Model Summary

Optiver Realized Volatility Prediction

1 Team information

Team & competitor name:	Michael Poluektov
Final score:	0.20013
Final position:	7th
Location:	Bristol, United Kingdom
Email:	michael.poluektov@gmail.com

2 Personal background

I am an undergraduate Computer Science student at the University of Bristol, going into 2nd year at the time of the competition. I had no prior experience in finance or data science, which is why I decided to join this competition. The time spent on the competition was anywhere between 3 to 12 hours per day during the 2 month period I was competing.

3 Summary

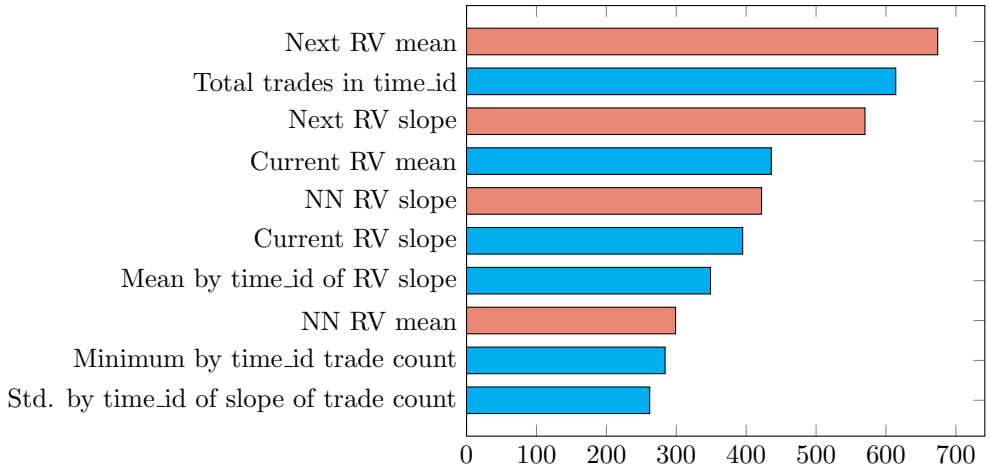
My best submission used an ensemble model of LightGBM and a basic feed-forward neural network with 3 hidden layers. The model itself was for the most part copied from @alexioslyon (2021)'s public notebook which was probably copied from/heavily inspired by other public kernels.
The main contribution which set me apart from other competitors was feature engineering, such as the use of time_id aggregations and re-ordering, which I suspect was used to some extent in most top 10 submissions.
Many other less significant changes were also made, such as using linear regression instead of directly passing features calculated over multiple time windows.
BarutaSHAP was used for feature selection, as well as a less formal intuitive approach of "remove all features that could break in 9 months".
On a standard Kaggle CPU kernel, the feature pre-processing takes about 2 hours, though it could be optimized by replacing Pandas with a GPU library such as RAPIDS, and the training takes about 30 minutes per model. Feature selection was incomplete, and could be improved with further processing time.

4 Feature Selection and Engineering

This section covers the feature engineering for my highest scoring model, using time_id relationships. See section 7 for a more practical model.

4.1 Feature importance plot

Below you can find a feature importance plot extracted from my LightGBM model, with 10 out of 198 most important features, ranked by the number of splits. Features that may include future information are marked in red.



4.2 Time ID relationships

As mentioned in the section above, many features are derived from various time_id relationships, which could be used thanks to the de-normalization of stock prices using tick sizes. These include:

- The mean, std, min and max of the following features from the 6, 20, 50 and 100 "closest" time_ids (as determined by the KNN algorithm on denormalized prices) :
  - WAP log return realized volatility (RV) mean, slope and error
  - Total volume
  - Trade log return realized volatility
  - Mean, std, min and max of each of these features by time_id
- The following features of the "next" time_id (as determined by brute forcing the shortest Hamilton path in the 6 closest time_ids, using Euclidean distances between the de-normalized prices as weights):
  - RV mean, slope and error
  - Total volume
  - Trade count mean, slope and error
  - Mean, std, min and max of these features by time_id.
- Various features using Euclidean distances between those time_ids.

