MTH 9899 Final Project Report

Group Name: Lexington Alpha
Group Member: Yigao (Hugo) Liu, Haocheng (Frank) Gu,
Chenyu (Phillip) Zhao, Zichao (David) Wang

May 22, 2019

1 Trim Data

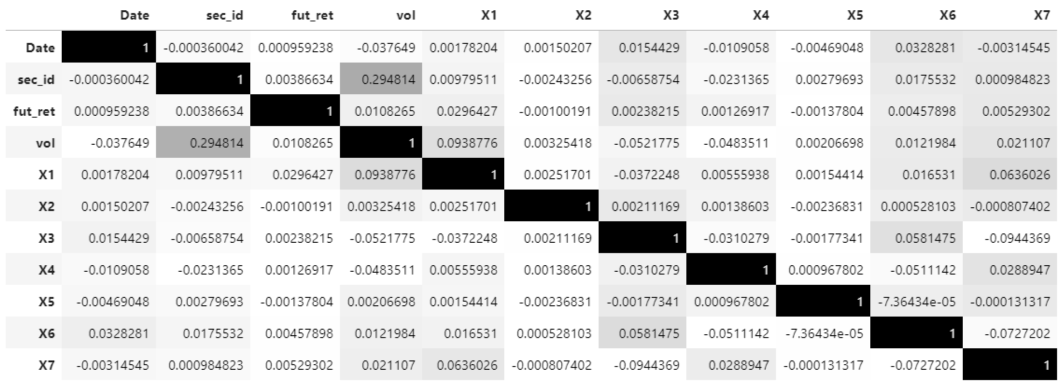
1.1 Observations before Trimming

Before applying any trimming technique, we first have several observations:

1. Label ”Time” is invalid. Therefore, we drop ”Time” column in our data.
2. ”fut_ret” is more likely to be residual return rather than raw return, since its sample average ($= 1.35 \times 10^{-4}$) is very close to 0.
3. ”X2” and ”X5” have bell-shape sample distributions, with mean very close to 0 and standard deviation almost exactly 3×10^{-4}, which means that they are probably just artificial white noise.

1.2 Correlation Heatmap and Population Histogram

The correlation heatmap is as follows.



	Date	sec_id	fut_ret	vol	X1	X2	X3	X4	X5	X6	X7
Date	1	-0.000360042	0.000959238	-0.037649	0.00178204	0.00150207	0.0154429	-0.0109058	-0.00469048	0.0328281	-0.00314545
sec_id	-0.000360042	1	0.00386634	0.294814	0.00979511	-0.00243256	-0.00658754	-0.0231365	0.00279693	0.0175532	0.000984823
fut_ret	0.000959238	0.00386634	1	0.0108265	0.0296427	-0.00100191	0.00238215	0.00126917	-0.00137804	0.00457898	0.00529302
vol	-0.037649	0.294814	0.0108265	1	0.0938776	0.00325418	-0.0521775	-0.0483511	0.00206698	0.0121984	0.021107
X1	0.00178204	0.00979511	0.0296427	0.0938776	1	0.00251701	-0.0372248	0.00555938	0.00154414	0.016531	0.0636026
X2	0.00150207	-0.00243256	-0.00100191	0.00325418	0.00251701	1	0.00211169	0.00138603	-0.00236831	0.000528103	-0.000807402
X3	0.0154429	-0.00658754	0.00238215	-0.0521775	-0.0372248	0.00211169	1	-0.0310279	-0.00177341	0.0581475	-0.0944369
X4	-0.0109058	-0.0231365	0.00126917	-0.0483511	0.00555938	0.00138603	-0.0310279	1	0.000967802	-0.0511142	0.0288947
X5	-0.00469048	0.00279693	-0.00137804	0.00206698	0.00154414	-0.00236831	-0.00177341	0.000967802	1	-7.36434e-05	-0.000131317
X6	0.0328281	0.0175532	0.00457898	0.0121984	0.016531	0.000528103	0.0581475	-0.0511142	-7.36434e-05	1	-0.0727202
X7	-0.00314545	0.000984823	0.00529302	0.021107	0.0636026	-0.000807402	-0.0944369	0.0288947	-0.000131317	-0.0727202	1

Figure 1: pairwise correlation heatmap

Here we find ”sec_id” has a strong positive correlation with volatility. Therefore, we can probably deduce that stocks in our universe are ordered by capitalization size in a decreasing way, since low cap size companies are not liquid enough, and therefore tend to have larger volatilities.

This also tells us that label ”vol” has relative less noise. We will use this finding later.

Now let’s have a look at the population histogram on the next page.

As discussed above, ”X2” and ”X5” are very likely to be pure white noise (or at least the cases where white noise just overwhelms our signal), because they are bell-shaped, and not skewed. ”X1” has a large portion of data close to 0. Therefore, I think though there is some composition of white noise in ”X1”, it is not pure noise. Let’s justify this assumption in the following section.

