Table of contents

MLOps Blog

F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose?

8 min

Jakub Czakon

9th May, 2023

ML Model Development

PR AUC and F1 Score are very robust evaluation metrics that work great for many classification problems but from my experience more commonly used metrics are Accuracy and ROC AUC. Are they better? Not really. As with the famous “AUC vs Accuracy” discussion: there are real benefits to using both. The big question is **when**.

There are many questions that you may have right now:

- When accuracy is a better evaluation metric than ROC AUC?
- What is the F1 score good for?
- What is PR Curve and how to actually use it?
- If my problem is highly imbalanced should I use ROC AUC or PR AUC?

As always it depends, but understanding the trade-offs between different

In this blog post I will:

- Talk about some of the most **common** binary classification **metrics** like F1 score, ROC AUC, PR AUC, and Accuracy
- **Compare them** using an example binary classification problem.

Table of contents

- **over the other** (F1 score vs ROC AUC).

Ok, let's do this!

You will learn about:

Evaluation metrics recap

I will start by introducing each of those classification metrics. Specifically:

- What is the **definition** and **intuition** behind it,
- The **non-technical explanation,**
- **How to calculate or plot it**,
- **When** should you **use it**.

> **Tip**
>
> If you have read my previous blog post, "24 Evaluation Metrics for Binary Classification (And When to Use Them)", you may want to skip this section and scroll down to the evaluation metrics comparison.

1. Accuracy

It measures how many observations, both positive and negative, were correctly classified.

$$ACC = \frac{tp + tn}{tp + fp + tn + fn}$$

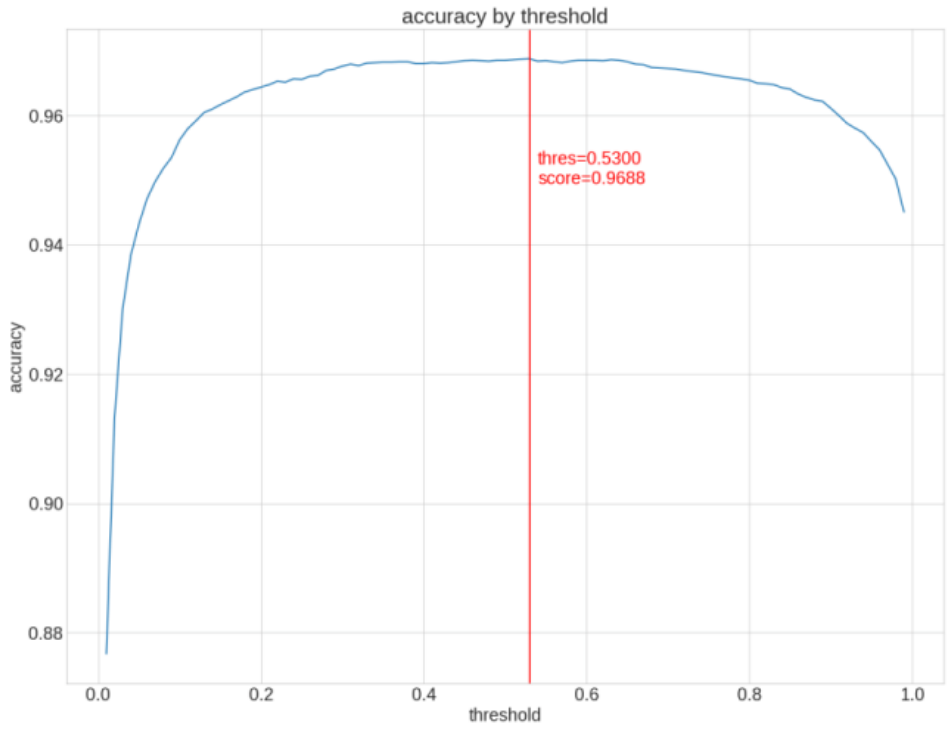
You **shouldn't use accuracy on imbalanced problems**. Then, it is easy to get a high accuracy score by simply classifying all observations as the majority class.

Table of contents

```
from sklearn.metrics import confusion_matrix,
accuracy_score

y_pred_class = y_pred_pos > threshold
tn, fp, fn, tp = confusion_matrix(y_true,
y_pred_class).ravel()
accuracy = (tp + tn) / (tp + fp + fn + tn)

# or simply

accuracy_score(y_true, y_pred_class)
```

Since the accuracy score is calculated on the predicted classes (not prediction scores) we **need to apply a certain threshold** before computing it. The obvious choice is the threshold of 0.5 but it can be suboptimal.

Let's see an example of **how accuracy depends on the threshold** choice:



Accuracy by threshold

You can use charts like the one above to determine the optimal threshold. In this case, choosing something a bit over standard 0.5 could bump the score by a tiny bit 0.9686->0.9688 but in other cases, the improvement can be more

Table of contents

So, **when** does it make sense to **use it**?

