Machine Learning Final Project

Team member: Wenhao Hu, Yuan Hu, Chengyu Huang, Yinzhi Wang

Table of Contents

[Summary 2](#_Toc483600588)

[General data exploration 2](#_Toc483600589)

[Notation 2](#_Toc483600590)

[Discrete feature 2](#_Toc483600591)

[Continuous feature 2](#_Toc483600592)

[Using historical value in machine learning 4](#_Toc483600593)

[Linear regression 4](#_Toc483600594)

[Neural Network 6](#_Toc483600595)

[Random Forest 6](#_Toc483600596)

[Extreme Gradient Boosting (XGBoost) 8](#_Toc483600597)

[Motivation 8](#_Toc483600598)

[Data exploration 8](#_Toc483600599)

[Model training 9](#_Toc483600600)

[Importance of synthesized features 11](#_Toc483600601)

[Extra Trees 12](#_Toc483600602)

[Model ensemble 12](#_Toc483600603)

# Summary

In this project, we started from data exploration. Then we considered different models, including linear regression, neural network, random forest, extra trees, and xgboost. The final result comes from model ensemble.

# General data exploration

## Notation

We will denote feature value for a given underlying at a given time by x{t, id} or y{t, id}. For example x2{t=10, id=8} is the value of feature x2 for id 8 at timestamp 10. x{t-1, id} refers to the feature value at the previous available timestamp to t for the given id. Similarly, x{t+1, id} refers to the feature at the next available timestamp to t.

## Discrete feature

There are 10 discrete features: x2,x6,x13,x25,x28,x29,x30,x42,x46,x51. Among them, x2, x13, x28 only depend on time, not individual underlying security.

x2 could be something related to the whole market. The correlation between x2{t, id} and y{t, id} is -1.79%.

x13 and x28 have very regular pattern. Starting from timestamp 0, x13 follows recursive pattern “2,2,2,2,1” and x28 follows “1,2,2,2,2”. There are only a few exceptions for this pattern. If each time point represents one trading day, then those exceptions may correspond to holidays. Therefore our guess is that 1 in x13 and x28 stand for one weekday (Monday to Friday). The signal from weekday (e.g. Thursday return being higher than Tuesday’s) is probably very small. Thus we can consider removing x13 and x28 from regression.

x25, x30, x51, x42 are more related to each other. All of them are integers ranging from 1 to 11. x25 and x51 won’t be 1 at the same time. x30 and x42 won’t be 1 at the same time. Most of the time, x30 and x51 are both 1 or both not 1. When one of them is 1 and the other is not, the second one usually is a big number. Same pattern exists for x25, x42 pair. x30{t+1, id} and x42{t+1, id} have high correlation with y{t, id} (23.3% and -24.2% respectively). Furthermore, (x25{t+1,id}+x30{t+1,id}) and (x51{t+1,id}+x42{t+1,id}) are more correlated with y{t, id}.

x6 of each underlying changes slowly over time. It looks like some fundamental feature of the underlying security.

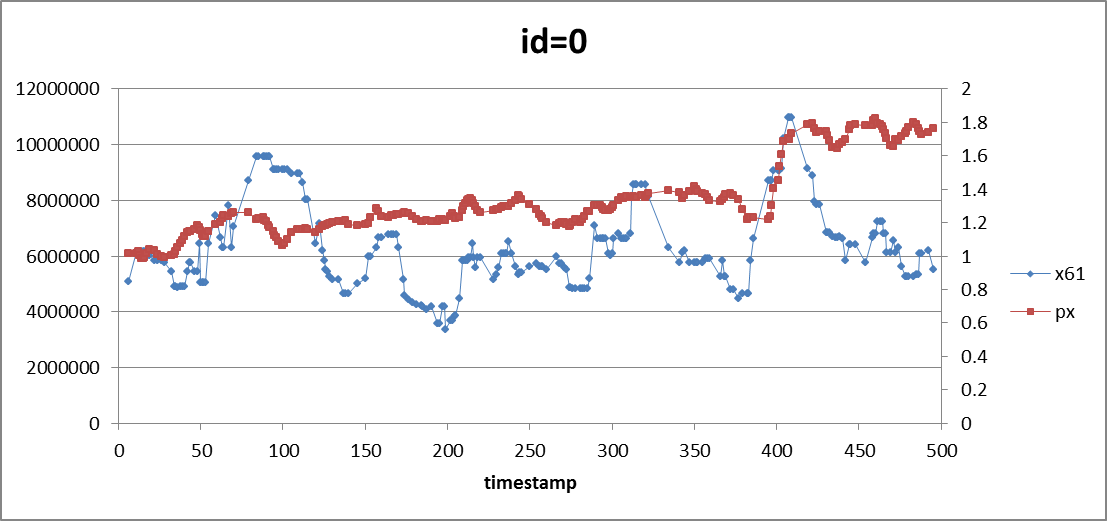
x29 are usually 1 or 3, and seldom become 2. We may consider using one-hot-encoding on it.