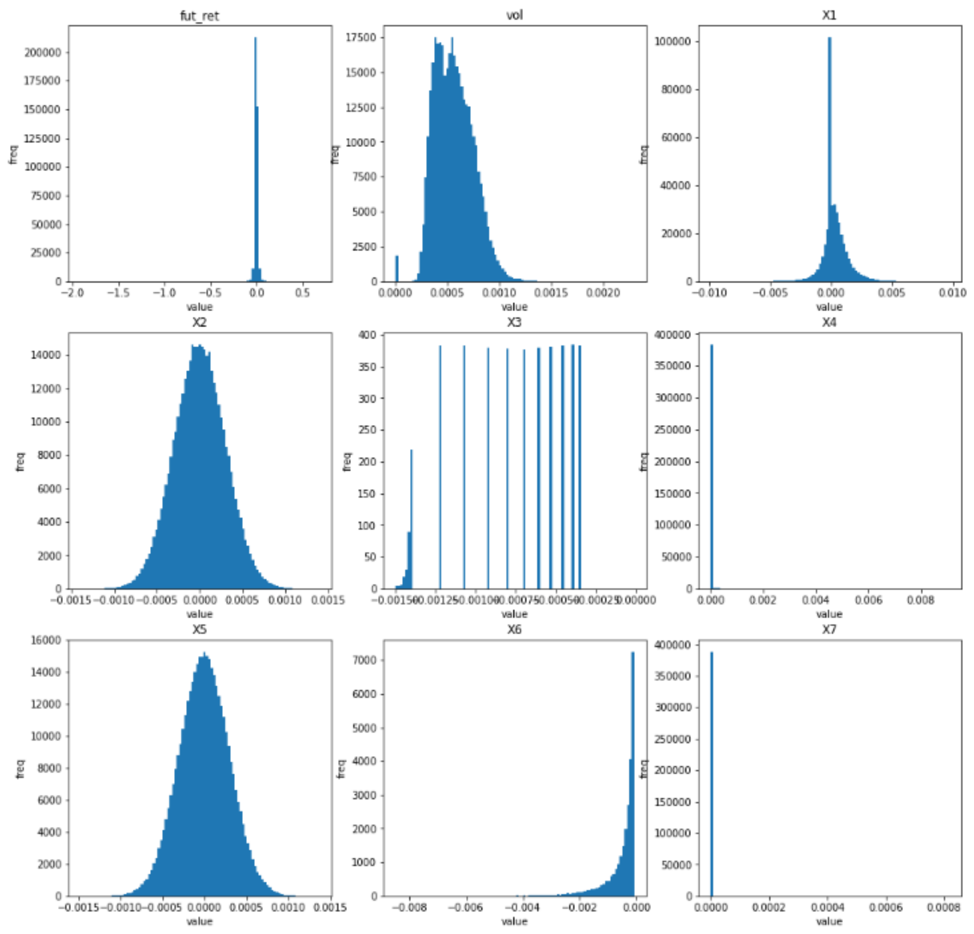


Figure 2: population histogram for each feature

1.3 Features Dominated by Random Noise

One way to further investigate whether a feature "X*" is noise is that for each ticker, we plot the 1st, 2nd and 3rd moment of "X*" across the time, and see if the value varies across each ticker.

- If it does not vary, "X*" is probably noise.
- Otherwise "X*" may contain some information.