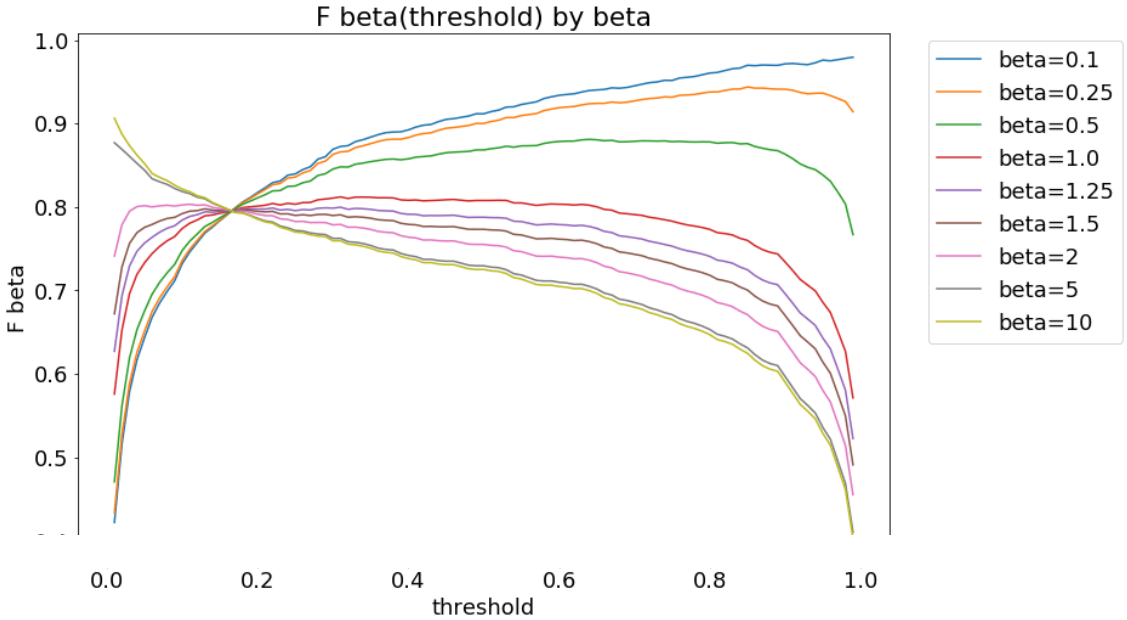
- When your **problem is balanced** using accuracy is usually a good start. An additional benefit is that it is really easy to explain it to non-technical stakeholders in your project,
- When **every class is equally important** to you.

2. F1 score

Simply put, it combines precision and recall into one metric by calculating the harmonic mean between those two. It is actually a **special case of** the more general function **F beta**:

$$F_{beta} = (1+\beta^2)\frac{precision * recall}{\beta^2 * precision + recall}$$

When choosing beta in your F-beta score **the more you care about recall** over precision **the higher beta** you should choose. For example, with F1 score we care equally about recall and precision with F2 score, recall is twice as important to us.



F beta threshold by beta

Table of contents

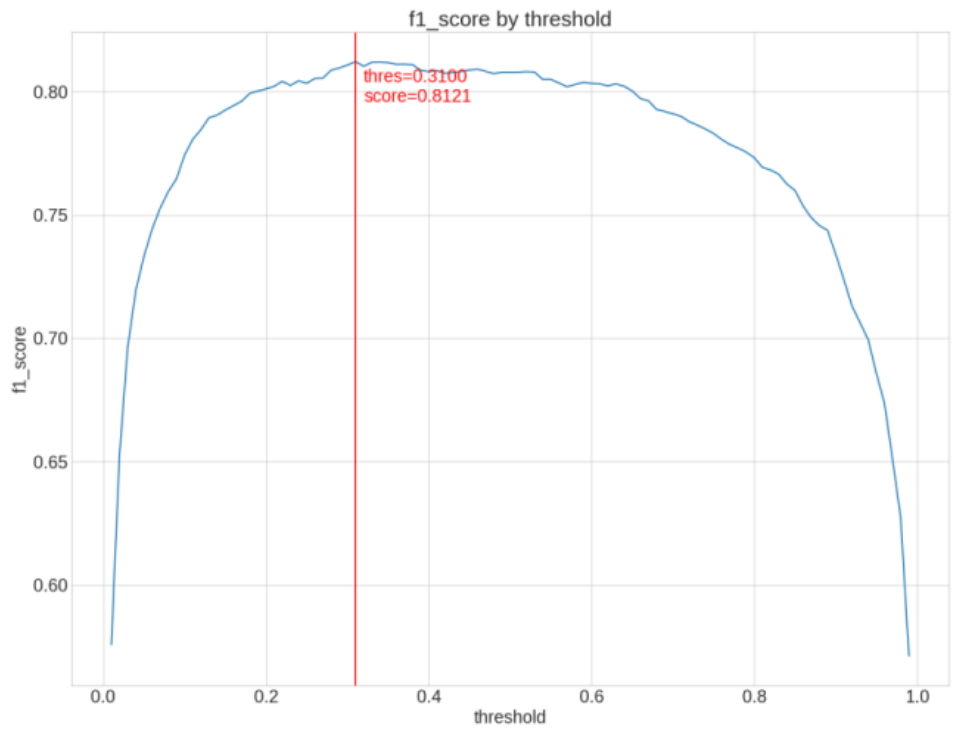
lower thresholds and with beta=1 it is somewhere in the middle.