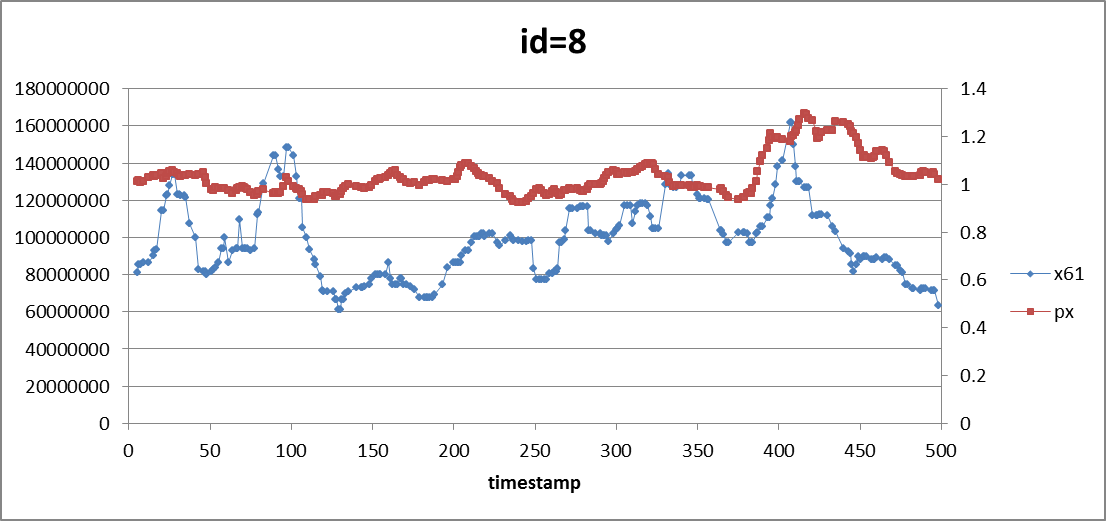
x46{t, id} has a positive (1.6%) correlation with y{t, id}.

## Continuous feature

There are 17 continuous features. x61 is always positive and has kind of log normal distribution. All the other continuous features have bell shape distribution centered close to 0.

Let’s first look at x61. If we think y as security return, then x61 has high correlation with security price. Below are 2 examples. x61 could be something related to the trading volume of the underlying. One way to normalize x61 is by using log return of x61, i.e. ln(x61{t, id}/x61{t-1, id}). In the first timestamp of some underlyings, x61 is 0. We need to special handle it.





We notice that a few features have high correlation (more than 20%) with previous y value. Such features include x17, x22, x49, x39. In other words, if we are allowed to look forward, we can use future values of them to more accurately predict current y!

Current or historical values of some features (e.g. x17, x22) also have relatively high correlation (around or above 2%) with y. We will discuss how to boost their correlation with y using historical time series.

## Using historical value in machine learning

Time series data may have autocorrelation, which can help prediction. For example, a stock price could demonstrate momentum or mean reversion. However, it is not easy to incorporate time series related feature into machine learning. One simple way to add some historical information is calculating moving average of some features or return (unfortunately we don’t have access to historical y in the test). We noticed that correlation can be boosted by using moving average. The table below shows how correlation with y increases with moving average window size. Window size of 1 is equivalent to not using moving average. When window size is 3, we calculate new feature x22\_avg3{t,id}=mean(x22{t,id}, x22{t-1,id}, x22{t-2,id}).

|  |  |  |
| --- | --- | --- |
| window size | x22 | x17 |
| 1 | 2.52% | 1.83% |
| 3 | 3.80% | 3.14% |
| 5 | 4.53% | 3.63% |

Moving average technique is also proved to increase prediction accuracy in our machine learning algorithm.

