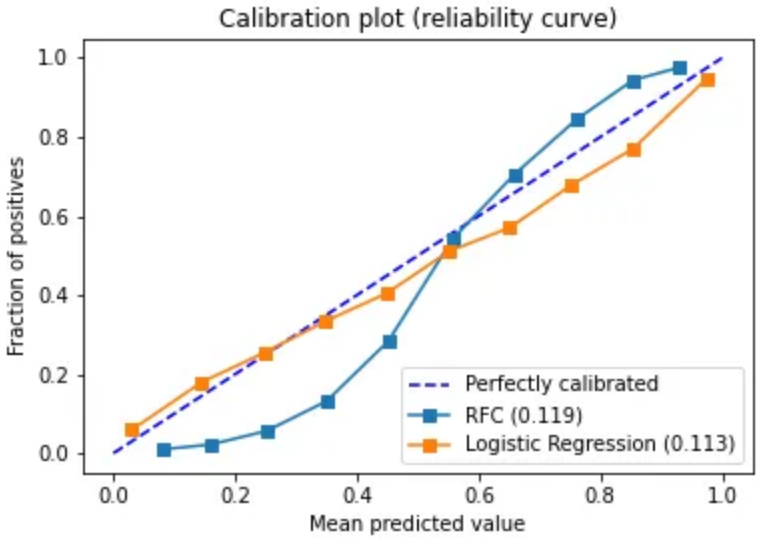
Member-only story

How Probability Calibration Works

Mattia Cinelli · Following
Published in Analytics Vidhya · 6 min read · May 28, 2020



Probability calibration is the process of calibrating an ML model to return the *true* likelihood of an event. This is necessary when we need the probability of the event in question rather than its classification.

Image that you have two models to predict rainy days, Model A and Model B. Both models have an accuracy of 0.8. And indeed, for every 10 rainy days,

both mislabelled two days.

But if we look at the probability connected to each prediction, we can see that Model A reports a probability of 80%, while Model B of 100%.

This means that model B is 100% sure that it will rain, even when it will not, while model A is only 80% sure. It appears that model B is overconfident with its prediction, while model A is more cautious.

And it's this level of confidence in predictions that makes Model A a more reliable model with respect to Model B; Model A is better despite the two models having the same accuracy.
Model B offers a more yes-or-no prediction, while Model A tells us the ***true likelihood of the event***. And in real life, when we look at the weather forecast, we get the prediction and its probability, leaving us to decide if, for example, a 30% risk of rain is acceptable or not.

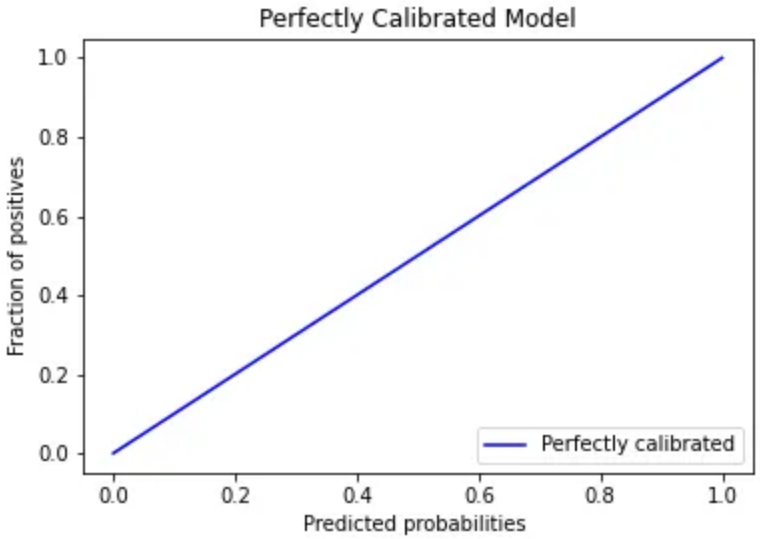
In data science, you will not always care about the probability associated with each prediction, but there are cases in which we want to know the probability.

Think, for example, in the medical sector, where a doctor has to decide if a patient labelled as needing an operation would really benefit from it. Or how safe an investment is, and so on.

Therefore, in these situations it is important to have a model with good accuracy but also that is well **calibrated**!