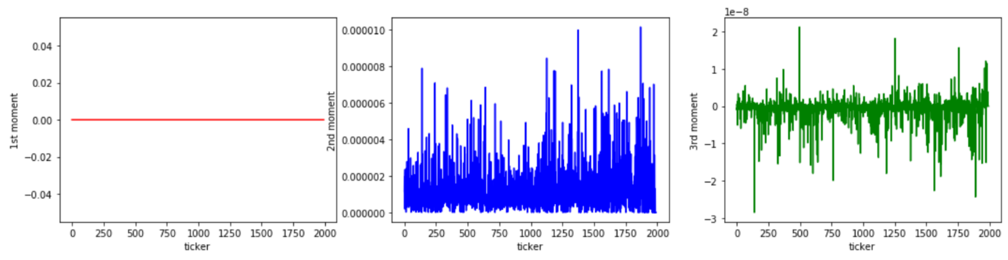


Figure 3: 1st, 2nd and 3rd moments of "X1" across different tickers

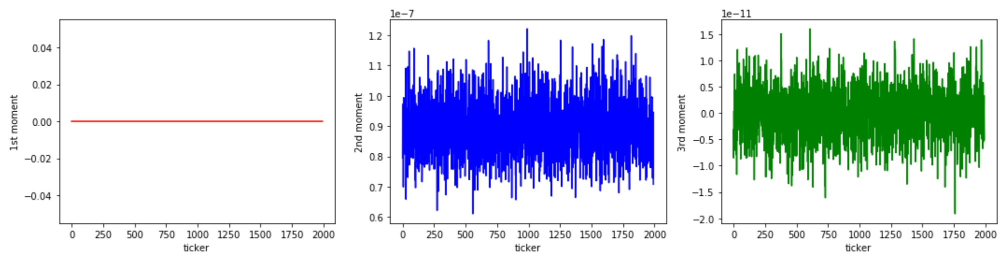


Figure 4: 1st, 2nd and 3rd moments of ”X2” across different tickers

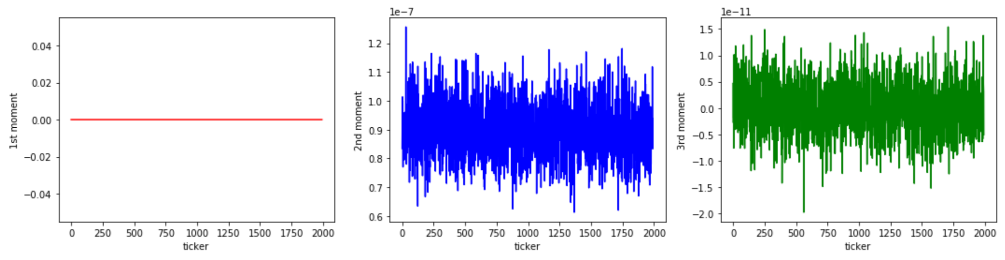


Figure 5: 1st, 2nd and 3rd moments of ”X5” across different tickers

We can find moments of ”X1”, ”X2” and ”X5” as above. From the 2nd and 3rd moment, there is some information in ”X1”, while there is no pattern in moments of ”X2” and ”X5”. We will only drop ”X2” and ”X5”.

1.4 Deal with Missing Values

In raw data, all features except for ”vol” do not have np.nan, while ”vol” has 30,098 np.nans. But after reshaping the data such that each feature as well as return is a dataframe, indexed by date with each ticker per column, we actually find that all features except for ”vol” have 5,861 np.nans, while ”vol” have $5861 + 30098 = 35959$ np.nans. This is probably because that in original raw data, different stocks have slightly different time stamps.

After further tests, we find that 1,541 (77.4% of total) tickers and 12 (6% of total) dates have np.nan. There is no way we can just drop a whole column or row in our dataframe.

Before starting to solve this problem, we first point out that there are two most common possibilities for missing values in a time series:

1. A few np.nans lie among valid data in a time series.
2. A big chunk of np.nans at the beginning (or in the middle) of our time period.

We can probably use some kind of moving average to fill the np.nans in case 1; but we have to treat case 2 more seriously. Let’s deal with np.nans in features except for ’vol’ first.

1.4.1 np.nans in Features other than ”vol”.

Due to our discussion above, in different feature dataframes (such as ”fut_ret” and ”X7”), all np.nans are in the same place. So if we fill the missing values using some moving average (such as exponential moving average), we will get $\frac{5861}{392539} \approx 1.5\%$ of our data points among which all the features are just filled in artificially.

Thus we have to drop them in the end and leave the time stamps of different stocks somewhat not identical.

1.4.2 np.nans in "vol"

"vol" is a label with relative small noise and lots of np.nans. We have to fill (at least some of) them in some way.

One way to fill np.nan is to use exponential moving average. We implement the moving average using historical data in order to avoid lookahead bias. But there is one parameter we have to decide: the center of mass (or effective lag) of our moving average. There are two concerns:

- we do not want to use a very long window length, because that will incorporate too much old information;
- we do not want to use a rather short window length either, because it will inherit the noise of its previous observations.

Therefore, we have to do some kind of parameter tuning.

Notice that "sec_id" has a strong correlation with "vol". Thus, one way to decide is that we try different window lengths of filling volatility, and pick the one with the strongest correlation with "sec_id". (Note that we actually think that "sec_id" is ranked by capitalization size.)

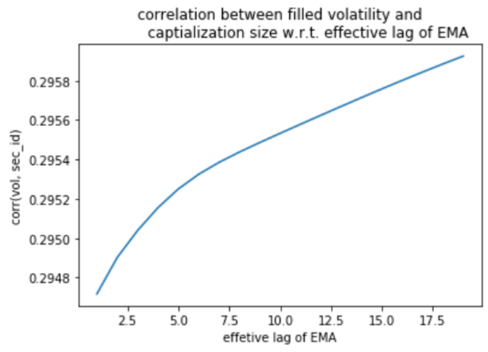


Figure 6: correlation with EMA-filled volatility w.r.t. sec_id

We are disappointed to see that the correlation keeps going up, and a EMA with $lag > 15$ actually does not make sense. Thus, we turn to the second strongest correlation: "vol" and "X1".