It can be easily computed by running:

```
from sklearn.metrics import f1_score

y_pred_class = y_pred_pos > threshold
f1_score(y_true, y_pred_class)
```

It is important to remember that F1 score is calculated from Precision and Recall which, in turn, are calculated on the predicted classes (not prediction scores).

How should we choose an optimal threshold? Let's plot F1 score over all possible thresholds:



F1 score by threshold

We can **adjust the threshold to optimize the F1 score**. Notice that for both precision and recall you could get perfect scores by increasing or decreasing the threshold. Good thing is, **you can find a sweet spot** for F1 score. As you

Table of contents

When should you use it?

- Pretty much in every binary classification problem where you care more about the positive class. **It is my go-to metric** when working on those problems.
- It **can be easily explained to business stakeholders** which in many cases can be a deciding factor. Always remember, machine learning is just a tool to solve a business problem.

3. ROC AUC

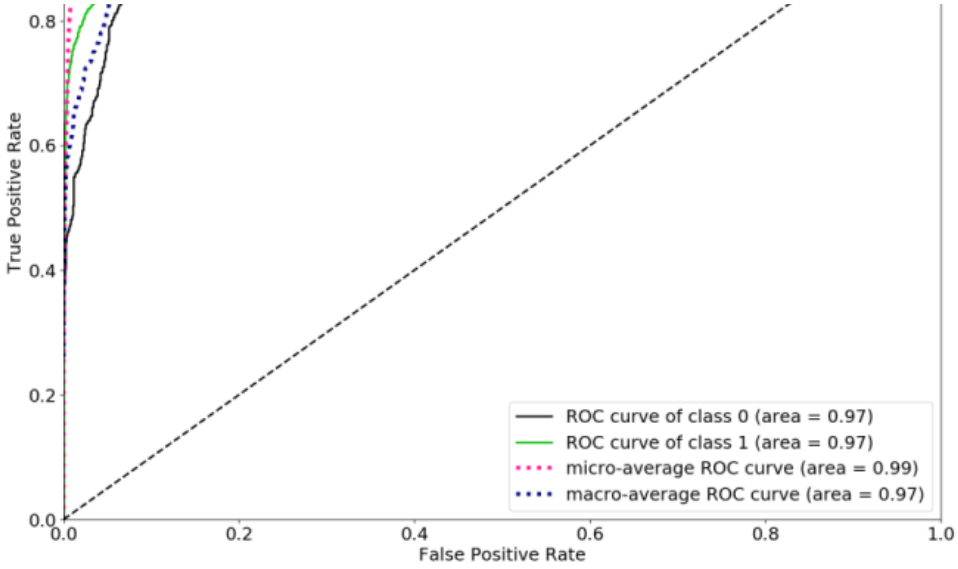
AUC means area under the curve so to speak about ROC AUC score we need to define ROC curve first.

It is a chart that visualizes the tradeoff between true positive rate (TPR) and false positive rate (FPR). Basically, for every threshold, we calculate TPR and FPR and plot it on one chart.

Of course, the higher TPR and the lower FPR is for each threshold the better and so classifiers that have curves that are more top-left-side are better.

An extensive discussion of ROC Curve and ROC AUC score can be found in this article by Tom Fawcett.

Table of contents



ROC curves

We can see a healthy ROC curve, pushed towards the top-left side both for positive and negative classes. It is not clear which one performs better across the board as with FPR < ~0.15 positive class is higher and starting from FPR~0.15 the negative class is above.

In order to get one number that tells us how good our curve is, we can calculate the Area Under the ROC Curve, or ROC AUC score. The more top-left your curve is the higher the area and hence higher ROC AUC score.

