Metrics which can be used to assess the model risk:

- Sharpe Ratio
- Max Drawdown

Sharpe Ratio

Sharpe Ratio describes the returns distribution. It is a ratio of mean returns divided by its standard deviation. The preferred distribution is with high mean and narrow distribution (and small tails). Such distribution will result in a high Sharpe Ratio (the higher Sharpe the better).

For me, when I'm looking for new models, I'm considering as good, all models with Sharpe Ratio > 1.0 on validation data and as a very good model Sharpe > 1.2.

Max Drawdown

Describes the maximum loss in the returns in a single round. The lower the max drawdown the better.

For me, when I'm looking for new models, I consider a model with max drawdown < 0.04 as a good model.

All code is available at https://github.com/pplonski/numerai_metric_research/blob/master/Risk_Metric_Research_v2.ipynb

How to improve risk metrics?

There are many ways to improve risk metrics. It is important to keep in mind that all metrics are connected and by fixing one metric we can worse on other metrics. Although the best models will perform well on all metrics. In this post I will show you 3 example ways to improve risk metrics:

- Feature usage selection (colsample_bytree

high vs low)

- Eras selection (hard vs easy eras)
- Feature Normalization

Feature usage selection (colsample_bytree

tuning)

In my examples, I'm using the Xgboost algorithm. It has a colsample_bytree

hyper-parameter. This parameter controls how many features are used for building a single tree. For example colsample_bytree=0.5

with 310 features will result in considering only 155 features, randomly picked for every tree. This helps to improve model generalization. In the case of risk metrics - the lower colsample_bytree

the higher Sharpe and lower max drawdown should be expected. In the case, when the algorithm that you are using hasn't got a similar hyperparameter to colsample_bytree

, you can try to run feature selection (for example by randomly selecting features or removing the least used features).

In my experiment I trained 25 Xgboost models with two different colsample_bytree

ranges:

- colsample_bytree

from [0.7. 1.0) range (high colsample_bytree

),

- colsample_bytree

from [0.0. 0.3) range (low colsample_bytree

).

Other hyperparameters were set to:

- learning_rate = 0.1

,

- n_estimators = 50

,

- max_depth = 5

.

Models were trained on 2-fold CV (with a split on eras, era split at 60), these models are used to compute training_*

metrics. To compute validation metrics val_*

the model was trained on all available data.

The results are presented in the charts below:

[

low_colsample_vs_high_colsample

767×882 41.2 KB

](https://forum.numer.ai/uploads/default/original/1X/eb9d537fe8502a2e8d7f35a7986d163f86393ce3.png)

Worth to notice:

- There is no increase in val_sharpe

as expected. Maybe models with low colsample_bytree

need more tuning?

- There is an improvement in decrease in val_max_drawdown

, High colsample_bytree

to low, min of val_max_drawdown

0.05637 -> 0.04887

Eras selection (hard vs easy eras)

We can use the “Example Predictions” model to divide the training data into two sets:

- Easy eras (eras with a high correlation of predictions),
- Hard eras (eras with a low correlation of predictions).

To do this, compute correlation for each era in training data and sort them based on correlation. Use the top 60 eras (from total 120 eras) as easy eras, and the 60 worst performing eras as hard eras.

I trained 25 Xgboost models on hard eras and 25 Xgboost models on easy eras. The hyper-parameters used:

- learning_rate = 0.1

,

- n_estimators = 50

,

- max_depth = 5

,

- colsample_bytree

random from [0.3, 0.7].

The results are in the picture below.

[

easy_vs_hard_eras

748×864 37.8 KB

](https://forum.numer.ai/uploads/default/original/1X/26dfac62443ef55c6e38273b2e978cf9e5b6d3fb.png)

Worth to notice:

- The maximum of val_sharpe

increases, easy eras -> hard eras, 0.6822 -> 0.7935,

- The minimum of max_val_drawdown

decreases, easy -> hard, 0.1193 -> 0.0726

- Keep in mind the risk is reduced a lot, however, there is a decrease in val_mean

(in returns), easy -> hard, 0.0264 -> 0.0150. You need to find a balance that will give high returns and make an acceptable risk.

Feature neutralization

Feature neutralization is a process of removing linear dependency between features and predictions. This step can help to reduce the risk.

I've run 100% feature neutralization on all predictions from earlier experiments. Below results.

Neutralization on high colsample_bytree

[

neutralized_high_colsample

779×868 46 KB

](https://forum.numer.ai/uploads/default/original/1X/abb6e2fdc30e4942f18e2cb176de77bc961b890b.png)

Worth to notice:

- Neutralization decreases the val_mean

, max 'val_mean` 0.2797 -> 0.02268

- Neutralization increase the val_sharpe

, max val_sharpe

0.9314 -> 1.0674

- Neutralization decrease val_max_drawdown

, min val_max_drawdown

, 0.0563 -> 0.0217

Neutralization on easy eras

[

neutralized_easy_eras

744×865 42.8 KB

](https://forum.numer.ai/uploads/default/original/1X/9641aa5eccdfa8d0b1f391c64710fa26466e5fae.png)

Worth to notice:

- Neutralization decreases the val_mean

, max ‘val_mean` 0.2797 -> 0.02268

- Neutralization increase the val_sharpe

, max val_sharpe

0.9314 -> 1.0674

- Neutralization decrease val_max_drawdown

, min val_max_drawdown

, 0.0563 -> 0.0217

Mix ideas

- In this post, there were presented a few ideas on how to reduce the risk.
- Keep in mind, that all metrics are connected, and changing one metric will change others as well. You need to find a model that will balance well between risk and profit metrics.
- It is important to mix different ideas and look for the original solutions (to have high MMC). For example, apply feature and eras selection and at the end the neutralization step. An example solution can be:
- Remove 10% of features (the least important)
- Remove 10 the easiest eras
- Appy neutralization at level 50%
- Run hyperparameters search for optimal learning_rate, max_depth, colsamples_bytree.
- Remove 10% of features (the least important)
- Remove 10 the easiest eras
- Appy neutralization at level 50%
- Run hyperparameters search for optimal learning_rate, max_depth, colsamples_bytree.
- When I was doing experiments I observed that better results are obtained when early stopping is used instead of a fixed number of estimators in Xgboost. I’m planning to make two more posts:
- Apply early stopping in Xgboost with custom evaluation metric.
- Apply AutoML solution on numer.ai data (I’m working on AutoML python packagehttps://github.com/mljar/mljar-supervised)
- Apply early stopping in Xgboost with custom evaluation metric.
- Apply AutoML solution on numer.ai data (I’m working on AutoML python packagehttps://github.com/mljar/mljar-supervised)