# Linear regression

Linear regression is the first attempt to build models. We hope to select a manageable feature set and use them to predict y. To start with, we manually plot the histograms of all features. Since most continuous features, except x61, have approximate normal distribution, we have no need to kick out some outliers and symmetrize the data.

To have a tough picture about the correlation between each feature and y, we first calculate the correlation matrix, and temporarily eliminate the nature of some categorical variables. The result is as following:

x22 0.0249892738141

x42 -0.00420657083337

x16 0.00099109531002

x17 0.018377663366

x30 0.00374679630541

x39 -0.00105240024912

x37 -0.00414168507656

x2 -0.0180132799944

x6 -0.00364487166947

x0 -0.0139979800721

x46 0.0156793139604

x61 -0.00517015035843

x65 -0.00357287767287

x53 0.0144026707995

x5 -0.000513159227908

x63 0.00127210825705

x49 0.010035693178

x21 -0.00379347752734

x25 0.00638947314136

x13 0.000986299006499

x28 0.00356738703551

x66 -0.00466806651276

x51 -0.00765786566378

x9 0.00363016564707

x64 -0.00112417121843

x62 0.00167477209072

x29 0.00120658402816

weight 0.00365830687918

According to the figures, there is no obvious “golden feature”. The highest correlation is between x22 and y, which is only 0.0250. By performing naïve linear regression of y on all continuous X features, the r2 score is unstable and sometimes negative when choose different the train sets and test sets.

As mentioned in the data exploration part, we found for the same id, previous values of some features have high correlation with y or shifted y. We replace the original features with these shifted features to X (the beginning and end points of the time series are omitted) and perform naïve linear regression again. This time the result of weighted r2, which is around 0.30, is much improved.

Lasso regression is used to reduce the number of features. We normalize all continuous input and use Lasso regression with initial alpha = 0.01, The feature number shrinks from 17 to 9 and unweighted r2\_score is 0.332. For cross validation, we only use previous data as train set to predict future y value. Besides, we also tried LassoLarsIC to do feature selection, with criteria = ‘bic’ or ‘aic’ respectively. The selected feature sets can be used as candidates for input of other advanced models later.

Ridge regression is another linear model we used. We use python package Ridge\_CV package to conduct cross validation and select alpha. However, there is a problem that the embedded fold selection is random, so we may use the future data to predict the previous value. The result is not satisfactory when we use the model to predict the test data, so we only use the linear model as a benchmark comparing with other models.

In addition to continuous features, we hope to take advantage of the information included in the categorical features. The first attempt is to use one-hot encoding instead of the original label encoding. We start from x29 since it only has three possible values. After replacing x29 with 3 dummy variables, we redo the linear, Lasso and Ridge regressions. With different selected features, the improvement is unstable and trivial.

We also attempt to handle x6 and x46 with one-hot encoding. Although x6 and x46 have high correlation with y and several variables are selected from Lasso, they range from 1 to 10, which means 10 variables need to be added, and do not improve r2 score significantly. Therefore, we still decide to keep label encoding for x6 and x46.

# Neural Network

We also tried neural network to build model. We start with inputting all features, and need to preprocess the data. The first and simple pre-processing approach is zero-center the data, and then normalize them. The method does not work categorical variables, so we keep the original values of categorical variables.

To train the NN model, we set the train set to be the data with time-stamp <400 and the test set to be those with time-stamp >=400. We first use the default parameters implied in the sklearn.neural\_network.MLPRegressor package, and choose hidden layer as (25, ). The r2 score is negatively close to zero.

When tuning the parameters, we tried different combinations of:

solver: {‘lbfgs’, ‘sgd’, ‘adam’} – adam runs more efficiently in this case.

alpha: range from 1e-9 to 1e-3 – alpha tends to be small.

hidden layer (a, b): ‘a’ ranges from 10 to 40, ‘b’ ranges from 1 to 30. – When b, which is the number of layers increases, we observe that the performance gets worse off. The reason is that as the neural network gets deeper, the overfitting will become a problem and the prediction power for test data will be weak.

learning\_rate: {‘constant’, ‘invscaling’, ‘adaptive’} – The default setting is ‘constant’, we prefer the learning rate to be ‘adaptive’ to ensure training loss keep decreasing.

activation: {‘identity’, ‘logistic’, ‘tanh’, ‘relu’} – We tried different activation functions, and finally choose ‘relu’, because as the network become deeper, we don’t want to suffer from dead neuron problem.

random\_state: None or any seed. – We tried different random generators, since the initial state may lead to different model and influence the performance.