Alternatively, it can be shown that ROC AUC score is equivalent to calculating the rank correlation between predictions and targets. From an interpretation standpoint, it is more useful because it tells us that this metric shows **how good at ranking predictions your model is**. It tells you what is the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance.

```
from sklearn.metrics import roc_auc_score

roc_auc = roc_auc_score(y_true, y_pred_pos)
```

- You **should use it** when you ultimately **care about ranking predictions** and not necessarily about outputting well-calibrated probabilities (read

this article by Jason Brownlee if you want to learn about probability calibration).

- You **should not use it** when your **data is heavily imbalanced**. It was

Table of contents

datasets is pulled down due to a large number of true negatives.

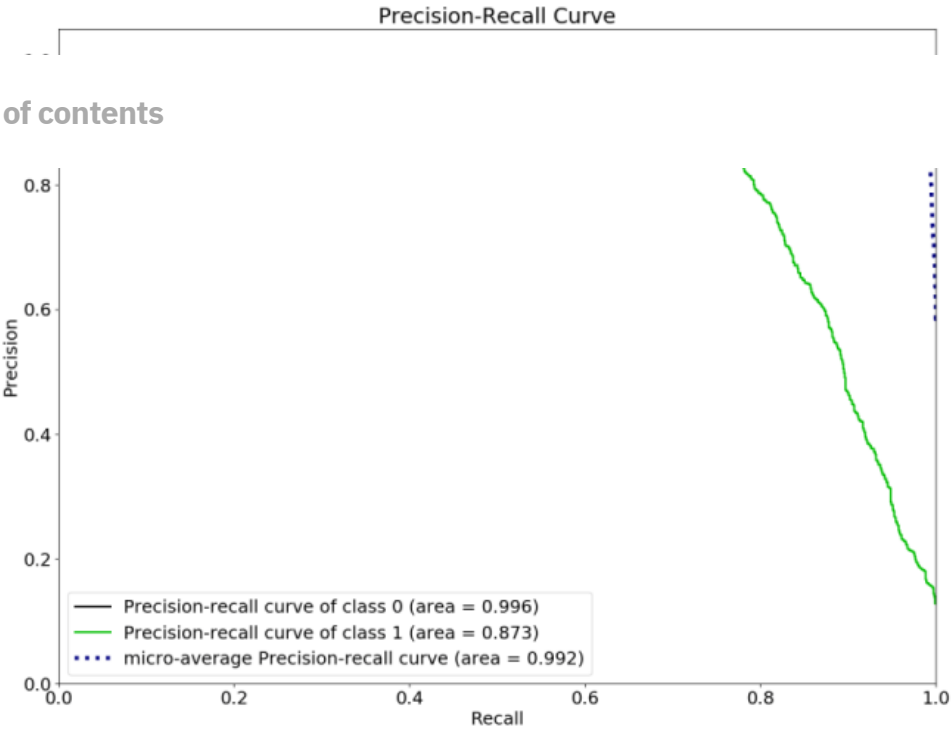
- You **should use it when you care equally about positive and negative classes**. It naturally extends the imbalanced data discussion from the last section. If we care about true negatives as much as we care about true positives then it totally makes sense to use ROC AUC.

4. PR AUC | Average Precision

Similarly to ROC AUC in order to define PR AUC we need to define what Precision-Recall curve.

It is a curve that combines precision (PPV) and Recall (TPR) in a single visualization. For every threshold, you calculate PPV and TPR and plot it. The higher on y-axis your curve is the better your model performance.

You can use this plot to make an educated decision when it comes to the classic precision/recall dilemma. Obviously, the higher the recall the lower the precision. Knowing **at which recall your precision starts to fall fast** can help you choose the threshold and deliver a better model.



Precision-Recall curve

We can see that for the negative class we maintain high precision and high recall almost throughout the entire range of thresholds. For the positive class, precision is starting to fall as soon as we are recalling 0.2 of true positives and by the time we hit 0.8, it decreases to around 0.7.

Similarly to ROC AUC score you can calculate the Area Under the Precision-Recall Curve to get one number that describes model performance.

You can also **think of PR AUC as the average of precision scores calculated for each recall threshold**. You can also adjust this definition to suit your business needs by choosing/clipping recall thresholds if needed.

```
from sklearn.metrics import average_precision_score

average_precision_score(y_true, y_pred_pos)
```

- when you want to **communicate precision/recall decision** to other stakeholders
- when you want to **choose the threshold that fits the business problem**.

When your data is **heavily imbalanced**. As mentioned before, it was discussed extensively in this article by Takaya Saito and Marc Rehmsmeier.

The intuition is the following: since PR AUC focuses mainly on the positive class (PPV and TPR) it cares less about the frequent negative class.

- when **you care more about positive than negative class**. If you care more

Table of contents

Evaluation metrics comparison

We will compare those metrics on a real use case. Based on a recent kaggle competiton I created an example fraud-detection problem:

- I selected only **43 features**
- I sampled **66000 observations** from the original dataset
- I adjusted the **fraction of the positive class to 0.09**
- I trained a bunch of lightGBM classifiers with different hyperparameters.

