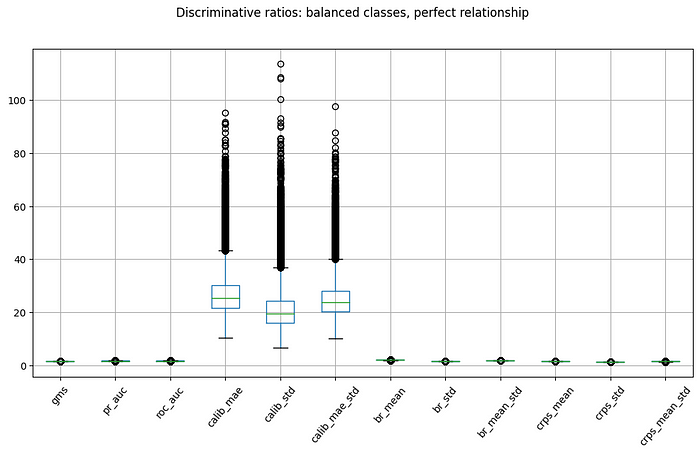
Note that GMS, PR/ROC AUC won’t give you a hand in such situation. You simply have no means anymore to tell apart a nice DS work and a random junk. Even probabilistic tools such as Brier and CRP scores are not that certain anymore. You almost can’t use their means (CRPS MEAN is totally off with 0.4969 to 0.4999, BR MEAN is better with 0.2488 to 0.3319 but for sure will have false positives with more trials), and probably you can’t be certain enough to judge only by STDs.

CALIB, on the other hand, keeps shining: CALIB MAE=0.0160, CALIB STD=0.0092 for a meaningful (as much as possible, given incomplete knowledge) model, and CALIB MAE=0.2527, CALIB STD=0.1428 for a “garbage model”. Not 100+ anymore, but still 20+ ratio in both means and stds. From a quick glance, it appears that we have a clear winner. But let’s be more rigorous and conduct 100000 experiment runs for each case, writing down number of “false alarms” for each metric, i.e., the cases when the metric mistakenly preferred a useless model over a meaningful one. Another useful quantity to track would be the lower and upper bounds for the ratios of metrics for meaningful and junk models: they would denote the *easiness of decision making,*as intuitively it’s much easier to believe in model’s superiority if it’s metric is 20 times better than the one of the competing model, compared to the case when it’s only 1.1 times better.

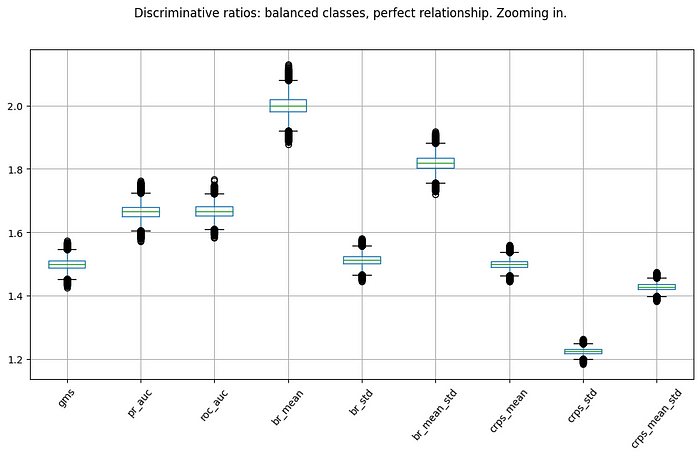
That should give us an impression of how often we will come to wrong conclusions if using one or another metric in our mundane DS cross-validation procedures when dealing with binary (and possibly other types of) classification, and how confidently we (on average) can judge the models using it.

*ratios, misclassifications = show\_ideal\_vs\_random(  
probs=None, realizations=None, nsummary\_runs=100\_000, gen\_func=lambda: generate\_data(size=10\_000, baseline\_prob=None, impact\_span=None)  
)  
misclassifications*

gms 0.0  
pr\_auc 0.0  
roc\_auc 0.0  
calib\_mae 0.0  
calib\_std 0.0  
calib\_mae\_std 0.0  
br\_mean 0.0  
br\_std 0.0  
br\_mean\_std 0.0  
crps\_mean 0.0  
crps\_std 0.0  
crps\_mean\_std 0.0



