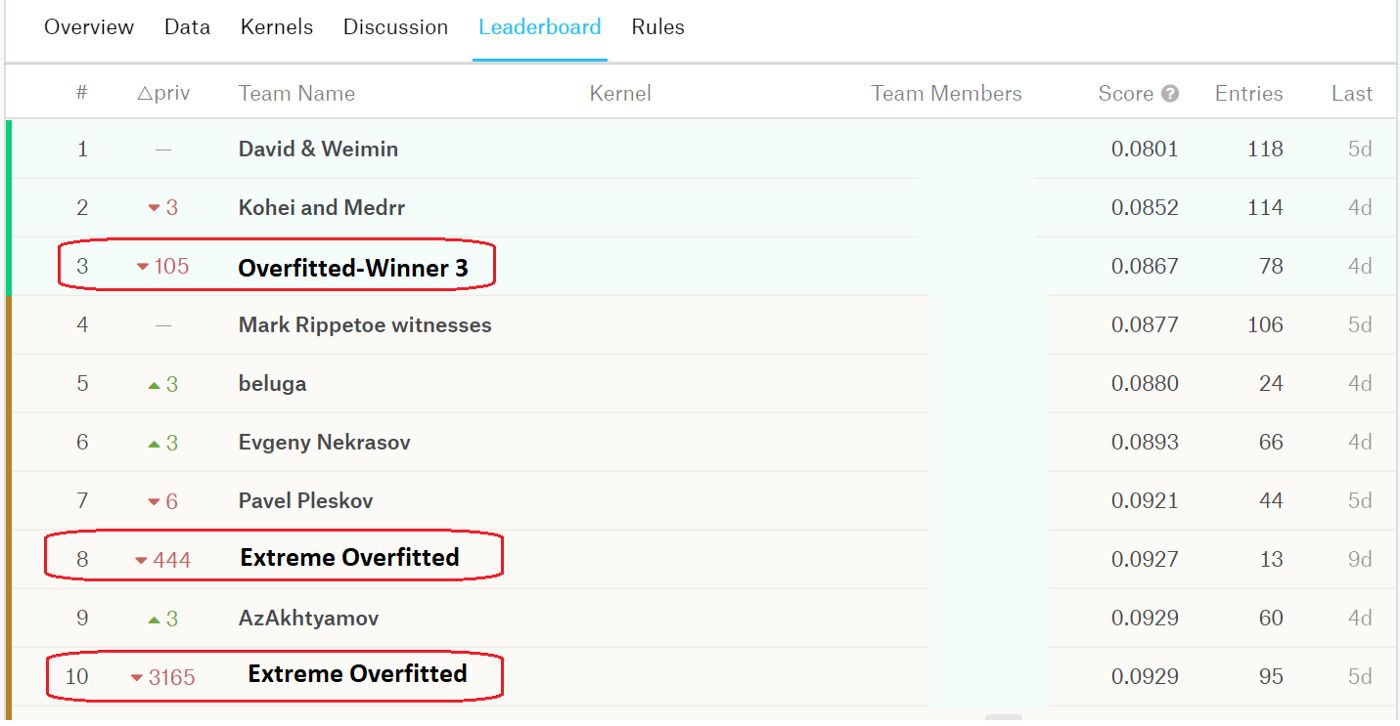
# Another Look into Overfitting

Over-fitting is the nightmare of AI and machine learning practitioners. The best way to see the impact of over-fitting on your life as a data scientist is by comparing public and private leader-boards of kaggle competitions. An example of such extreme cases can be seen in the following screen-shot from the private leader-board of a competition that supposedly winner 3 drops by 105 and even a more extreme case is where rank 10 drops by 3165!!



In this post, we are going to look at a few possible scenarios that over-fitting could destroy our models.

## Basic Setup

Let us assume that, in the context of supervised learning, we are given the training (X\_train, Y\_train) and test datasets (X\_test, Y\_test typically unknown to us). For instance, a dataset of images and corresponding labels in the case of image classification. Our task is to develop a learning model on the training dataset and then deploy it on the test dataset and achieve the best performance possible.

## Over-fitting to Training Data

This is the foremost type of over-fitting that can happen easily. This is how it goes. We convert the learning problem into an optimization problem and define a loss function. We use an optimization algorithm in a loop and minimize the cost function and update the weights iteration by iteration. This is the first loop that we create.

A picture containing screenshot

Description automatically generated

As expected, the loss value gradually decreases. However, if we blindly continue to train the model, the model overfits to the training data!

To monitor and avoid over-fitting to the training dataset, it is common practice to separate a portion (typically 10%-20%) of the data as the validation data (X\_val, Y\_val). By monitoring the performance of the model as it is trained in each iteration on the local validation dataset, we can see that the validation loss will get worse at some point (over-fitting point). This is the point where usually we either stop the training or reduce the learning rate and continue the training until reaching a plateau.

A close up of a logo

Description automatically generated

Congratulations! Our first baby model, which due to the validation dataset is also not an over-fitted model, is ready for deployment. We deploy it on the test dataset and get a decent performance probably close to the validation score. Time to rest up! But NO, what to do with the temptation to improve our model performance? :)

## Over-fitting to Validation Data

What comes next is more interesting. After building the first baby model we typically, in the hope to get better performance, perform hyper-parameter optimization. What that means is that we play around with the model architecture, add and remove layers, regularization, change the learning rate, and sometimes become creative and come up with strange techniques! And unknowingly we close another loop and over-fit to the validation data.

A close up of a map

Description automatically generated

This is the point where we achieve impressive performance on the validation data however when we deploy the model on the test dataset we realize that the model performs poorly. It has been argued that by closing the second loop, we leak information from the validation data to the training process and that is why the model performs much better on the validation data than the test dataset.

To avoid this type of over-fitting, it is advised to perform K-fold cross-validation. To this end, we randomly partition the training dataset into K folds, train a model on (K-1) folds and validate it on the remaining fold. We repeat this process for all folds creating K models. We may observe that some models perform better than others. The average score across all models is considered the final performance of the model. In other words, when we report/decide on the performance of the model based on the local validation data, we do not pick the best validation score but take the average score. The intuition behind cross-validation is that it is technically harder to overfit to the whole dataset than over-fitting to the 10%-20% of the dataset.

Congratulations again! The model you pick using K-fold cross-validation is likely to suffer less from the over-fitting problem or does it?

## Over-fitting to Test Dataset

Over-fitting to test data is another type of over-fitting that mostly happens when your test data is small or only a small portion is public, for instance, the public leaderboards of Kaggle competitions. Even though we perform K-fold cross-validation, we are going to deploy the final model on the test data anyway. One thing that may be ignored is the creation of another outer loop by continuous testing and extra tunning of your model solely based on the result we get from the small test data.

A close up of a map

Description automatically generated

The remedy to this type of over-fitting is to mainly trust your K-fold cross-validation results. This way, you leak less information from the test data to training and will not contribute to the most shameful fault of a data scientist! On top of this, the classical techniques such as data augmentation and regularization should always help with reducing the over-fitting problem at no danger.

## Conclusion

Machine learning models are prone to over-fitting to the training or even test data. We need to watch out the extra loops that we create during the development process and avoid leaking information from the test data to the model.