I wanted to have an intuition as to which models are “truly” better. Specifically, I suspect that the model with only 10 trees is worse than a model with 100 trees. Of course, with more trees and smaller learning rates, it gets tricky but I think it is a decent proxy.

So for combinations of **learning_rate** and **n_estimators**, I did the following:

- defined hyperparameter values:

```
MODEL_PARAMS = {'random_state': 1234,
                'learning_rate': 0.1,
                'n_estimators': 10}
```

- trained the model:

```
model = lightgbm.LGBMClassifier(**MODEL_PARAMS)
model.fit(X_train, y_train)
```

- predicted on test data:

```
y_test_pred = model.predict_proba(X_test)
```

Table of contents


```
y_test_pred = model.predict_proba(X_test)
```


For a full code base go to this repository.

You can also go here and explore experiment runs with:

- evaluation metrics
- performance charts
- metric by threshold plots

Let's take a look at how our models are scoring on different metrics.

Short ID	learning_rate	n_estimators	roc_auc	avg_precision	f1_score	accuracy
BIN-103	0.05	3000	0.965006	0.871118	0.804844	0.96827
BIN-102	0.05	1500	0.964014	0.865733	0.79717	0.967437
BIN-101	0.1	1500	0.965926	0.873456	0.807781	0.968573
BIN-100	0.1	600	0.964205	0.864721	0.790019	0.966225
BIN-99	0.1	300	0.96103	0.845597	0.761358	0.962211
BIN-98	0.1	10	0.9198	0.694194	0.446406	0.93351
BIN-97	0.1	100	0.951547	0.806408	0.711948	0.956002

Runs table | See in Neptune

On this problem, all of those metrics are ranking models from best to worst very similarly but there are slight differences. Also, the scores themselves can vary greatly.

In the next sections, we will discuss it in more detail.

5. Accuracy vs ROC AUC

The first big difference is that you **calculate accuracy on the predicted classes** while you **calculate ROC AUC on predicted scores**. That means you will have to find the optimal threshold for your problem.

Table of contents

negative classes. That means if our **problem is highly imbalanced** we get a really **high accuracy score** by simply predicting that **all observations belong to the majority class.**

On the flip side, if your problem is **balanced** and you **care about both positive and negative predictions**, **accuracy is a good choice** because it is really simple and easy to interpret.

Another thing to remember is that **ROC AUC is especially good at ranking** predictions. Because of that, if you have a problem where sorting your observations is what you care about ROC AUC is likely what you are looking for.

Now, let's look at the results of our experiments:

Short ID	learning_rate	n_estimators	roc_auc	accuracy
BIN-101	0.1	1500	0.965926	0.968573
BIN-103	0.05	3000	0.965006	0.96827
BIN-100	0.1	600	0.964205	0.966225
BIN-102	0.05	1500	0.964014	0.967437
BIN-99	0.1	300	0.96103	0.962211
BIN-97	0.1	100	0.951547	0.956002
BIN-98	0.1	10	0.9198	0.93351

Experiments sorted by ROC AUC score | See in Neptune

The first observation is that models rank almost exactly the same on ROC AUC and accuracy.

0.97 for the best one. Remember that predicting all observations as majority class 0 would give 0.9 accuracy so our worst experiment BIN-98 is only

slightly better than that. Yet the score itself is quite high and it shows that **you should always take an imbalance into consideration when looking at accuracy.**

Table of contents

Note:

→ There is an interesting metric called Cohen Kappa that takes imbalance into consideration by calculating the improvement in accuracy over the "sample according to class imbalance" model.

Read more about Cohen Kappa here.

6. F1 score vs Accuracy

Both of those metrics take class predictions as input so you will have to adjust the threshold regardless of which one you choose.

Remember that the **F1 score** is balancing precision and recall on the **positive class** while **accuracy** looks at correctly classified observations **both positive and negative**. That makes a **big difference** especially for the **imbalanced problems** where by default our model will be good at predicting true negatives and hence accuracy will be high. However, if you care equally about true negatives and true positives then accuracy is the metric you should choose.

If we look at our experiments below:

Short ID	learning_rate	n_estimators	f1_score	accuracy
BIN-103	0.05	3000	0.804844	0.96827
BIN-102	0.05	1500	0.79717	0.967437
BIN-100	0.1	600	0.790019	0.966225
BIN-99	0.1	300	0.761358	0.962211
BIN-97	0.1	100	0.711948	0.956002
BIN-98	0.1	10	0.446406	0.93351

Table of contents

Experiments sorted by F1 score | See in Neptune

In our example, both metrics are equally capable of helping us rank models and choose the best one. The **class imbalance** of 1-10 **makes our accuracy** really **high** by default. Because of that, even the worst model has very high accuracy and the improvements as we go to the top of the table are not as clear on accuracy as they are on F1 score.