Almost all r2 scores are negative when apply the model to test set. The best performance(around -10bps) is usually achieved when there is only one or two layer. We conduct cross validation manually by using different train and test sets, like use time-stamp from 0 to 300 to predict y of time-stamp from 300 to 400. Still, the results are similar, indicating that neural network suffers severe overfitting problem and may not be a good model choice in this case.

In neural network, we don’t consider the time series effect or use shifted variables.

# Random Forest

In this part, we start by just using Lasso regression and random forest. Here are the features we choose using BIC:

x46, x0, x16, x61, x53, x22\_avg, x17\_avg, x2\_avg, x25\_30\_avg

We also tried AIC, and the features this model choose are:

x16, x39, x0, x61, x65, x53, x49, x66, x64, x62, x22\_avg, x17\_avg, x2\_avg, x25\_30\_avg, x42\_51\_avg, x46

We apply these features to random forest respectively, and the parameters we use for trees are:

N\_estimators = 2000, max\_depth = 10, min\_samples\_leaf = 250

The result of simple random forest is not table, so we developed further on random forest. In this new model, we use two kinds of time window to train y, long-term and short-term. Long-term is from t-299 to t and short-term is from t-99 to t. For some timestamps, information too long ago doesn’t matter any more, so short-term will have more prediction power than long-term, and vice versa.

For each set of data points in one time stamp, we update our time window, thus we can get new updated features for the model. Before each training, we cleaned our training data and applied Lasso regression to select features. In the new model, BIC performed better than AIC. In theory, BIC is more sparse than AIC, it can give us features with significant information and ignore those features which might have weak information but will also bring a lot of noises.

We use features get from BIC to train our long-term and short-term trees. So we can get two predicted results here. But which result should we choose? That depends on the result of y{t-1}. We know from y{t-1} whether short term can predict y{t-1} well. If short term predicts y{t-1} well, we will use short term data to predict y{t}, otherwise we will switch to long term to predict y{t}. For each testing data, we retrain the whole model from Lasso regression to random forest training. We find there is stickiness in the results. Short time window works take lead for a while and long time window also will works well for a while.

For the random trees, we use:

n\_estimators = 100, max\_depth = 10, , min\_samples\_leaf = 250

We also use one trick after we get the predicted result. We divide the predicted result by a denominator the shrink our result because we want to minimize our variance. This is sort of like variance and bias trade-off. We found that after we divided the result by a denominator which ranges from 1.5 to 4, our R2 became better. Because when we get a predicted y{t} which is too big or too low, the denominator will reduce the impact and make the variance smaller.

In the out of sample results, the R-squared is about 20~35bps if we know y{t-1}. However, we don’t know y{t-1} in this project. So if I simply use the data 0 to T to predict T to T+100, my result goes down to negative several bps. However, the R2 is not bad if we only need to predict short time results. So I solved the y{t-1} problem by using forecast y{t-1} as real data. Since the predicted result we use is y{t-1} divided by a denominator and the real result is the predicted y{t-1}, so there are still differences between predicted result and real result here. So the long-term and short-term window will keep updating. Data before timestamp 400 is used for training and data after is using as out-of-sample. The out-of-sample R-squared using this method is about 3~6bps.

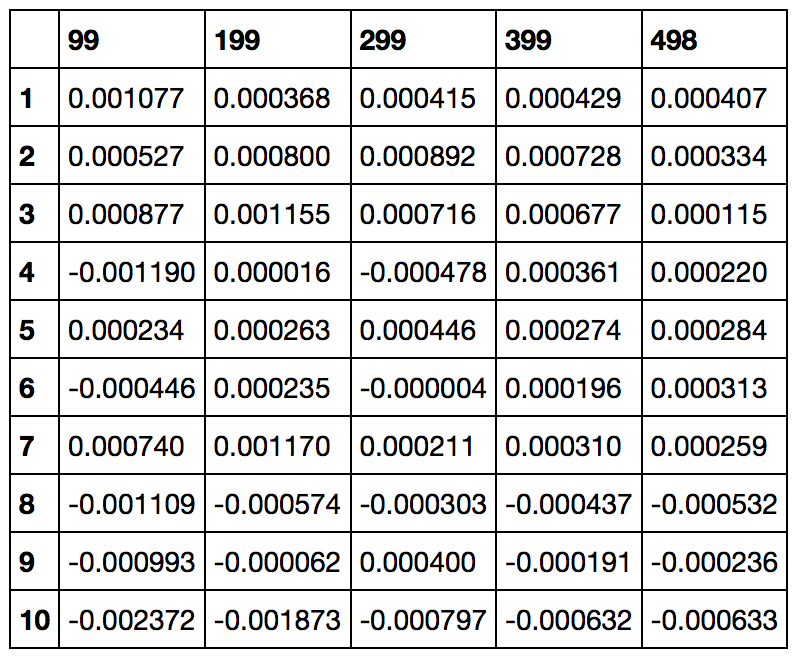
In conclusion, I think this method is more useful in practice than in this project. In practice, we know y{t-1}, which is the key indicator in this method.