A model would be perfectly calibrated when the predicted probability reflects the true likelihood of the prediction [1].

Thus, like in our Model A example, the probability of 0.8 is equal to the accuracy for all those events with the same prediction probability.Therefore, there will be an x=y relationship between the predicted probabilities and the fraction of positive entries in the data. This can be visualized with the following plot:



> The perfectly calibrated line of an ideal model. A model is perfectly calibrated if, for any p, a prediction of a class with confidence p is correct 100*p% of the time.[2]

Naturally, our models will rarely be so perfect. Therefore, we need to understand how to see the prediction probability, how to numerically quantify it, and how to calibrate our model in order to improve it.

How to find the prediction probability?

Let's now imagine a model for a binary classification:

```
# Create dataset of classification task with many redundant and few
# informative features

import numpy as np
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import brier_score_loss, precision_score, recall_score, f1_score
np.random.seed(0)

X, y = datasets.make_classification(n_samples=100000, n_features=20, n_informative=2, n_redundan

train_samples = 100
X_train = X[:train_samples]
X_test = X[train_samples:]
y_train = y[:train_samples]
y_test = y[train_samples:]

model_rfc = RandomForestClassifier()
model_rfc.fit(X_train, y_train)
y_pred_rfc = model_rfc.predict(X_test)

print("Precision: %0.2f" % precision_score(y_test, y_pred_rfc))
print("Recall: %0.2f" % recall_score(y_test, y_pred_rfc))
print("F1: %0.2f\n" % f1_score(y_test, y_pred_rfc))

model_lr = LogisticRegression()
model_lr.fit(X_train, y_train)
y_pred_lr = model_lr.predict(X_test)

print("Precision: %0.2f" % precision_score(y_test, y_pred_lr))
print("Recall: %0.2f" % recall_score(y_test, y_pred_lr))
print("F1: %0.2f" % f1_score(y_test, y_pred_lr))

# Precision: 0.85
# Recall: 0.92
# F1: 0.89

# Precision: 0.84
# Recall: 0.86
# F1: 0.85
```

probability_calibration.py hosted with ❤ by GitHub view raw

The random forest classifier (RFC) got an F1 score of 0.89, which is not bad. The logistic regression performed just a bit worse than RF with a score of 0.85. But how well calibrated are they?

To do so, we can use the out-of-the-box function predict_proba().

Note:not all ML algorithms have this function. For others we need the decision_function() (see Comparison of Calibration of Classifiers for details).

In the case of RandomForestClassifier, we can use predict_proba() and obtain the predicted class probability for each input sample.

The result would be something like this:

```
model.predict_proba(X_test)[:, 1]
# array([0.18, 0.84, 0.91, ..., 0.38, 0.54, 0.84])
```

This is not enough if we want to see the relationship between probability and positive prediction. For that,we need to use the function calibration_curve.

This function takes in as input the true label used in the test and the result of predict_proba(), and the number of bins we want to use to summarise the result (default 10).

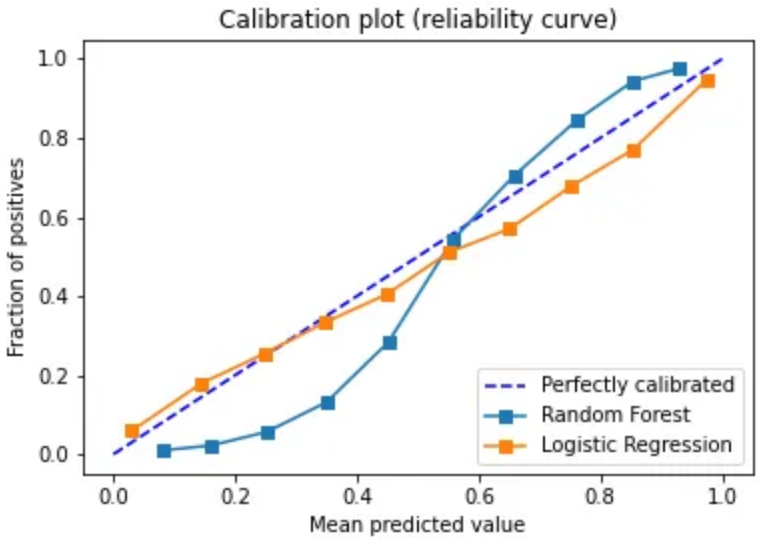
Calibration_curve returns two results: the fraction of positives, that is, the proportion of positive class samples, per bin, and the mean predicted value, thus the average value of all predicted probability in each bin.
If we plot these two results, we will obtain a **calibration plot.**