To my surprise, the biggest boost in performance was provided mostly from the less "leaky" features (out of those based on price de-normalization), such as those based on Nearest Neighbors and not the ones based on the "next" time ID. This may be due to an error I made by using Euclidean distances between re-normalized stock prices rather than some kind of function that does not depend on price (such as $\sqrt{\sum(\log(\frac{u_i}{v_i})^2)}$).
In addition to future information, these features allowed the model infer the following:

- An approximation of the current time of day, based on the index of the current time_id in the re-ordering of the closest time_ids.
- Data about the stock market activity in the current hour/day/month.
- The price differences of various stocks between market close/market open.

The idea of restoring individual stock prices to leverage time_id relationships came from @nquay3 (2021)'s post.

4.3 Traditional features

The most reliable traditional features also ended up being the simplest ones:

- WAP1 & WAP2 Realized volatility
- Trade counts/sizes

- Volume imbalances
- Price spreads
- Other common features

Their means, minimums, maximums, their aggregations over time etc. have been shown to be more important than many quantitative features such as:

- Quad or tri-power quarticity
- Beta coefficient
- Other metrics outlined in various research papers on the subject (such as Corsi et al. (2008))

Out of many such features I had attempted to use, only realized quadpower quarticity (second degree volatility of the WAP) has been deemed as useful by BarutaSHAP.

The main practical trick which may have set this submission apart from other competitors is feature selection and manual dimensionality reduction. Manual feature selection included decisions such as:

- Not using stock_ids or stock_id aggregations as part of the feature set. This lead the model to be more robust to mergers, stock splits or other considerable changes in a stock's behaviour which may have happened over the 9 month period between the capture of the train and test datasets. Only the nearest neighbour stock aggregations were used, and I believe that in a real life setting it would also be relevant to build a model that's only aware of the past couple of days of a stock's performance (in addition of some kind of external option to account for such events). It is, important to note, however, that stock_id as a categorical feature did improve RMSPE of a fully leak-free model by roughly 0.00300.
- No specific stock interactions were used either, for the same reason. While I believe that this would lead to a considerable performance increase in a real-life scenario, the competition's format didn't seem to encourage it.
- Trade sizes and order counts were used, however after some consideration I suspect that over-reliance on these features may have negatively impacted the model in the long run, as those features have the potential to diverge from their respective means over time.

The remaining features have then been filtered with an incomplete run of BarutaSHAP, as mentioned in the summary. This was essentially achieved through collecting data from multiple re-runs of XGBoost, and pruning features that were unimportant or highly correlated with other features. This method has been shown to be more successful than simply keeping the top X best features from the importance plot, as it also considered feature interactions and could be used to define that threshold of the amount of features to keep.
The main drawback was the significant computational cost, which has been amplified by my lack of access to any hardware more performant than Kaggle's free CPU notebooks (although I don't think this would be a problem for the host). Overall, a total of about 50 features deemed to be of no significant importance have been pruned, with an extra 60 which have not yet been confirmed as important.
To reduce dimensionality, the features calculated over multiple time windows in a time_id have not been passed directly to the models. Instead, the mean, slope and error of those features were calculated using a simple least squares model (directly solving a linear regression problem). This has cut the number of features used almost in half, and provided better way to interpret feature importance.

5 Models, training and CV

As mentioned in the overview section, the final submission is based on a simple LightGBM and FFNN ensemble. I do not take credit for the models themselves as they were taken from a public notebook @alexioslyon (2021), and none of the alternative models I've tried managed to produce a better performance. While I believe the above mentioned models were the right choice, they could likely be improved with further hyper-parameter tuning to account for the change in features, perhaps making use of Bayesian regression or a similar algorithm.
The FFNN model in the submitted version contained 3 hidden layers with 128, 64 and 32 neurons. The "swish" activation function was used. The LightGBM model also used quite basic hyper-parameters which can be found in the code.
Training took about an hour on a CPU notebook, which I believe is insignificant compared to other proposed models. Training time could also improve as more features could be thrown away. feature pre-processing takes

about 2 hours, an hour of which is taken up by brute forcing the re-ordering of time_ids. That could also be improved by using a GPU library.
The cross-validation techniques were also taken from a public notebook. The LightGBM model uses a standard KFold, which is clearly sub-optimal, as a time_id based GroupKFold would clearly be more accurate. The FFNN uses a "KFold based on the KNN+ algorithm" found in @alexioslyon (2021)'s notebook and described by @AmbrosM (2021).