# Extreme Gradient Boosting (XGBoost)

## Motivation

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient and portable. Its robustness and flexibility datasets has been approved in Kaggle data science competitions. Considering the characteristics of financial panel data (complex features and extremely low signal-to-noise ratio), it should be able to show excellent without too much data exploration.

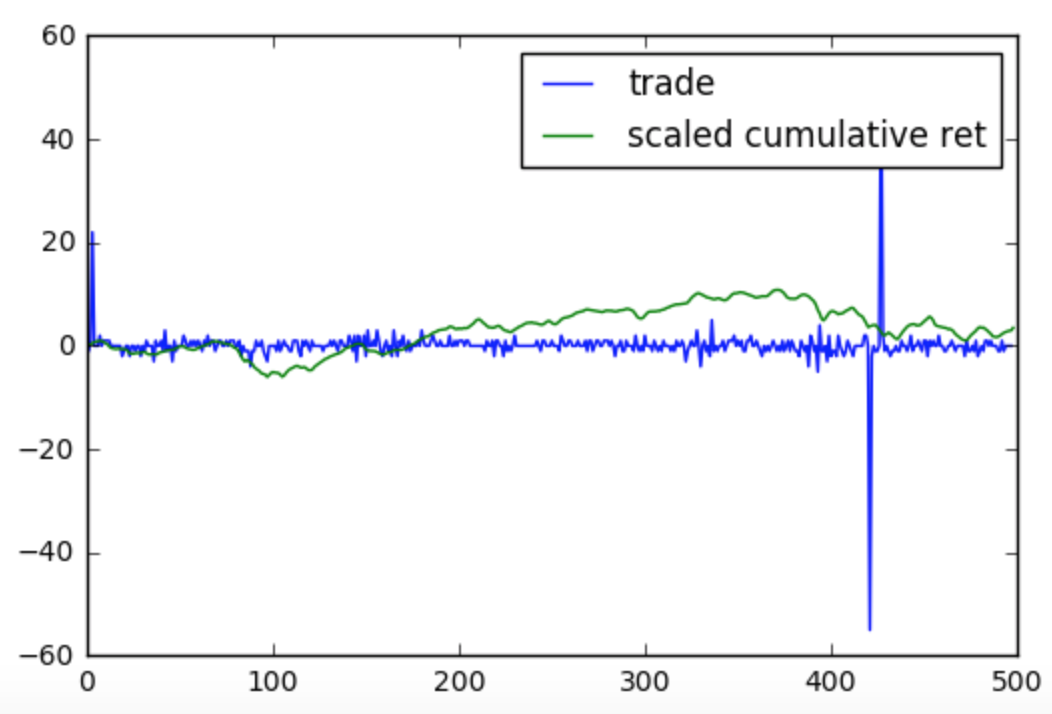
## Data exploration



According to earlier section, some synthesized feature exhibits higher correlation with y, e.g. the moving average of x22 and x17. Following this idea, we also checked the differencing of all continuous variables, which show their value to be considered in our model. Other than that, we think the categorical feature x2 is very special since it’s homogeneous under timestamp. We think it might be a macro economic indicator affecting all the equities. To verify our assumption, we looked into the average y under a certain time period and the same x2. As we can see in the above table, they become more and more stable across all the time period. At the same time, the average y is roughly monotonically decreasing as x2 increases. As a result, we decide to replace x2 with the averaged y and remove the old x2. Considering the robustness of xgboost, we decide not to abandon too many features but throw in more synthesized features unless it has obviously correlation with original data. The noise feature will be handled by more randomization and regularization when we train the model.

