Member-only story

Probability Calibration for Imbalanced Dataset

A suggestion to Undersampling method

Kyosuke Morita · Following

Published in Towards Data Science · 8 min read · Nov 18, 2019

Photo by Bharathi Kannan on Unsplash

When we are trying to build a machine learning model for real-world problems, we are often faced with imbalanced datasets. Resampling method, especially undersampling is one of the most widely used methods to overcome the class imbalance (I also showed how those resampling methods worked on my Master dissertation in the other medium post). However, implementing this method tends to increase false positive due to the different class distribution in train and test set. This biases the classifier and increases false positive. [1] Pozzolo, et al., (2015) argues that we can correct the bias due to the undersampling by using Bayes Minimum Risk theory. This helps us to find the correct classification threshold.

Content

1. How does Probability Calibration work?

Photo by John Schnobrich on Unsplash

As briefly mentioned above, undersampling causes a bias in the posterior probabilities. This is due to the characteristic of random undersampling, which downsizes the majority class by removing them randomly until both classes have the same number of observations. This makes the class distribution of the training set different from the one in the test set. So how exactly probability calibration using Bayes Minimum Risk theory works on this problem? — the basic idea of this method is trying to reduce/remove the bias caused by random undersampling by taking into the undersampling ratio β account. Let's have a look into some definitions:

Let *ps* be the probability of the prediction being a positive class after random undersampling;

$$p_s = p(+|x, s = 1)$$

, and p be the probability of the prediction given features (unbalanced). We can write *ps as a function of p;*

$$p_s = \frac{p}{p + \beta(1 - p)}$$

, where β is a probability of selecting negative class with undersampling, which can be expressed below.

$$\beta = p(s = 1|-)$$

, which can be written

$$\beta = \frac{N^+}{N^-}$$

The equation above can be solved for p and expressed as below.

$$p = \frac{p_s}{p_s + (\frac{1-p_s}{\beta})}$$

So after applying undersampling ratio β, we can calculate for p, which is the unbiased probability.

The threshold for this is can be;

$$\tau = \pi^+$$

, which is a probability of a positive class in a dataset.

$$\pi^+ = p(+)$$

This was a brief introduction of probability calibration by using Bayes Minimum Risk theory. Now we will go on to see how this works in an example with codes.

2. Example


Capital One
VISA

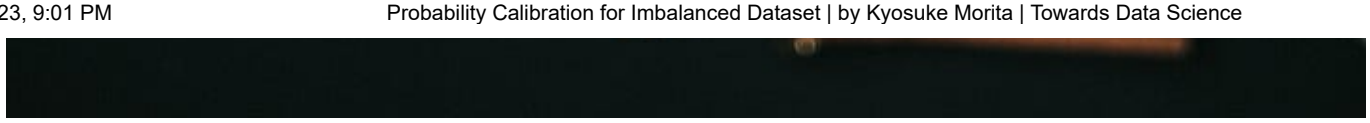


Photo by Nathan Dumlao on Unsplash

In this section, we will see how the probability calibration technique the model performance on a binary classification problem on the famous credit card fraud dataset on Kaggle. This dataset consists of 28 PCA features (all of them are anonymous) plus the Amount feature. The target feature is binary, either fraud or not. The positive class is 0.17% of a whole dataset, which is severely imbalanced. Let's go through the example with codes.

First, import the packages.

```
## config
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import io, os, sys, types, gc, re
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import average_precision_score,
confusion_matrix, precision_score, recall_score,
precision_recall_curve, f1_score, log_loss
from sklearn.decomposition import PCA
pca = PCA(n_components=1,random_state=42)
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=42)
from imblearn.ensemble import BalancedBaggingClassifier
from src.functionalscripts.BMR import *

def make_prediction(model,X,threshold):
    y_pred = model.predict_proba(X)
    y_predicted = np.where(y_pred[:,1]>=threshold,1,0)
    return y_pred, y_predicted
```

```
def evaluation(true, pred):
    print('F1-score: ' + str(round(f1_score(true,pred),4)), '\n'
'Precision: ' + str(round(precision_score(true,pred),4)), '\n'
'Recall: ' + str(round(recall_score(true,pred),4)), '\n'
'Log loss: ' + str(round(log_loss(true,pred),4)), '\n'
'Cohen-Kappa: ' + str(round(cohen_kappa_score(true,pred),4)), '\n'
'Confusion matrix:' + '\n' + str(confusion_matrix(true,pred)))
```

Now read the dataset.

```
# read the data
df = pd.read_csv('src/resources/data/creditcard.csv')
```

Here are the first few rows of the dataset.