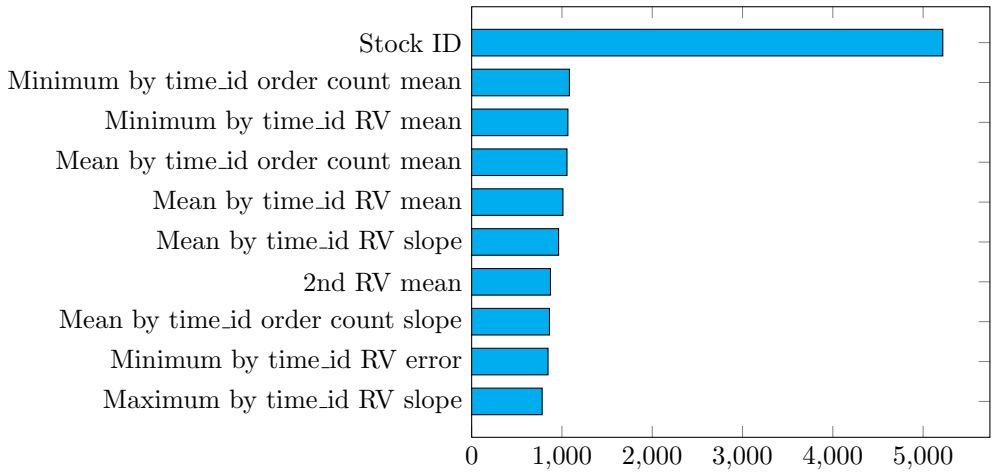
6 Interesting findings

All relevant interesting findings have been mentioned in the above sections. To recapitulate:

1. Manual feature engineering combined with simple models seems to be more reliable than models working directly on time-series data. See [5].
2. Using linear regression and inputting mean, slope and error of variables is an improvement over simply inputting those features over multiple time windows. See [4.3].
3. Tick sizes can be exploited to obtain the order in which the data was captured, which greatly improves competition score. See [4.2].
4. Dedicated features selection algorithms seem to yield a performance improvement compared to inbuilt feature importance rankings. See [4.3].

7 Simple & leak-free model

Upon experimenting with late submissions, I have created a model with an RMSPE score of 0.22955 on the private dataset with the following specifications: A single LightGBM model, using 45 features, none of which were generated from the test dataset – i.e. the model would output the same predictions for each time_id, independent on the number of time_ids provided as input. The features used were a subset of the features of the main model with a categorical stock_id feature added. Here is an importance plot of the 10 best features:



8 Model execution time

8.1 Submitted model

As mentioned in the overview section, the submitted, highest scoring version's execution time is roughly:

- 2 hours for feature pre-processing
- 30 minutes of training per model (an hour in total)

Since the above mentioned model is unpractical, inference time was not measured.

8.2 Practical model

The model described in section [7] has an execution time of:

- 11 minutes for feature pre-processing
- 5 minutes for training
- 0.2 seconds for inference per time_id (112 windows, feature processing included)

References

@alexioslyon. 2021. https://www.kaggle.com/alexioslyon/lgbm-baseline

@AmbrosM. 2021. https://www.kaggle.com/c/optiver-realized-volatility-prediction/discussion/268813#1494997

Corsi, F., Mittnik, S., Pigorsch, C., & Pigorsch, U. 2008, Econometric Reviews, 27, 46, doi: 10.1080/07474930701853616

@nquay3. 2021. https://www.kaggle.com/c/optiver-realized-volatility-prediction/discussion/256725