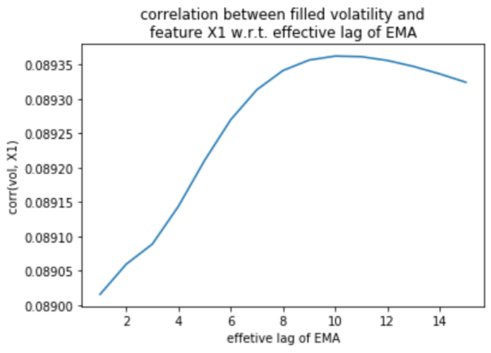


Figure 7: correlation with EMA-filled volatility w.r.t. X1

Based on the plot, we will choose $lag = 10$.

We also need to test if there are multiple consecutive np.nans in our time series. In order to investigate on this problem, we use exponential moving average to fill "vol" multiple times. If the number of np.nan keeps decreasing, we actually have consecutive np.nans in the middle of our dataframe. The chart is as below.

Thus, after filling once, the number of np.nan does not further decrease any more. This indicates that there are no consecutive (i.e., more than one) np.nans in the middle of our "vol" dataframe.

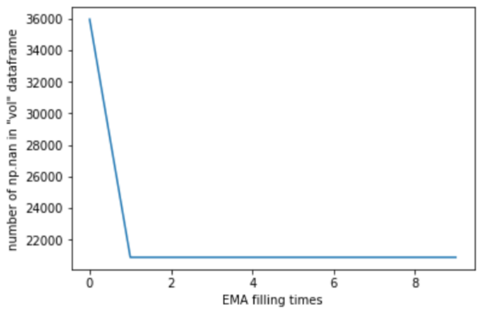


Figure 8: number of np.nans in ”vol” w.r.t. filling times using exponential moving average

After filling with exponential moving averages, we can find that there are still 20,887 np.nans in ”vol”. The plot showing the number of np.nan with respect to date can be found as below.

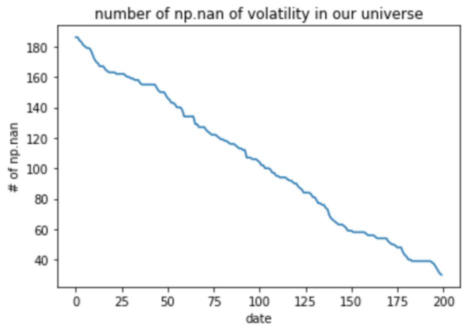


Figure 9: number of np.nan of volatility w.r.t. date

Here we kind of know what is going on. As time goes by, the number of np.nan in volatility dataframe decreases monotonically. It is very likely to be the case that some stocks do not have any observation of volatility at the beginning; but later more and more stocks tend of have valid volatility observations. Thus we will adopt the simplest way: just leave all remaining np.nans unfilled. The intuition behind it is that we have no reason to treat a np.nan the same as 0, because the latter is a valid value, whereas the former is not.

1.5 Normalize Data

The final step is to normalize our data. We set our features into four sets and treat them differently:

1. sec_id, Date: we will divide each value by the sample maximum.
2. X2, X5: we will not normalize them because we will drop them in the end anyway.
3. vol: we will first add a minor positive number (10^{-4}) in order to avoid numerical error, and then take log and compute z-score. New feature will be named ”log_vol”.
4. X1, X3, X4, X6, X7: we will compute z-score of them directly. New features will be named ’X*_norm’.

In conclusion, in the final data, we have the following columns: Date, sec_id, fut_ret, vol, log_vol, X1, X3, X4, X6, X7, X1_norm, X3_norm, X4_norm, X6_norm, X7_norm.

2 Regression

2.1 Different Regressions

To start evaluating the predictability of the current data, we start of with OLS to get a general views. We first utilize all data and features for the regression. The out of sample R-square ends up at 11.6 bps. Basically, the result is saying that prediction is barely better than mean.

Therefore, we look at the predictions and compare them with the actual data. We see that the actual data is much more scattered than our predictions (See figure: Prediction vs. Actual).

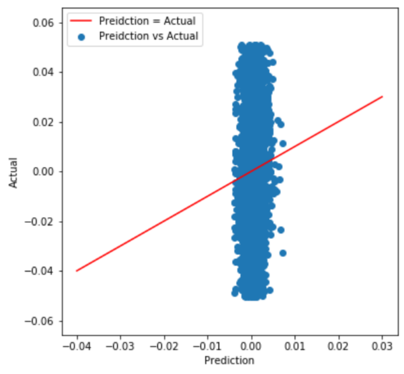


Figure 10: Prediction vs. Actual

Then We suspect that there may be too many outliers that affect our predictions. We use Median Absolute Deviation (MAD) to clip data with future return greater than (less than) + (-) 4. However, R^2 decreases.