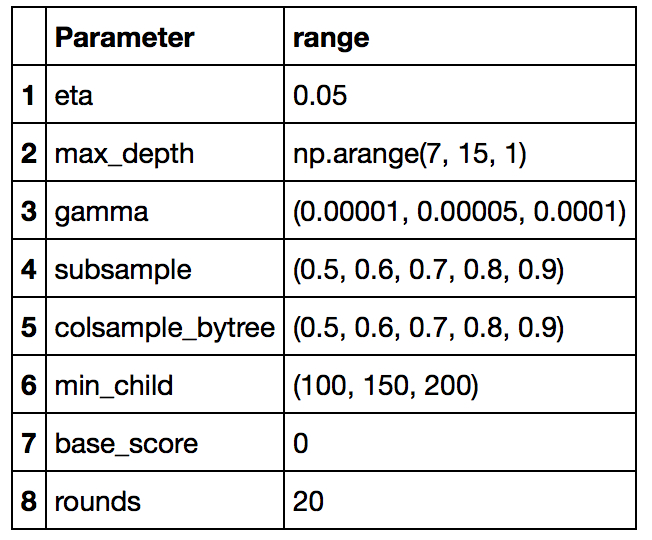
## Model training

In our cross validation, the training set we choose is timestamp in the interval of [0, 400], the testing set is timestamp in the interval of [430, 498]. This pick of training/testing set splitting is based on the observation that there is an event happened during the time period [400, 430]. More explicitly, we made an assumption on the dataset that the change of constituent id in the panel data is mainly due to the rebalance of portfolio. The disappearance of an ID from the dataset indicates that it’s sold out in this timestamp. As we can see in the following graph, it shows the change of assets number as time. There are two extremely sharp up/down peaks at



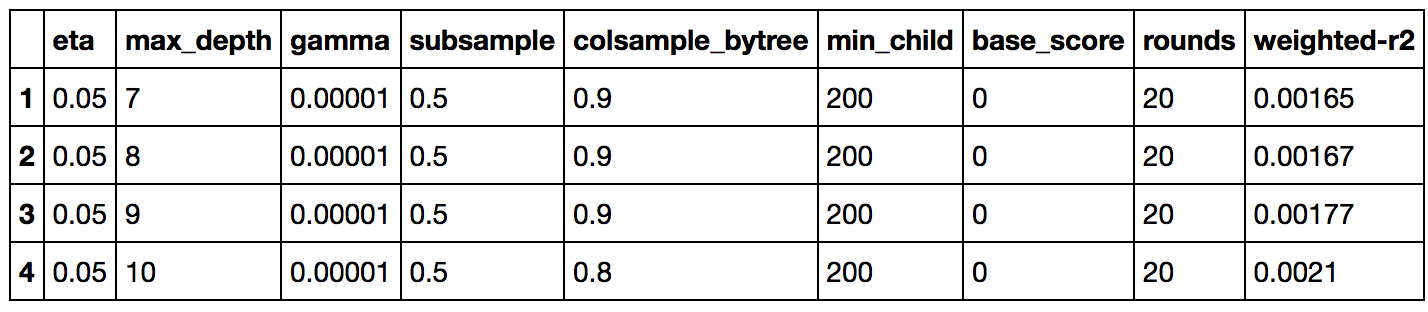
timestamp 420/425. In other words, large amounts of stocks are sold/bought at these two timestamps. We also compared the scaled averaged y with it and found a long-term downside around this period. As a result, it is reasonable for us to assume it to be an event of stop loss. Due to the extremely volatile environment of the market, excluding them from validation set can make our CV more reliable.

To find out the optimal parameters for XGB The parameter sets, we attempted to exploited the grid search method to iterate through all the possible combination of parameters. Based on intuitive estimation, we choose the range of parameters to be as following table. Among all the parameters, the ‘max\_depth’, ‘subsample’, ‘colsample\_bytree’, ‘min\_child’, ‘base\_score’ are the most influential components on the performance of xgboost. The ‘base\_score’ of 0 is based on the average level of ‘y’. The ‘max\_depth’ can largely decide how aggressive the training dataset is fitted. Typically, a higher ‘max\_depth’ will lead to a higher in-sample r-squared, while noise rather than information will be fitted if it’s too large. ‘subsample’, and ‘min\_child’ are all parameters to control the happening of overfitting. Note that ‘colsample\_bytree’ and ‘gamma’ are two very important parameters based on the fact that large amount of weak features are considered. To fully extract the information from these weak features, column sampling can brought them into splitting decision. At the same time, ‘gamma’ can prevent the potential of overfitting.