7. ROC AUC vs PR AUC

What is common between ROC AUC and PR AUC is that they both look at prediction scores of classification models and not thresholded class assignments. What is different however is that **ROC AUC looks at** a true positive rate **TPR and** false positive rate **FPR** while **PR AUC looks at** positive predictive value **PPV** and true positive rate **TPR**.

Because of that **if you care more about the positive class, then using PR AUC**, which is more sensitive to the improvements for the positive class, is a better choice. One common scenario is a highly imbalanced dataset where the fraction of positive class, which we want to find (like in fraud detection), is small. I highly recommend taking a look at this kaggle kernel for a longer discussion on the subject of ROC AUC vs PR AUC for imbalanced datasets

If you care equally about the positive and negative class or your dataset is quite balanced, then going with **ROC AUC** is a good idea.

Table of contents

Short ID	learning_rate	n_estimators	roc_auc	avg_preci…
BIN-101	0.1	1500	0.965926	0.873456
BIN-103	0.05	3000	0.965006	0.871118
BIN-100	0.1	600	0.964205	0.864721
BIN-102	0.05	1500	0.964014	0.865733
BIN-99	0.1	300	0.96103	0.845597
BIN-97	0.1	100	0.951547	0.806408
BIN-98	0.1	10	0.9198	0.694194

Experiments sorted by ROC AUC score | See in Neptune

They rank models similarly but there is a slight difference if you look at experiments BIN-100 and BIN 102.

However, the improvements calculated in Average Precision (PR AUC) are larger and clearer. We get from 0.69 to 0.87 when at the same time ROC AUC goes from 0.92 to 0.97. Because of that ROC AUC can give a **false sense of very high performance** when in fact your model can be doing not that well.

8. F1 score vs ROC AUC

One big difference between F1 score and ROC AUC is that the first one takes predicted classes and the second takes predicted scores as input. Because of that, **with F1 score you need to choose a threshold** that assigns your observations to those classes. Often, you can improve your model performance by a lot if you choose it well.

So, **if you care about ranking predictions**, don't need them to be properly calibrated probabilities, and your dataset **is not heavily imbalanced** then I would go with **ROC AUC**.

Table of contents

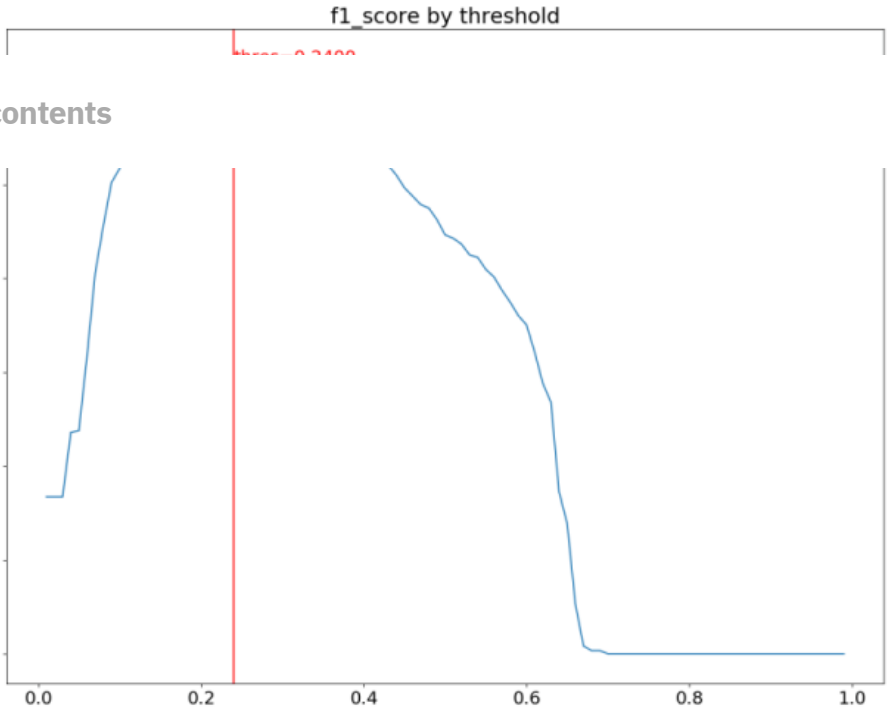
positive class, I'd consider using **F1 score**, or Precision-Recall curve and **PR AUC**. The additional reason to go with F1 (or Fbeta) is that these metrics are **easier to interpret and communicate to business** stakeholders.