Given our finance intuition, we sometimes notice that when there are multiple signals in the same period, market may react to the strongest signal. So we try to pick the most positive and negative signal among X1, X3, X4, X6, X7 and set the less strong signal to be 0. As the result of this data manipulation, out of sample R^2 has no improvement.

We also try to have 2 models based on the SEC ID, but the result has a small improvements, almost 12bps.

LASSO and Ridge provide similar in sample and out sample R^2 since we have a large data set and we have already picked out two variables that we believe are random noise. Therefore LASSO and Ridge cannot become very helpful in this case.

Interesting, using LASSO enable us to observe the change of variable coefficients. It turns out Date and SEC ID drop to 0 first as penalty increases while X1 stays till the end (See figure: Coefficients of Variables as Penalty Increases).

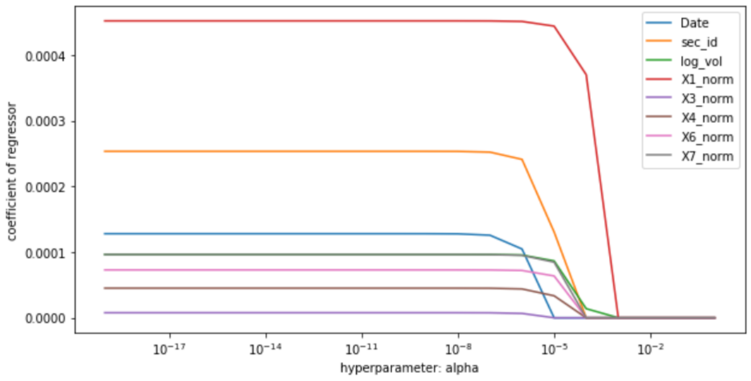
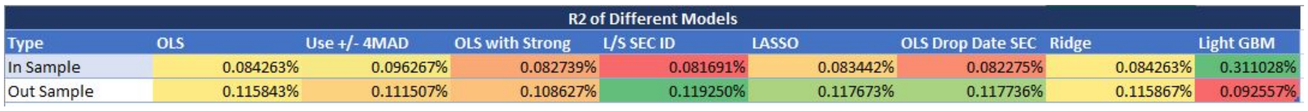


Figure 11: Coefficients of Variables as Penalty Increases

After experimenting with different regression models, we wonder if something more sophisticated like Light GBM will give us any surprises. However, it turns out Light GBM is only better at fitting in sample data.

2.2 Models Result

(See figure:R^2 of Different Models).



R2 of Different Models								
Type	OLS	Use +/- 4MAD	OLS with Strong	L/S SEC ID	LASSO	OLS Drop Date SEC	Ridge	Light GBM
In Sample	0.084263%	0.096267%	0.082739%	0.081691%	0.083442%	0.082275%	0.084263%	0.311028%
Out Sample	0.115843%	0.111507%	0.108627%	0.119250%	0.117673%	0.117736%	0.115867%	0.092557%

Figure 12: R^2 of Different Models

3 Tree and Random Forest

3.1 Data

Since the tree algorithm won't be influenced by order-preserving transformation, so the only important parts is how we deal with default values and outlier. The volatility I used is log-vol, because other teammates may need this, and log has no influence on my model. About feature X1 to X7, we find out that there are a lot of zeros. So we think zeros may mean something different. But I first tried default setting, which is all the features are split by a random chosen number.

3.2 Default Tree and Forest

About the parameters in the tree, I use cross validation to get the best. And it turns out that the only important parameter is the depth of the tree. Following is the cross validation plot about the depth.

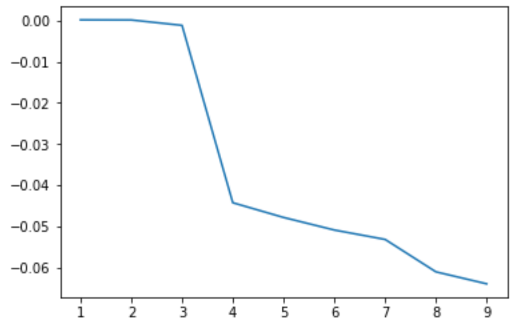


Figure 13: cv for default tree

We can see that 3 is the best depth for one single tree, which we think is not that reasonable, since we have 7 features in total. But still, we tried to use random forest on this tree to see the result.

The next thing we did is to use the cross validation to get the best parameters for forest. Again, the only important parameter here is the total numbers of trees. Figure 14 is a picture of the cross validation and r^2 result.
Actually on average, 100 to 300 hundreds trees give us a similar result.The out-of-sample r^2 for this model is about 6 bps. Following is the hist of the out-of-sample output.
From Figure 15, we can see that this predictor is just predicting the mean values. And also, there is no obvious tail effect in this distribution, which is disobey our opinion on future return.