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9       V10       V11       V12       V13       V14  \
0  0.098698  0.363787  0.090794 -0.551600 -0.617801 -0.991390 -0.311169
1  0.085102 -0.255425 -0.166974  1.612727  1.065235  0.489095 -0.143772
2  0.247676 -1.514654  0.207643  0.624501  0.066084  0.717293 -0.165946
3  0.377436 -1.387024 -0.054952 -0.226487  0.178228  0.507757 -0.287924
4 -0.270533  0.817739  0.753074 -0.822843  0.538196  1.345852 -1.119670

        V15       V16       V17       V18       V19       V20       V21  \
0  1.468177 -0.470401  0.207971  0.025791  0.403993  0.251412 -0.018307
1  0.635558  0.463917 -0.114805 -0.183361 -0.145783 -0.069083 -0.225775
2  2.345865 -2.890083  1.109969 -0.121359 -2.261857  0.524980  0.247998
3 -0.631418 -1.059647 -0.684093  1.965775 -1.232622 -0.208038 -0.108300
4  0.175121 -0.451449 -0.237033 -0.038195  0.803487  0.408542 -0.009431

        V22       V23       V24       V25       V26       V27       V28  \
0  0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
1 -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2  0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
3  0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
4  0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153

   Amount  Class
0  149.62      0
1    2.69      0
2  378.66      0
3  123.50      0
4   69.99      0
>>>
```

First few rows of the dataset

Class column is our target variable and Amount column is the transaction amount. Now see the ratio of the positive class.

```
# The percentage of positive class in this dataset
len(df[df['Class']==1])/len(df)
```

The positive class is as I mentioned above, 0.17%. Now we will move on to the model building. We will use logistic regression just to see. Let's normalise the Amount column and split into train and test dataset.