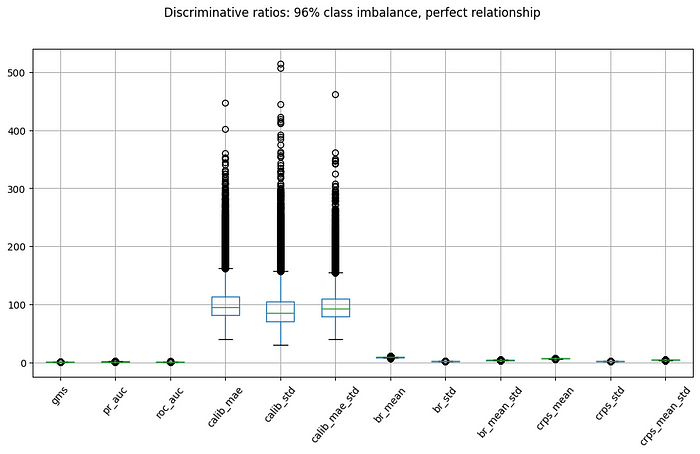
CALIB flavors are outstanding



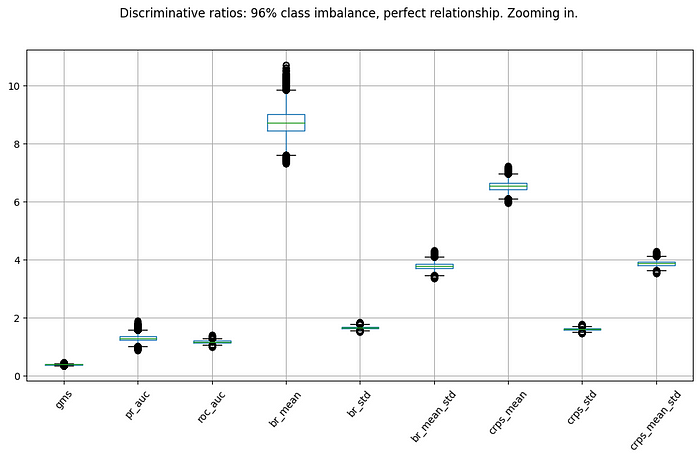
All are doing well, but BR is slightly better than the rest.

*ratios, misclassifications = show\_ideal\_vs\_random(  
probs=None, realizations=None, nsummary\_runs=100\_000, gen\_func=lambda: generate\_data(size=10\_000, baseline\_prob=0.04, impact\_span=0.04)  
)  
misclassifications*

gms 1.00000  
pr\_auc 0.00116  
roc\_auc 0.00001  
calib\_mae 0.00000  
calib\_std 0.00000  
calib\_mae\_std 0.00000  
br\_mean 0.00000  
br\_std 0.00000  
br\_mean\_std 0.00000  
crps\_mean 0.00000  
crps\_std 0.00000  
crps\_mean\_std 0.00000



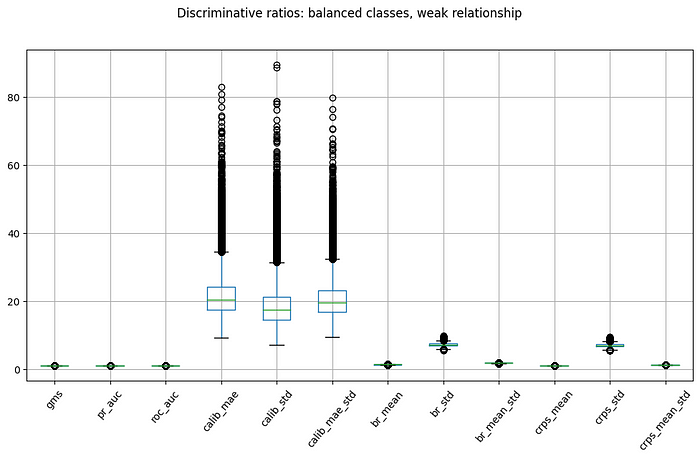
CALIB still great, the rest worsened



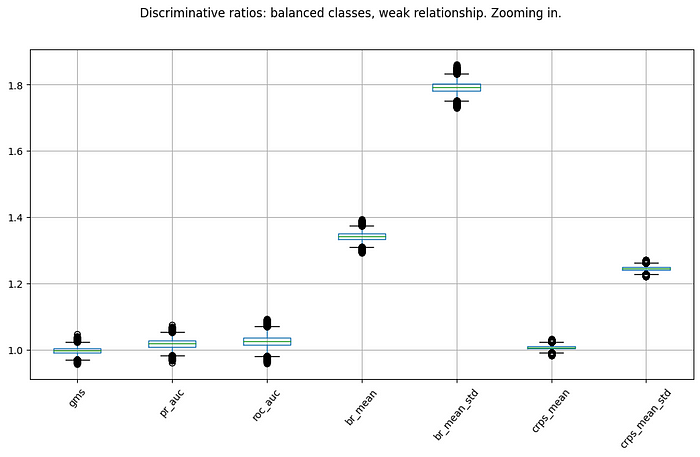
Beyond CALIB, only BR and CRPS are worth using

*ratios, misclassifications = show\_ideal\_vs\_random(  
probs=None,  
realizations=None,  
nsummary\_runs=100\_000,  
gen\_func=lambda: generate\_data(size=10\_000, baseline\_prob=0.04, impact\_span=0.04, fuzzy\_influence=True),  
)  
misclassifications*

gms 0.61855  
pr\_auc 0.08957  
roc\_auc 0.05973  
calib\_mae 0.00000  
calib\_std 0.00000  
calib\_mae\_std 0.00000  
br\_mean 0.00000  
br\_std 0.00000  
br\_mean\_std 0.00000  
crps\_mean 0.12553  
crps\_std 0.00000  
crps\_mean\_std 0.00000



CALIB weakened but still shines



Alternatively, only BR flavors are good enough

SPEED CONSIDERATIONS



CALIB and properscoring procedures are the fastest. ROC/PR AUCs are significantly slower.