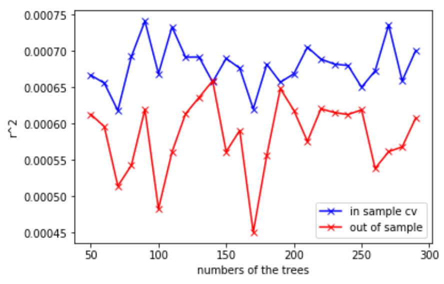


Figure 14: cv for defalut tree

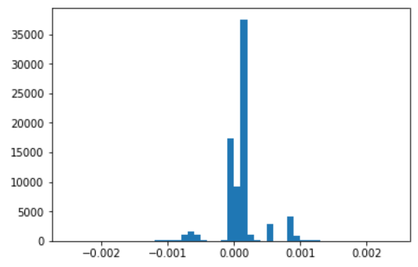


Figure 15: hist of default tree

3.3 Self-designed Tree and Forest

We think this model has a problem which is the it randomly split feature X1 to X7. By observation, we find that there are a lot of zeros in those features, so we think 0 should mean something different from non-zero values. So we change the tree like this: if a feature has a lot of 0, we first see whether 0 is a good split point. If it is, we use 0 to split. After this, we randomly choose values from non-zero values to split the points.

Again we do the cross validation for both single tree and forest. Following is the cross validation result for one single tree.

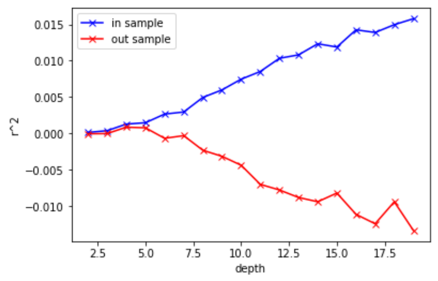


Figure 16: cv of self-designed tree

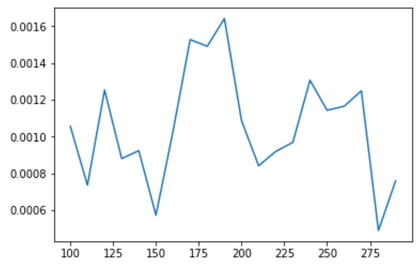


Figure 17: cv of self-designed forest

From Figure 16, we can see that the best depth now is 6 or 7, so we finally choose 7 as the best depth and build the forest. Following is the cross validation on the forest.

From Figure 17, we can see that 170 to 190 gives me a similar result, so we finally choose 190 as the best parameter. Following is the hist for the out-of-sample output.

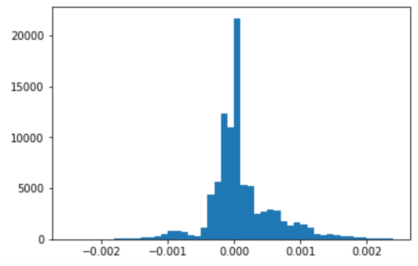


Figure 18: hist of self-designed forest

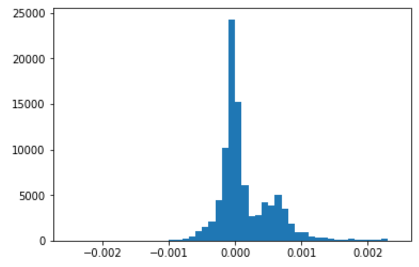


Figure 19: hist of new forest

From Figure 18, now we can see that the there is a fat tail on the distribution, which is in accordance with our common sense. Also, the average out of sample r^2 is about 12-13 bps.

3.4 Adding More Randomness

The last thing we did is that we want to add more randomness in this model, so we change the tree as following, on each node, we randomly select 1/3 features, and use the best from them. Following is the hist for this model.

The correlation of this model is about 60% compared to the former one, and the out of sample r^2 is about 10-11 bps. From Figure 19, we can still see a fat tail in this distribution, so we finally use both of these models to do the aggregation.

4 Neural Network

4.1 Data

The version of data used in Neural Network is v3. Firstly we dropped all row with NAN vol. Then we replaced 0 vol with the minimum value of vol column. Doing this, we assume the 0 vol caused by very low liquidity within observing window, rather than missing value.

After get rid of all zero and missing vol in data, we generated a new column of fut_ret/vol. In Neural Network models, instead of predicting fut_vol, we predict fut_ret/vol. Before calculating R-square, we multiply our prediction by vol to get our prediction for fut_vol.

In this section, the package we used is *torch*. We split data into 2:2:6 portion and use them as test, validation and training data. Due to the limitation of computation ability, we did not use cross validation to tune the hyper parameter and only tuned them on a grid.