```
# Normalise the Amount feature
df['amt'] =
preprocessing.normalize(np.array(df['Amount']).reshape(-1,1),
norm='l2')
```

Now, the data are ready, let's split into train (80% of whole data) and test (20% of whole data) dataset and implement random undersampling.

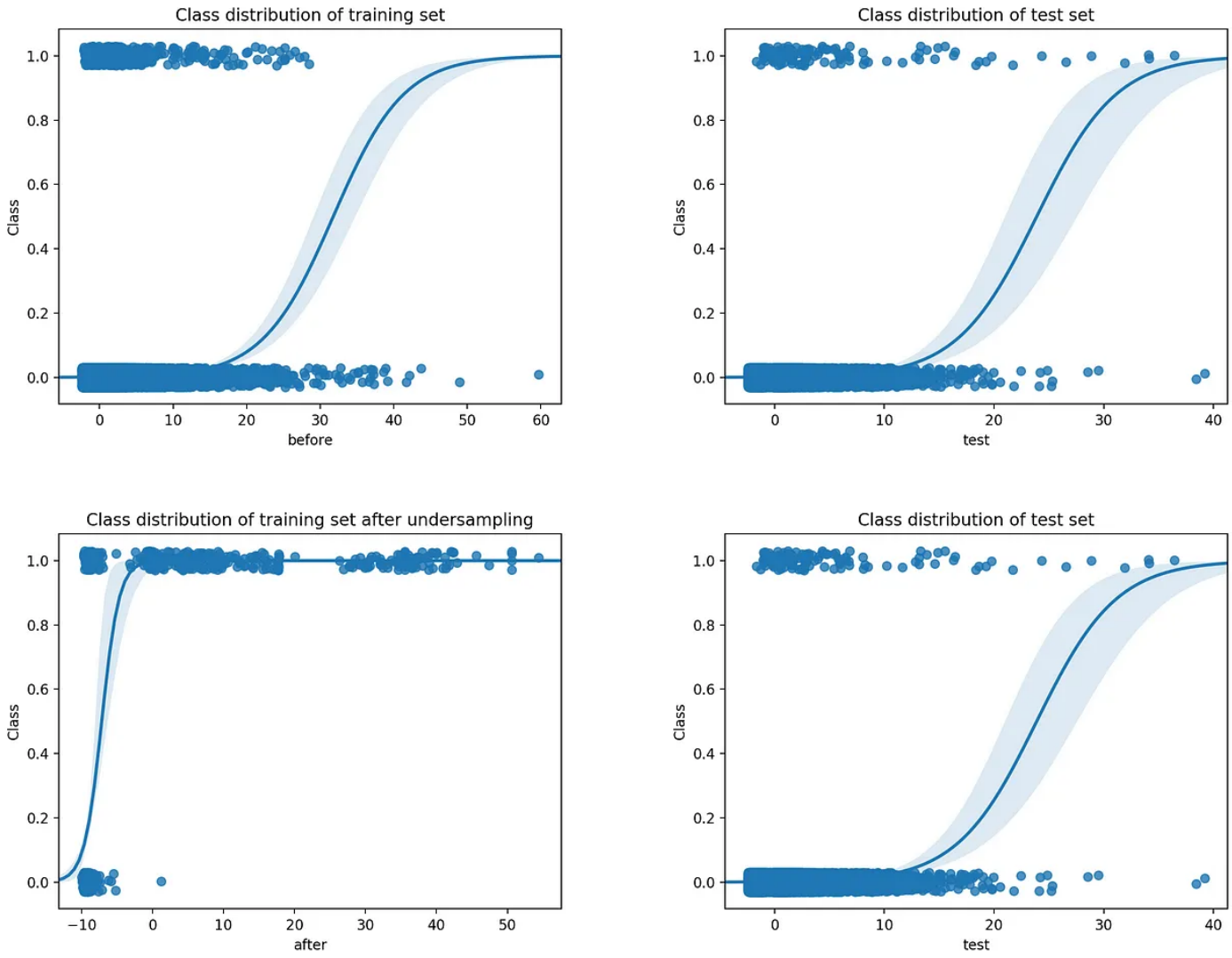
```
# Split the dataset into train and test, and drop unnecessary
features
tr, te = train_test_split(df.drop(['Amount','Time'],1),
test_size=.2,random_state=42)

# Rundom Under Sampling (RUS)
tr_x_rus, tr_y_rus = rus.fit_resample(tr.drop(['Class'],1),tr.Class)
```

We can check the distribution of classes in train, test and train set after random undersampling.

```
# See the class distribution with features
feats_distribution_rus =
pd.DataFrame(pca.fit_transform(tr_x_rus),columns=
['after']).reset_index(drop=True)
feats_distribution_rus['Class'] = tr_y_rus

sns.regplot(x='after',y='Class',data=feats_distribution_rus,logistic=
True, n_boot=500, y_jitter=.03)
plt.title('Class distribution of training set after undersampling')
plt.show()
```



Class distributions of before and after undersampling and test set comparison

We have the distribution of train set in the left upper corner, distribution after random undersampling in the left down corner and right-hand side are

both test set (for comparison). y-axis shows the class and 1 means the transaction was a fraud and otherwise not a fraud. We can see similar distributions in train and test set as they were created by random sampling. On the other hand, in the train set after random undersampling demonstrates quite different distribution from others. Let's see how this difference impact on the classifiers.

Now build models. For the comparison, let's see the performance of RUS bagging (Random Undersampling + bagging) as well.

```
# Logistic regression
logit = LogisticRegression(random_state=42,solver='lbfgs',
max_iter=1000)

# Rundom Under Sampling (RUS)
tr_x_rus, tr_y_rus = rus.fit_resample(tr.drop(['Class'],1),tr.Class)
logit_rus = logit.fit(tr_x_rus,tr_y_rus)

# RUS bagging
bc = BalancedBaggingClassifier(base_estimator=logit,random_state=42)
logit_bc = bc.fit(tr.drop(['Class'],1),tr.Class)
```

Now we implement the probability calibration method using Bayes Minimum Risk. Here we create beta (minority selection ratio), tau (threshold) and calibration functions.