Calibration plot

Let's see an example of a calibration plot:

```python
import matplotlib.pyplot as plt
from sklearn.calibration import CalibratedClassifierCV, calibration_curve

fig = plt.figure()
ax = fig.add_subplot()
ax.plot([0, 1], [0, 1], "b--", label="Perfectly calibrated")
ax.set_ylabel("Fraction of positives")
ax.set_xlabel("Mean predicted value")
ax.set_title('Calibration plot (reliability curve)')

prob_pos_rfc = model_rfc.predict_proba(X_test)[:, 1]
fraction_of_positives_rfc, mean_predicted_value_rfc = calibration_curve(y_test, prob_pos_rfc, n_
ax.plot(mean_predicted_value_rfc, fraction_of_positives_rfc, "s-", label="%s" % ('Random Forest'

prob_pos_lr = model_lr.predict_proba(X_test)[:, 1]
fraction_of_positives_lr, mean_predicted_value_lr = calibration_curve(y_test, prob_pos_lr, n_bin
ax.plot(mean_predicted_value_lr, fraction_of_positives_lr, "s-", label="%s" % ('Logistic Regress

ax.legend(loc="lower right")
plt.show()
fig.savefig('02_rf_lr.png', tight=True, quality=100)
```

probability_calibration_2.py hosted with ❤ by GitHub view raw



The plot above is commonly referred to as the **calibration plot** or **reliability diagram (or curve).** In our example, it contains calibration curves for the random forest and logistic regression classifiers, as well as the diagonal dotted line that represents the perfectly calibrated model.

On the x-axis, is the mean predicted value for each bin, and on the y-axis is the fraction of positives.

The random forest has its typical sigmoid shape;the RFC is overconfident in its prediction, pushing the probability toward zero and one. Logistic regression on the other hand usually returns a well calibrated prediction. Therefore, it is usually plotted together with other methods even if it is not directly in use.

Note: When a model goes **below the diagonal**, the model is *over-forecasting*. **Above the diagonal**, the model is *under-forecasting*.

Now, we have a visual representation of the calibration curves, but how can we decide which model is better calibrated? For this we need the Brier Score!

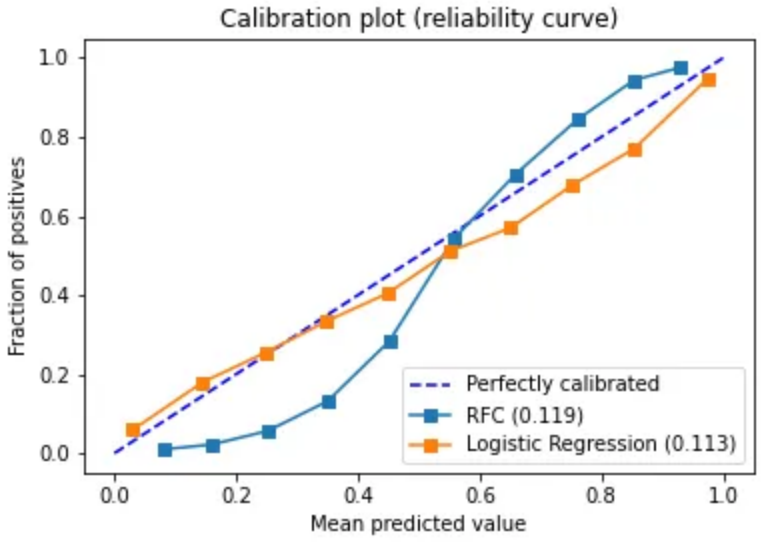
Brier score

The Brier score loss (wiki) is a single specific number that we can refer to, to assess how good our model is. It is a loss score so the smaller it is, the more calibrated the model.