4.2 Module

In order to build a fine tuned network for this problem, we tried different module.

- Activation: We tried PReLU and it is better than vanilla ReLU
- Drop Layer: We add $p = 0.5$ drop layer after each fully connected layer, when the number of neurons is big.
- Regularization: we add L1/L2 regularization term in our optimizer to shrink the weights.
- Batch Normalization: We normalize the output of each hidden layer.

4.3 Observation and Best Network

4.3.1 Observation

We observed that after build the neural network deep, it tends to predict the mean value of whole set's fut_ret. This will lead to very low even negative R square. No matter how we change hyper parameters, after the network converges, this phenomenon happens.

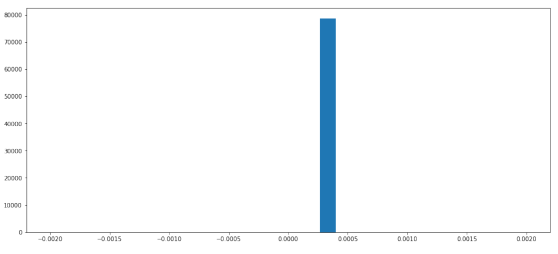


Figure 20: Deep NN's Prediction

To avoid this, we can only apply very simple network structure. We did not benefit from depth of neural network.

4.3.2 Best Result

The best tuned hyper-parameters are given below.

- one hidden layer, four neurons
- no drop layer
- with batch normalization
- with l2 regularization, alpha 1e-1
- use PReLU as activation function
- use Adam as optimizer, learning rate 1

This network gives 5-9 bps in-sample R square and 7-12 bps out-of-sample R square. The variation mainly due to randomness in initialization. Here we use HE normal initialization.

Here we draw histogram of our prediction. In Figure 21, the left is our prediction's histogram and the right is real fut_ret histogram. This two pictures are in same scale. We can see our prediction is near 0. Though it cannot capture the fat tail of return. It can mainly predict rise or drop.

If we zoom in, we can see our prediction's histogram more clearly from Figure 22.

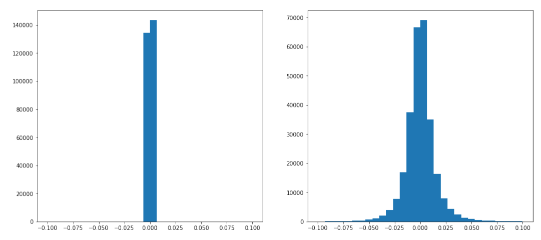


Figure 21: histogram of prediction

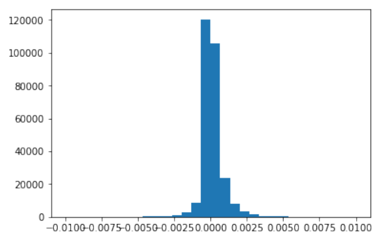


Figure 22: histogram of prediction, zoom in

5 Aggregation

In this section, we want to aggregate all our existing models with decent performance. We have six models, and they are Neural Network, Light GBM, two Regression models with different selected features, two Random Forest one of which add more randomness. You can find detail of all these models from previous sections. They all have near 10 bps R square in test case.

5.1 Methodology

In sense of aggregation, we want to build two layers model. As for lower layer, it refers to six decent models we already build in previous sections. We want to use the prediction of these 6 models as input columns for upper level model. So for upper layer, it uses prediction of different lower level models and predict the fut_ret.

We train six fine tuned existing models on first 150 days, and gives out-of-sample prediction for latter 50 days. Then we split latter 50 days into training and test set for upper layer model. The first three quarter of latter 50 days is training set for upper layer model, and last quarter is test set.

From this point, when we mention train or test, we refer to upper layer model. So both train and test are from latter 50 days of our data.

5.2 How to Deal with NaN

In vol column, there is lots of NaN. There are two ways to deal with this problem. First way is, we fill all NaN value with 0, which is a consistent way. And second way is, we assume the data with NaN vol has different property from data with non-NaN vol, and model them differently.