```
# BMR (Bayes Minimum Risk) implementation
# Pozzolo et al., 2015, Calibrating Probability with Undersampling

class BMR:
    def beta(binary_target):
        return binary_target.sum()/len(binary_target)
```

```
    def tau(binary_target, beta):
        return binary_target.sum()/len(binary_target)

    def calibration(prob, beta):
        return prob/(prob+(1-prob)/beta)
```

Apply those calibration techniques to both predicted probabilities by RUS and RUS Bagging.

```
# Calibration
beta = BMR.beta(tr.Class)
tau = BMR.tau(tr.Class,beta)

# with RUS
y_pred_calib_rus =
BMR.calibration(prob=logit_rus.predict_proba(te.drop(['Class'],1))
[:,1],beta=beta)

y_predicted_calib_rus = np.where(y_pred_calib_rus>=tau,1,0)

# wtih RUS bagging
y_pred_calib_bc =
BMR.calibration(prob=logit_bc.predict_proba(te.drop(['Class'],1))
[:,1],beta=beta)

y_predicted_calib_bc = np.where(y_pred_calib_bc>=tau,1,0)
```

Now we have all the predictions, let's evaluate them and see how are their performance. I set the thresholds for RUS and RUS bagging model as 0.5.

```
# Evaluation
## Random Under Sampling (RUS)
y_pred_rus, y_predicted_rus = make_prediction(model=logit_rus,
X=te.drop(['Class'],1), threshold=.5)
evaluation(te.Class, y_predicted_rus)

## RUS + Bagging
y_pred_bc, y_predicted_bc = make_prediction(model=logit_bc,
X=te.drop(['Class'],1), threshold=.5)
evaluation(te.Class, y_predicted_bc)
```

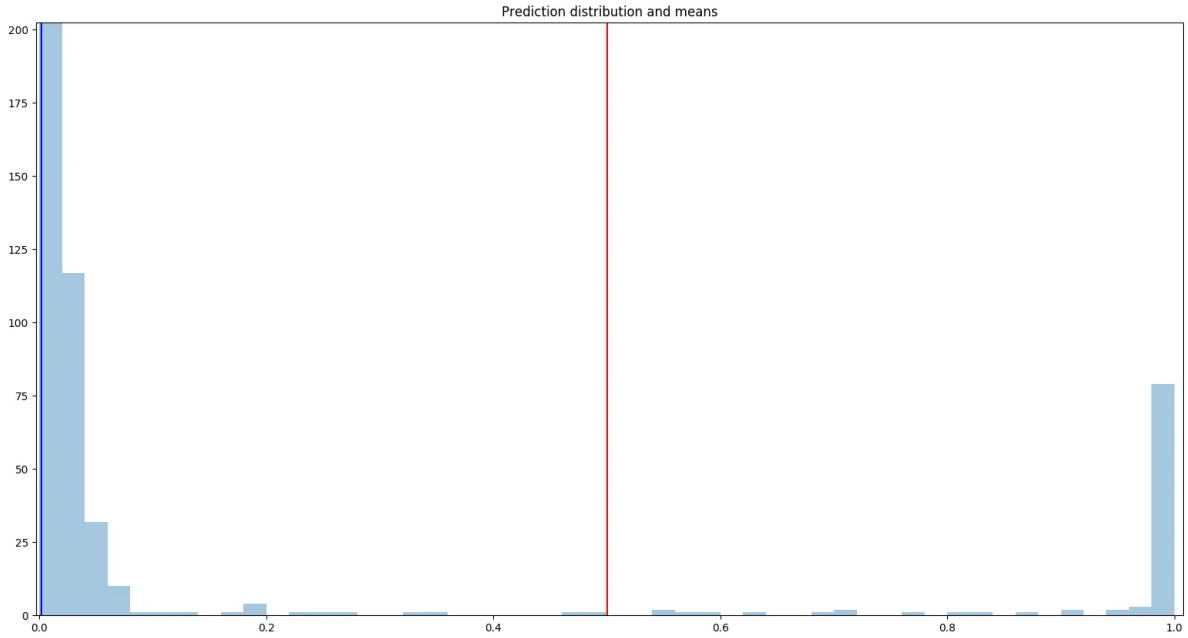
```
## Calibration with Rundom undersampling
evaluation(te.Class, y_predicted_calib_rus)