Let's take a look at the experimental results for some more insights:

Short ID	learning_rate	n_estimators	f1_score	roc_auc
BIN-101	0.1	1500	0.807781	0.965926
BIN-103	0.05	3000	0.804844	0.965006
BIN-102	0.05	1500	0.79717	0.964014
BIN-100	0.1	600	0.790019	0.964205
BIN-99	0.1	300	0.761358	0.96103
BIN-97	0.1	100	0.711948	0.951547
BIN-98	0.1	10	0.446406	0.9198

Experiments sorted by F1 score | See in Neptune

Experiments rank identically on F1 score (*threshold=0.5*) and ROC AUC. However, the F1 score is lower in value and the difference between the worst and the best model is larger. For the ROC AUC score, values are larger and the difference is smaller. Especially interesting is the experiment BIN-98 which has F1 score of 0.45 and ROC AUC of 0.92. The reason for it is that the threshold of 0.5 is a really bad choice for a model that is not yet trained (only 10 trees). You could get a F1 score of 0.63 if you set it at 0.24 as presented below:



F1 score by threshold

Table of contents

Note:

→ If you would like to easily log those plots for every experiment I attach a logging helper at the end of this post.

Final thoughts

In this blog post, you've **learned about a few common metrics** used for evaluating binary classification models.

We've discussed how they are defined, how to interpret and calculate them and when should you consider using them.

Finally, **we compared those evaluation metrics** on a real problem and discussed some typical decisions you may face.

With all this knowledge, you're now ready to choose a proper evaluation metric for your next binary classification problem!

Bonus:

Table of contents

threshold charts described in this post

- binary classification metrics cheetsheat which contains information about 24 evaluation metrics for binary classification problems.

Check those out below!

Logging function

You can **log all of those metrics and performance charts** that we covered for your machine learning project and explore them in Neptune using our Python client and integrations (in the example below, I use Neptune-LightGBM integration).

- install the client:

```
pip install -U neptune-lightgbm
```

- import and run:

```
import neptune

run = neptune.init_run(...)
neptune_callback = NeptuneCallback(run=run)

gbm = lgb.train(
        params,
        lgb_train,
        callbacks=[neptune_callback],
)

custom_score = ...

# log score to neptune
run["logs/custom_score"] = custom_score
```

- explore everything in the app.

Table of contents

Binary classification metrics cheatsheet

We've created a nice cheatsheet for you which takes all the content I went over in this blog post and puts it on a few-page, digestible document which you can print and use whenever you need anything binary classification metrics related.

Get your binary classification metrics cheatsheet

Download

Jakub Czakon

Mostly an ML person. Building MLOps tools, writing technical stuff, experimenting with ideas at Neptune.

Follow me on

Read next

Table of contents

Models and Algorithms

Machine learning has expanded rapidly in the last few years. Instead of simple, one-directional, or linear ML pipelines, today data scientists and developers run multiple parallel experiments that can get overwhelming even for large teams. Each experiment is expected to be recorded in an immutable and reproducible format, which results in endless logs with invaluable details.

We need to narrow down on techniques by comparing machine learning models thoroughly with parallel experiments. Using a well-planned approach is necessary to understand how to choose the right combination of algorithms and the data at hand.

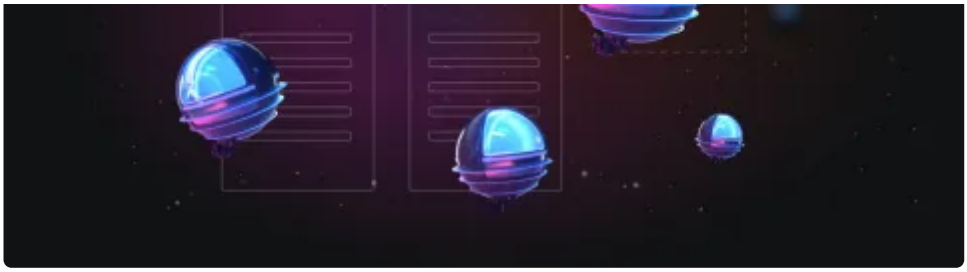
So, in this article, we're going to explore how to approach comparing ML models and algorithms.

Comparing ML models is part of the broader process of tracking ML experiments.

Other than that, experiment tracking is about storing all the important data and metadata, debugging model training, and, generally, analyzing the results of experiments.