It is computed with the following method, brier_score_loss(y_test, prob_pos). Let's see an example:

```
fig = plt.figure()
ax = fig.add_subplot()
ax.plot([0, 1], [0, 1], "b--", label="Perfectly calibrated")
ax.set_ylabel("Fraction of positives")
ax.set_xlabel("Mean predicted value")
ax.set_title('Calibration plot (reliability curve)')

model_rfc_score = brier_score_loss(y_test, y_pred_rfc, pos_label=y.max())
fraction_of_positives, mean_predicted_value = calibration_curve(y_test, prob_pos_rfc, n_bins=10)

ax.plot(mean_predicted_value, fraction_of_positives, "s-", label="%s (%0.3f)" % ('RFC', model_rf
print("Brier score RFC: %0.2f" % brier_score_loss(y_test, y_pred_rfc))


model_lr_score = brier_score_loss(y_test, prob_pos_lr, pos_label=y.max())
fraction_of_positives, mean_predicted_value = calibration_curve(y_test, prob_pos_lr, n_bins=10)

ax.plot(mean_predicted_value, fraction_of_positives, "s-", label="%s (%0.3f)" % ('Logistic Regre
print("Brier score LR: %0.2f" % brier_score_loss(y_test, y_pred_lr))

ax.legend(loc="lower right")
plt.show()
fig.savefig('03_rf_lr.png', tight=True, quality=100)
# Brier score RFC: 0.12
# Brier score LR: 0.11
```

probability_calibration_3.py hosted with ❤ by GitHub view raw



In our example, it returns a value of 0.12 for random forest and 0.11 for logistic regression.

Now we have a system to evaluate the calibration curve. Let's see how we can improve it. For this we need to **calibrate** the model.

Probability calibration

The probability calibration of a model is a re-scaling of the model, it can be done using the scikit function CalibratedClassifierCV

There are two arguments of the function we have to consider: the **methods** and the **validations**:

1. **Two methods of calibration:**

- *Sigmoid scaling or Platt's method*. This is suitable for models with a sigmoid curve (like RFC) and it works well with small datasets.
- *Isotonic regression* — a non-parametric approach. It is a more powerful calibration method but it tends to overfit and is not advised for small datasets.

2. Two possible validations:

- **Before**, using classical cross validation:

```
X_train, X_test, y_train, y_test = train_test_split(...)
model = ...
calibrator = CalibratedClassifierCV(model, cv=3)
calibrator.fit(trainX, trainy)
predictions = calibrator.predict(testX)
```

probability_calibration_before.py hosted with ❤ by GitHub view raw

- **After** the model is trained using "prefit" model. This is usually more suitable for larger datasets:

```
X_train, X_train2, y_train, X_test, X_test2, y_test = ...
model = ...
model.fit(trainX, trainy)
calibrator = CalibratedClassifierCV(model, cv='prefit')
calibrator.fit(valX, valy)
predictions = calibrator.predict(testX)
```

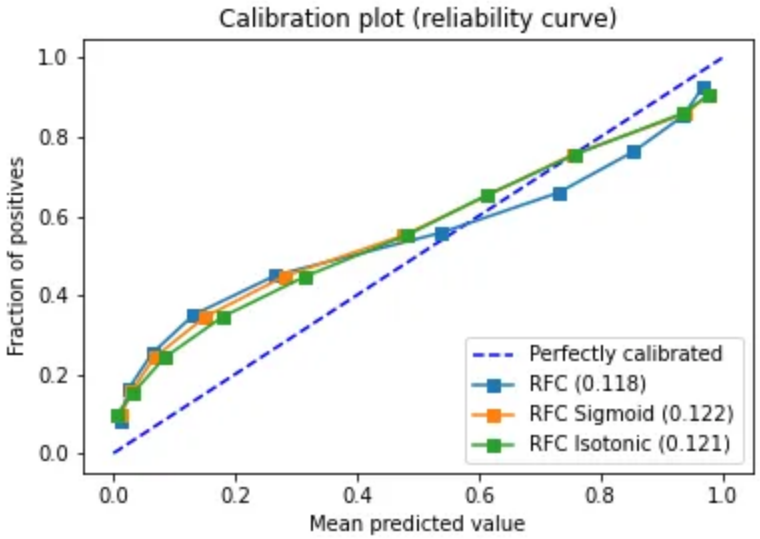
probability_calibration_after.py hosted with ❤ by GitHub view raw

Let's see an example:

Cross Validation

```python
def get_calibration_curve_values(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    prob_pos = model.predict_proba(X_test)[:, 1]
    model_score = brier_score_loss(y_test, y_pred, pos_label=y.max())

    print("F1: %0.2f" % f1_score(y_test, y_pred))
    fraction_of_positives, mean_predicted_value = calibration_curve(y_test, prob_pos, n_bins=10)

    return(fraction_of_positives, mean_predicted_value, model_score)

fig = plt.figure()
ax = fig.add_subplot()
ax.plot([0, 1], [0, 1], "b--", label="Perfectly calibrated")
ax.set_ylabel("Fraction of positives")
ax.set_xlabel("Mean predicted value")
ax.set_title('Calibration plot (reliability curve)')

model_rfc = RandomForestClassifier()
mpv, fp, score = get_calibration_curve_values(model_rfc, X_train, y_train, X_test, y_test)
ax.plot(mpv, fp, "s-", label="%s (%1.3f)" % ('RFC', score))

calibrator_sigmoid = CalibratedClassifierCV(RandomForestClassifier(), cv=5, method='sigmoid')
mpv, fp, score = get_calibration_curve_values(calibrator_sigmoid, X_train, y_train, X_test, y_te
ax.plot(mpv, fp, "s-", label="%s (%1.3f)" % ('RFC Sigmoid', score))

calibrator_isotonic = CalibratedClassifierCV(RandomForestClassifier(), cv=5, method='isotonic')
mpv, fp, score = get_calibration_curve_values(calibrator_sigmoid, X_train, y_train, X_test, y_te
ax.plot(mpv, fp, "s-", label="%s (%1.3f)" % ('RFC Isotonic', score))

ax.legend(loc="lower right")
plt.show()
fig.savefig('04_before.png', tight=True, quality=100)
# F1: 0.89
# F1: 0.88
# F1: 0.88
```

probability_calibration_4.py hosted with ❤ by GitHub view raw

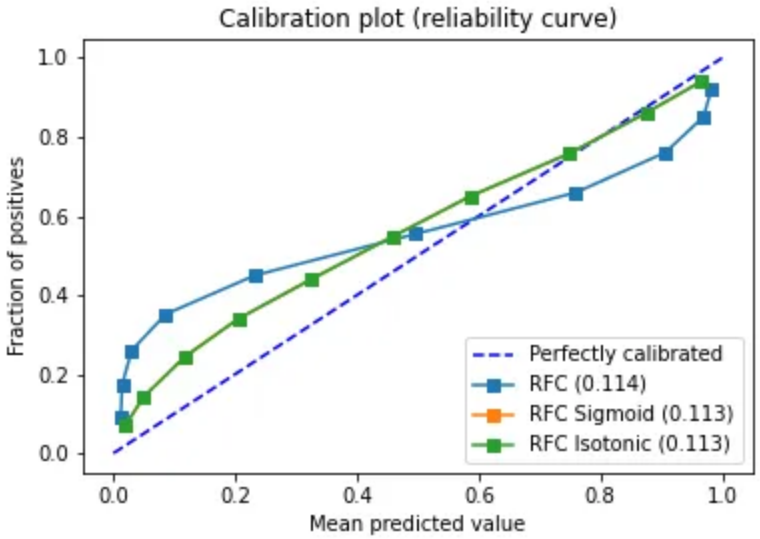


The F1 score of this method has not changed but the Brier score has indeed improved.

Prefit

```
2
3   fig = plt.figure()
4   ax = fig.add_subplot()
5   ax.plot([0, 1], [0, 1], "b--", label="Perfectly calibrated")
6   ax.set_ylabel("Fraction of positives")
7   ax.set_xlabel("Mean predicted value")
8   ax.set_title('Calibration plot (reliability curve)')
9
10  X_train, X_train2, y_train, y_train2 = train_test_split(X_train, y_train, test_size=0.5)
11
12  model_rfc = RandomForestClassifier()
13  mpv, fp, score = get_calibration_curve_values(model_rfc, X_train, y_train, X_test, y_test)
14  ax.plot(mpv, fp, "s-", label="%s (%1.3f)" % ('RFC', score))
15
16  calibrator_sigmoid = CalibratedClassifierCV(model_rfc, cv='prefit', method='sigmoid')
17  mpv, fp, score = get_calibration_curve_values(calibrator_sigmoid, X_train2, y_train2, X_test, y_
18  ax.plot(mpv, fp, "s-", label="%s (%1.3f)" % ('RFC Sigmoid', score))
19
20  calibrator_isotonic = CalibratedClassifierCV(model_rfc, cv='prefit', method='isotonic')
21  mpv, fp, score = get_calibration_curve_values(calibrator_sigmoid, X_train2, y_train2, X_test, y_
22  ax.plot(mpv, fp, "s-", label="%s (%1.3f)" % ('RFC Isotonic', score))
23
24  ax.legend(loc="lower right")
25  plt.show()
26  fig.savefig('05_after.png', tight=True, quality=100)
27  # F1: 0.89
28  # F1: 0.89
29  # F1: 0.89
```