## Calibration with RUS bagging
evaluation(te.Class, y_predicted_calib_bc)
```

Here are the results. As we expected, there are so many false positives, which we can see from the precision scores.

	RUS	RUS Bagging	Calibration with RUS	Calibration with RUS Bagging
F1 Score	0.0782	0.0999	0.0782	0.0999
Precision	0.0408	0.0528	0.0408	0.0528
Recall	0.9286	0.9286	0.9286	0.9286
Log loss	1.3006	0.9938	1.3006	0.9938

Oh wait, probability calibration didn't change anything at all? There must be something wrong. Let's see the prediction distributions of predicted probability of RUS Bagging with calibration. The blue verticle line is the mean of predicted probability by RUS Bagging with calibration and red verticle line is the mean of predicted probability by RUS Bagging model. Obviously their means are quite far away, for calibrated probability mean is 0.0021 and before calibration is 0.5. Considering the positive class exists 0.17% in a whole dataset, the calibrated probability seems quite close to the actual distribution.



So if we modify the threshold, it should work better and here we can see the results with modified thresholds. Thresholds before calibration and after calibration on RUS model are set at 0.99 and calibration with RUS Bagging is set at 0.8.

	RUS	RUS Bagging	Calibration with RUS	Calibration with RUS Bagging
F1 Score	0.5559	0.7788	0.7692	0.7914
Precision	0.4162	0.7364	0.7732	0.8315
Recall	0.8367	0.8265	0.7653	0.7551
Log loss	0.0794	0.0279	0.0273	0.0236

Summary of results after thresholds are modified

As we can see, after calibration those scores improved, especially the difference between before and after calibration on the random undersampling model are significant.

So by correcting the biased probability by using probability calibration, we could see the performance improvement.

3. Conclusion and thoughts

In this blog post, we went through Pozzolo, et al. (2015). This answers our experience that having more false positive after applying random undersampling. It was quite interesting to see how the resampling method biases the distribution. This would be causing a similar problem in time-series problems as the target variable we are predicting can be quite different from when we trained the model.

Would be interesting to see if this sort of method works for different types of classifiers like tree-based or neural network and also different types of resampling methods.

The codes for this post is available on my GitHub page.

Wrap up

- Introduced the resampling causes a bias in a posterior distribution
- Introduced probability calibration method by using Bayes Minimum Risk theory (Pozzoli, et al. 2015)
- Showed the example of this method
- Confirmed that this method corrects the bias and improve the model results