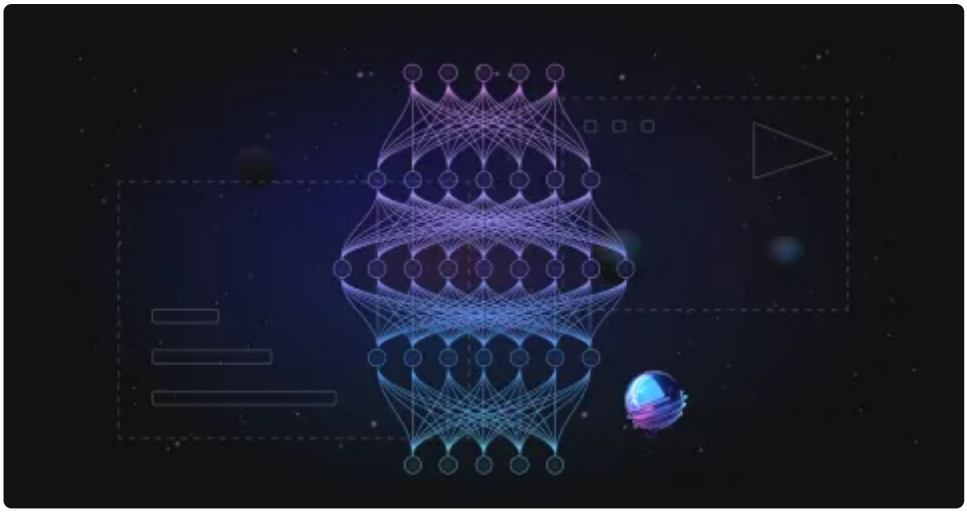
—

Table of contents



How to Build ML Model Training Pipeline

by Henrique Pett, 10 min read



What Does GPT-3 Mean For the Future of MLOps? With David Hershey

by Kurtis Pykes, 23 min read

Read more →

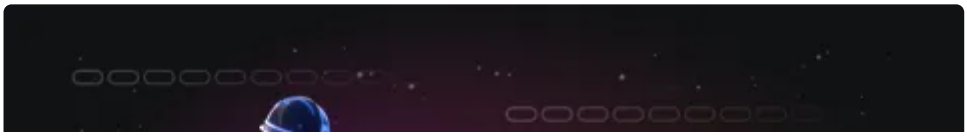
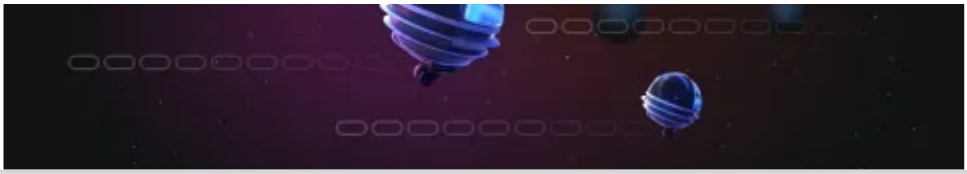
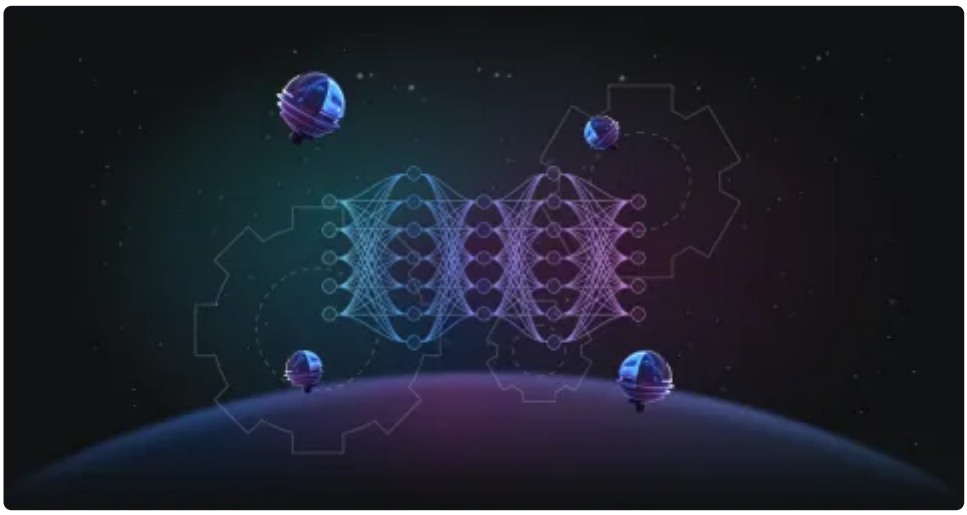


Table of contents





How to Build ETL Data Pipeline in ML

by Natasha Sharma, 7 min read

Read more →

Newsletter

Top MLOps articles, case studies, events (and more) in your inbox every month.

Your e-mail

Get Newsletter

Table of contents

PRODUCT

DOCUMENTATION

COMPARE

COMMUNITY

COMPANY

The Best MLOps Tools MLOps at a Reasonable Scale ML Metadata Store
MLOps: What, Why, and How Experiment Tracking in Machine Learning