probability_calibration_5.py hosted with ❤ by GitHub view raw



With this method not only the Brier score is improved but also the F1.

> ***Note:*** *In this particular case, the F1 has improved, but it is not always the case! most of the time accuracy and other metrics would decrease in performance. Be aware of this trade-off.*
> *Besides, sigmoid and isotonic calibration overlap, this is rarely the case as well.*

Sources

In order to understand the concept of the Calibrate Probability I had to look at several source, here a list of the ones I rely the most.

1. How to Calibrate Probabilities for Imbalanced Classification
2. Model Calibration — is your model ready for the real world? — Inbar Naor — PyCon Israel 2018
3. Safe Handling Instructions for Probabilistic Classification | SciPy 2019 | Gordon Chen
4. Applied ML 2020–10 — Calibration, Imbalanced data

5. plot_calibration_curve
6. Classifier calibration
7. Calibration of Models — Nupur Agarwal

I wrote this article as a way to organise and test my knowledge on this topic. you find errors, or you find it useful, or have suggestions please leave a comment.

For the code in jupyter click here.

Originally published at https://mattiacinelli.com on May 27, 2020.

Python | Probability | Calibration | Machine Learning | Data Science

More from the list: "ML"

Curated by Himanshu Birla


Kyosuke... in Towards Dat...

Probability Calibration for Imbalanced Dataset


Jason Yo... in Towards Dat...

Why Calibrators? Part 1 of the Series on Probabilit...


Jaideep Ray

Probability c why it matte

✦ · 8 min read · Oct 20, 2019

7 min read · Oct 4, 2020

3 min read · Ju

View list

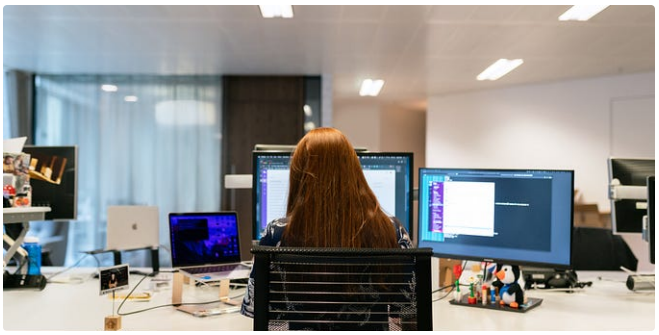
Written by Mattia Cinelli

Following

321 Followers · Writer for Analytics Vidhya

Data Scientist

More from Mattia Cinelli and Analytics Vidhya



Mattia Cinelli in Towards Data Science

SOLID Coding in Python

An overview of the 5 SOLID principles of coding that would improve your Python...

✦ · 8 min read · Jun 29, 2021

1.3K 12

Sajal Rastogi in Analytics Vidhya

Wifi -Hacking using PyWifi

.

4 min read · Feb 6, 2021

259 1

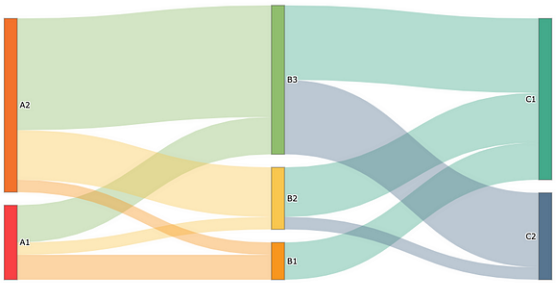
Kia Eisinga in Analytics Vidhya

How to create a Python library

Ever wanted to create a Python library, albeit for your team at work or for some open...