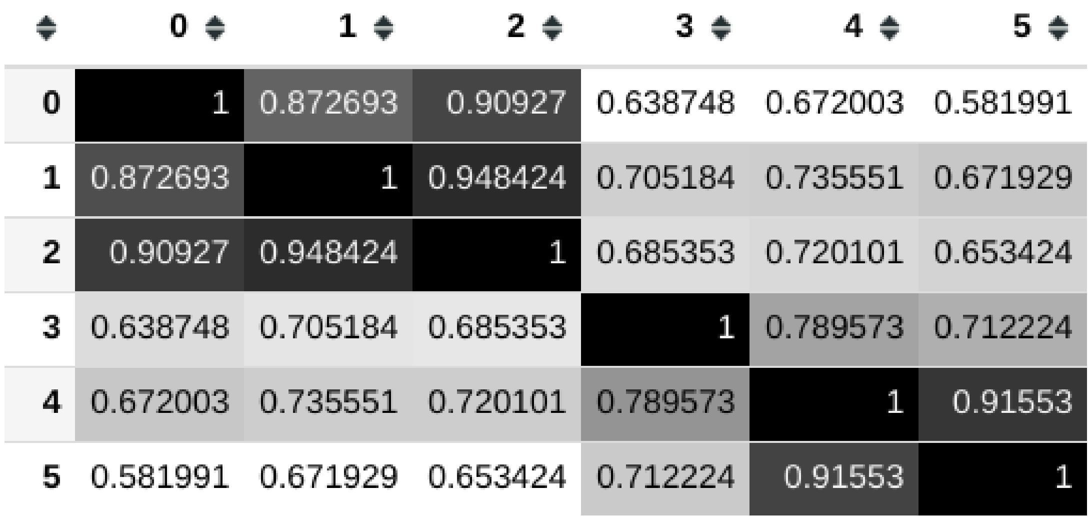
For these two methods, the lower level models have similar performance. But after aggregation, the upper level give different results. It turns out, the second way gives us higher R square. So from this point, we adopt second way.

As for how to model data with NaN vol, we use linear regression on X1 to X7. Because we have an observation that when vol is NaN, 99% of data gives 0 fut_ret. Instead of giving 0 prediction, which is a kind of data-driven, we use linear regression, the most explainable model to predict.

So when we do the test, firstly we split the data to two according to whether vol is NaN or not. Than we use two-layer model to predict non-NaN vol data, and linear regression to predict NaN vol data. Then we join these predictions together as our final result.

5.3 Method

First, we see the correlation of predictions on training set from lower level models. Note that, training set is in sense of upper level model,it is still test set for lower level models.



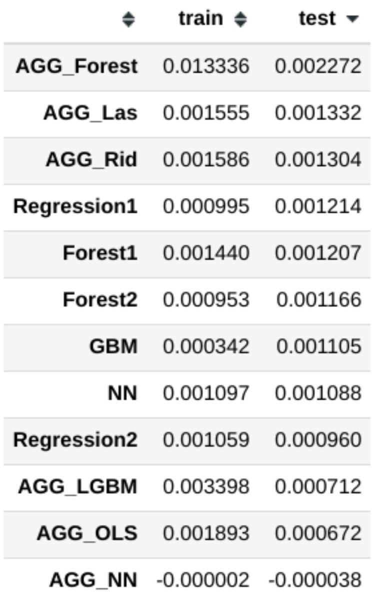
	0	1	2	3	4	5
0	1	0.872693	0.90927	0.638748	0.672003	0.581991
1	0.872693	1	0.948424	0.705184	0.735551	0.671929
2	0.90927	0.948424	1	0.685353	0.720101	0.653424
3	0.638748	0.705184	0.685353	1	0.789573	0.712224
4	0.672003	0.735551	0.720101	0.789573	1	0.91553
5	0.581991	0.671929	0.653424	0.712224	0.91553	1

Figure 23: Correlations between features in training set

Now we build model for upper level. The methods we tried list in below.

- Neutral Network: Even after the simplest net, the prediction is mean of data, which is the same problem we talked in previous section.
- OLS: Bad approach, as there is high correlation between features
- Lasso: very small penalty(1e-7) leads to choose only one feature. We let it choose two features, otherwise upper level do not add value.
- Ridge
- Random Forest
- Light GBM

Figure 24 gives all train and test R square from lower level models and upper level models we tried.



	train	test
AGG_Forest	0.013336	0.002272
AGG_Las	0.001555	0.001332
AGG_Rid	0.001586	0.001304
Regression1	0.000995	0.001214
Forest1	0.001440	0.001207
Forest2	0.000953	0.001166
GBM	0.000342	0.001105
NN	0.001097	0.001088
Regression2	0.001059	0.000960
AGG_LGBM	0.003398	0.000712
AGG_OLS	0.001893	0.000672
AGG_NN	-0.000002	-0.000038

Figure 24: Results(in order of test R square)

So aggregating did give us better result. We choose Random Forest aggregation with 6 other models as our final choice.

When using random forest as our upper level model, feature importance is:

- Neural Network: 0.26931832
- Regression 1: 0.1825856
- Random Forest 1: 0.17319282
- Random Forest 2:0.14946527
- Regression 2: 0.13878568
- Light GBM: 0.08665231

6 Further Possible Improvements

After the final presentation, we actually find that there may still be some possible improvements to further enhance our prediction. One (and probably the most important one) is to use ”fut_ret”/”vol” rather than ”vol” itself as the dependent variable and exclude ”vol” from our regressors.