CONCLUSION:

When dealing with (at least, binary) classification, prefer CALIB or similar as a metric for CV. It’s most consistent across different class balances, has the lowest % of cases when Random beats Meaningful (be the full or only partial information available), has the top discriminative power and it’s easy to judge by and fast enough to calculate.

Worse in terms of discriminativeness, but more widely known, and still by a big margin better than the rest, is the Brier score.

Refrain from using other (especially layer 2 and 1) metrics as your primary (guiding the model selection process). If you, nevertheless, should, use ROC AUC, not PR AUC — the former has lower percent of misjudgments.

CRPS is bad when the relationship is weak, prefer BR. And generally, there is no point in uniting STD with MEAN, although oftentimes STD is a good discriminator, too.

Here’s the notebook [link](https://github.com/fingoldo/articles/blob/main/Imbalanced%20Classification%20Metrics.ipynb)to reproduce & experiment with.

P.S.:

I have a (mostly Russian) telegram [channel](https://t.me/AspiringDataScience) where I share my thoughts on data science, welcome to join it.

BONUS: CALIB code

from numba import njit  
from math import floor  
from matplotlib import pyplot as plt  
  
@njit()  
def fast\_calibration\_binning(y\_true: np.ndarray, y\_pred: np.ndarray, nbins: int = 100):  
 """Computes bins of predicted vs actual events frequencies. Corresponds to sklearn's UNIFORM strategy."""  
  
 pockets\_predicted = np.zeros(nbins, dtype=np.int64)  
 pockets\_true = np.zeros(nbins, dtype=np.int64)  
  
 min\_val, max\_val = 1.0, 0.0  
 for predicted\_prob in y\_pred:  
 if predicted\_prob > max\_val:  
 max\_val = predicted\_prob  
 elif predicted\_prob < min\_val:  
 min\_val = predicted\_prob  
 span = max\_val - min\_val  
 multiplier = nbins / span  
 for true\_class, predicted\_prob in zip(y\_true, y\_pred):  
 ind = floor((predicted\_prob - min\_val) \* multiplier)  
 pockets\_predicted[ind] += 1  
 pockets\_true[ind] += true\_class  
  
 idx = np.nonzero(pockets\_predicted > 0)[0]  
  
 hits = pockets\_true[idx]  
 freqs\_predicted, freqs\_true = min\_val + (np.arange(nbins)[idx] + 0.5) \* span / nbins, hits / pockets\_predicted[idx]  
  
 return freqs\_predicted, freqs\_true, hits  
  
  
def show\_calibration\_plot(  
 freqs\_predicted: np.ndarray,  
 freqs\_true: np.ndarray,  
 hits: np.ndarray,  
 show\_plots: bool = True,  
 plot\_file: str = "",  
 plot\_title: str = "",  
 figsize: tuple = (12, 6),  
):  
 """Plots reliability digaram from the binned predictions."""  
 fig = plt.figure(figsize=figsize)  
 plt.scatter(freqs\_predicted, freqs\_true, marker="o", s=5000 \* hits / hits.sum(), c=hits, label="Real")  
 x\_min, x\_max = np.min(freqs\_predicted), np.max(freqs\_predicted)  
 plt.plot([x\_min, x\_max], [x\_min, x\_max], "g--", label="Perfect")  
 if plot\_title:  
 plt.title(plot\_title)  
 if plot\_file:  
 fig.savefig(plot\_file)  
 if show\_plots:  
 plt.show()  
 else:  
 plt.close(fig)  
  
  
def fast\_calibration\_report(y\_true: np.ndarray, y\_pred: np.ndarray, nbins: int = 100, show\_plots: bool = True, plot\_file: str = "", figsize: tuple = (12, 6)):  
 """Bins predictions, then computes regresison-like error metrics between desired and real binned probs."""  
  
 freqs\_predicted, freqs\_true, hits = fast\_calibration\_binning(y\_true=y\_true, y\_pred=y\_pred, nbins=nbins)  
 diffs = np.abs((freqs\_predicted - freqs\_true))  
 calibration\_mae, calibration\_std = np.mean(diffs), np.std(diffs)  
  
 if plot\_file or show\_plots:  
 show\_calibration\_plot(  
 freqs\_predicted=freqs\_predicted,  
 freqs\_true=freqs\_true,  
 hits=hits,  
 plot\_title=f"Calibration MAE={calibration\_mae:.4f} ± {calibration\_std:.4f}",  
 show\_plots=show\_plots,  
 plot\_file=plot\_file,  
 figsize=figsize,  
 )  
  
 return calibration\_mae, calibration\_std