7 min read · Jan 27, 2020

2K 24



Mattia Cinelli in Analytics Vidhya

How to do a Sankey Plot in Python

If you have been more than five seconds on r/dataisbeautiful/, you will have probably...

✦ · 4 min read · May 3, 2020

270 2

See all from Mattia Cinelli | See all from Analytics Vidhya

Recommended from Medium

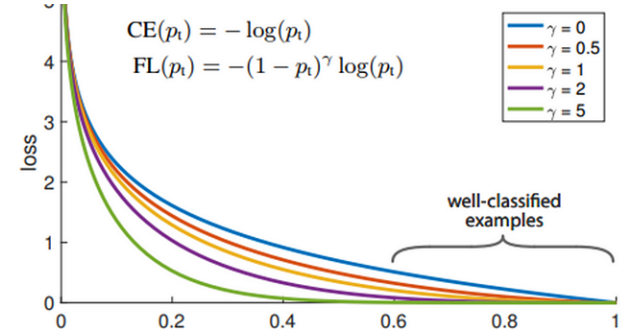
Chandra Prakash Bathula

Machine Learning Concept 68: Platt's Scaling

Platt's Scaling:

3 min read · Apr 13

5



Uman Niyaz in Data Science @ Ecom Express

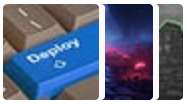
Focal loss for handling the issue of class imbalance

Text classification is widely used in various industries to tackle business challenges by...

9 min read · Jun 12

53

Lists



Predictive Modeling w/ Python

20 stories · 473 saves

Practical Guides to Machine Learning

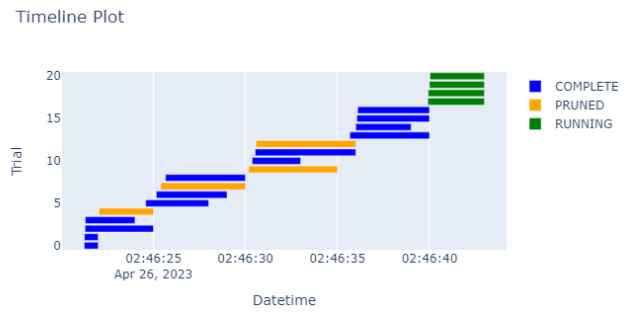
10 stories · 544 saves

Coding & Development

11 stories · 207 saves

New_Reading_List

174 stories · 143 saves



Hiroki Takizawa in Optuna

Introducing Timeline Plot: a new feature of Optuna v3.2

The newest version of Optuna, Optuna v3.2, an open source software tool for black box...

4 min read · Jun 23

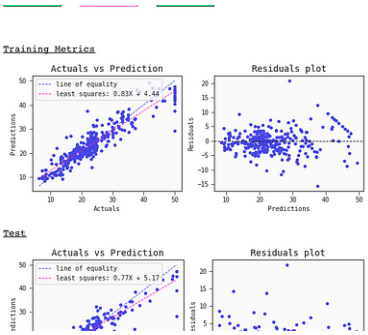
14

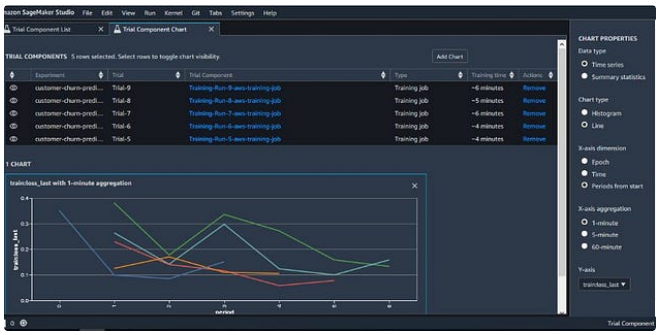
Juan Broglio

Gamma Regression vs Linear Regression (in Python)

General Linear Models and Gamma Regression

4 min read · Aug 9





Casper Skern Wilstrup

Symbolic Regression: a Simple and Friendly Introduction

Symbolic Regression is like a treasure hunt for the perfect mathematical equation to...

3 min read · May 5

18 1

Yugank Aman

Top MLOps Tools to Manage Machine Learning Lifecycle

Businesses continue transforming their operations to increase productivity and...

10 min read · May 30

94