Reference

[1] Pozzolo, et al., Calibrating Probability with Undersampling for Unbalanced Classification (2015), 2015 IEEE Symposium Series on Computational Intelligence

Machine Learning | Imbalanced Data | Bayes Theorem | Data Science | Calibration

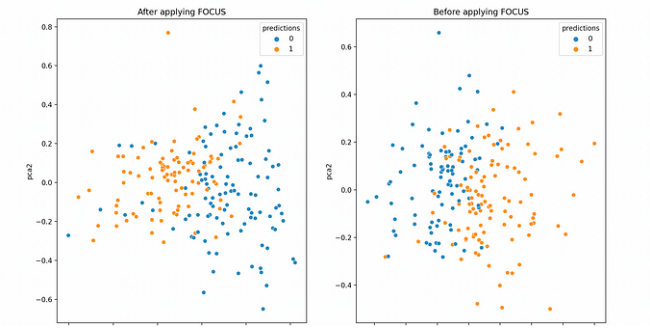
Written by Kyosuke Morita

146 Followers · Writer for Towards Data Science

Following

Senior data scientist at a bank in London.

More from Kyosuke Morita and Towards Data Science



Kyosuke Morita in Towards Data Science

CFXplorer: Counterfactual Explanation Generation Python...

Introduces a Python package for generating counterfactual explanations for tree-based...

✦ · 9 min read · Aug 18

77

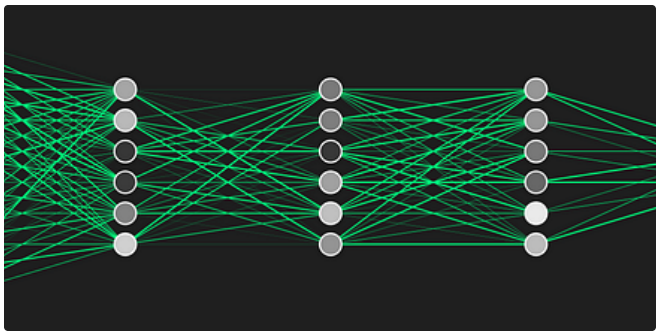
Antonis Makropoulos in Towards Data Science

How to Build a Multi-GPU System for Deep Learning in 2023

This story provides a guide on how to build a multi-GPU system for deep learning and...

10 min read · Sep 17

552 11



Callum Bruce in Towards Data Science

How to Program a Neural Network

A step-by-step guide to implementing a neural network from scratch

· 14 min read · Sep 24

485 4

Kyosuke Morita in Towards Data Science

The deferred acceptance (DA) algorithm utilised in school choic...

How do we make a stable match between students and schools?

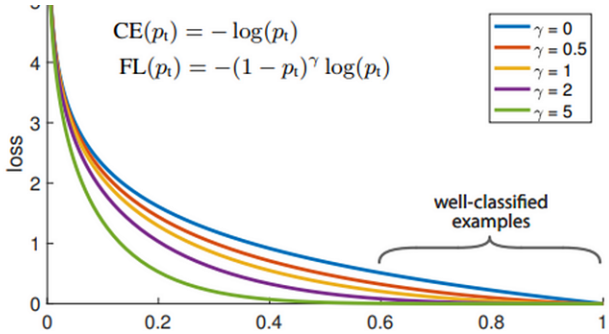
· 7 min read · Aug 5, 2022

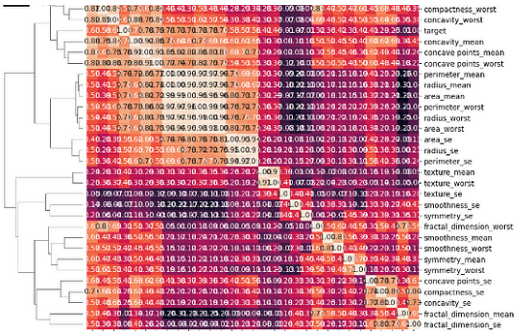
107 1

See all from Kyosuke Morita

See all from Towards Data Science

Recommended from Medium





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

Hazal Gültekin

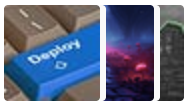
Part1: Breast Cancer Classification

In this article, I will tell you about “Analysis of breast cancer data, especially using the...

7 min read · Sep 28

50

Lists



Predictive Modeling w/ Python

20 stories · 473 saves

Practical Guides to Machine Learning

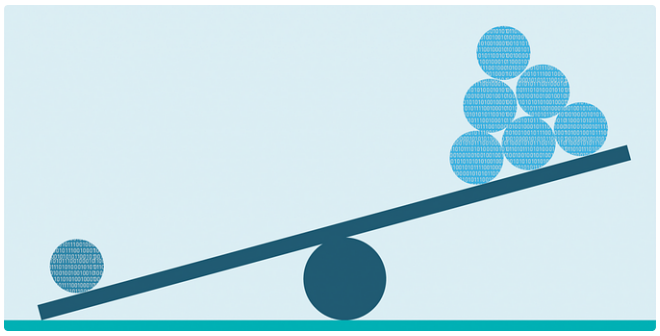
10 stories · 544 saves

Natural Language Processing

689 stories · 304 saves

New_Reading_List

174 stories · 143 saves



The Data Beast

Handling imbalanced data with XGboost:: Class_weight parameter

In XGBoost, the class_weight the parameter is used to adjust the weights of different classe...

2 min read · Apr 12

Shunya Vichar

XGBoost: Giving Your Models an Incremental Boost, One Gradient ...

Let’s unravel the idea of incremental learning in XGBoost through a fictional story. Assume...

4 min read · May 18

12 35

Tarun_KS

Model Evaluation Metrics—Gini Coefficient

About Gini Coefficient, Deriving it with AUC score and its advantages and disadvantages.

4 min read · Jun 25

66

Chandra Prakash Bathula

Machine Learning Concept 68: Platt's Scaling

Platt's Scaling:

3 min read · Apr 13

5

See more recommendations