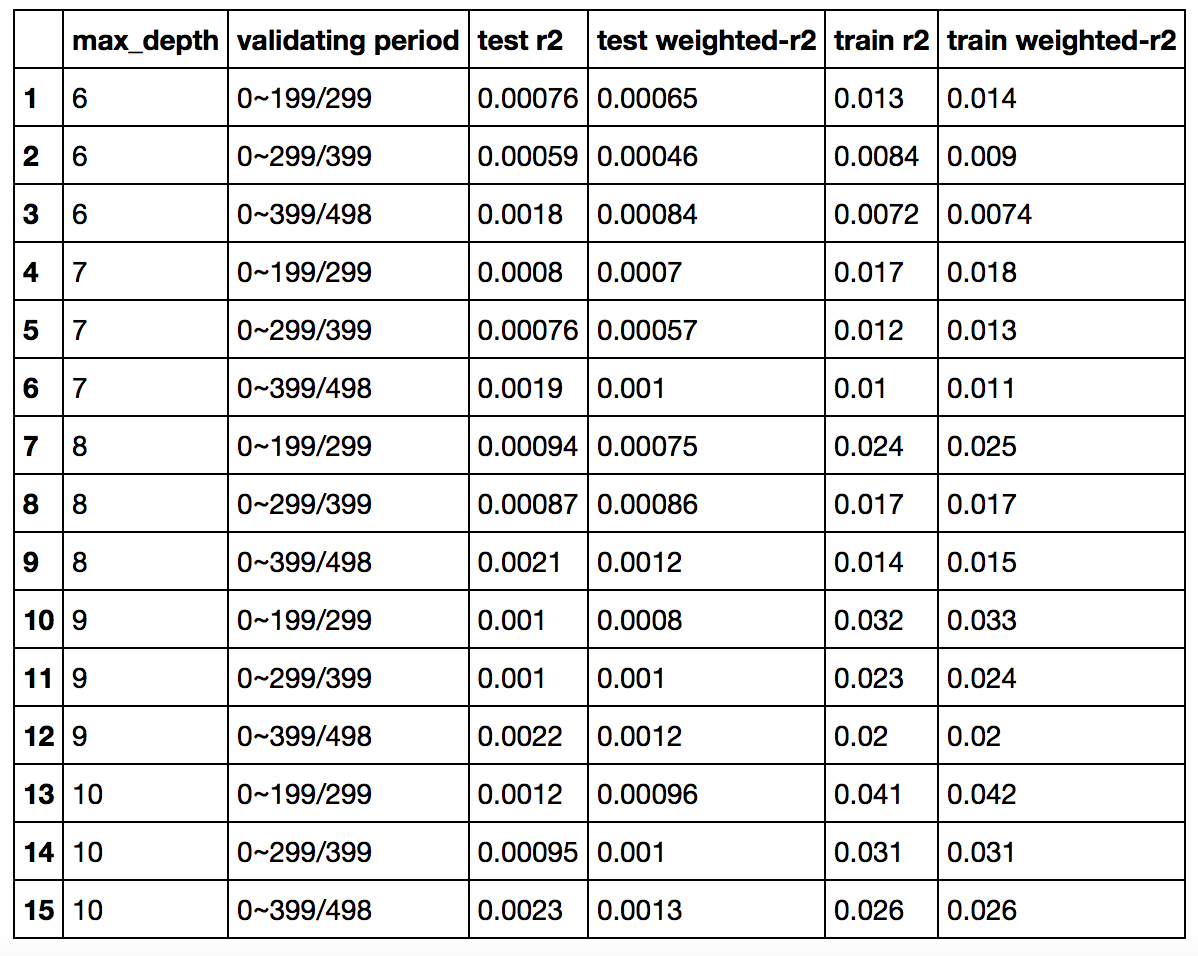


Except for the model-related parameters in XGBoost, we aggregated 10 xgboost estimators with different random seeds to remove the performance instability of our estimators. As a result, the out-of-sample weighted r-square of our estimator is more reliable and the optimal parameters can be correctly selected by grid search.

Due to the time limit, we only finished the calculations with gammas of 0.00001 and 0.00005. . The optimal parameter sets are concluded in the following table:

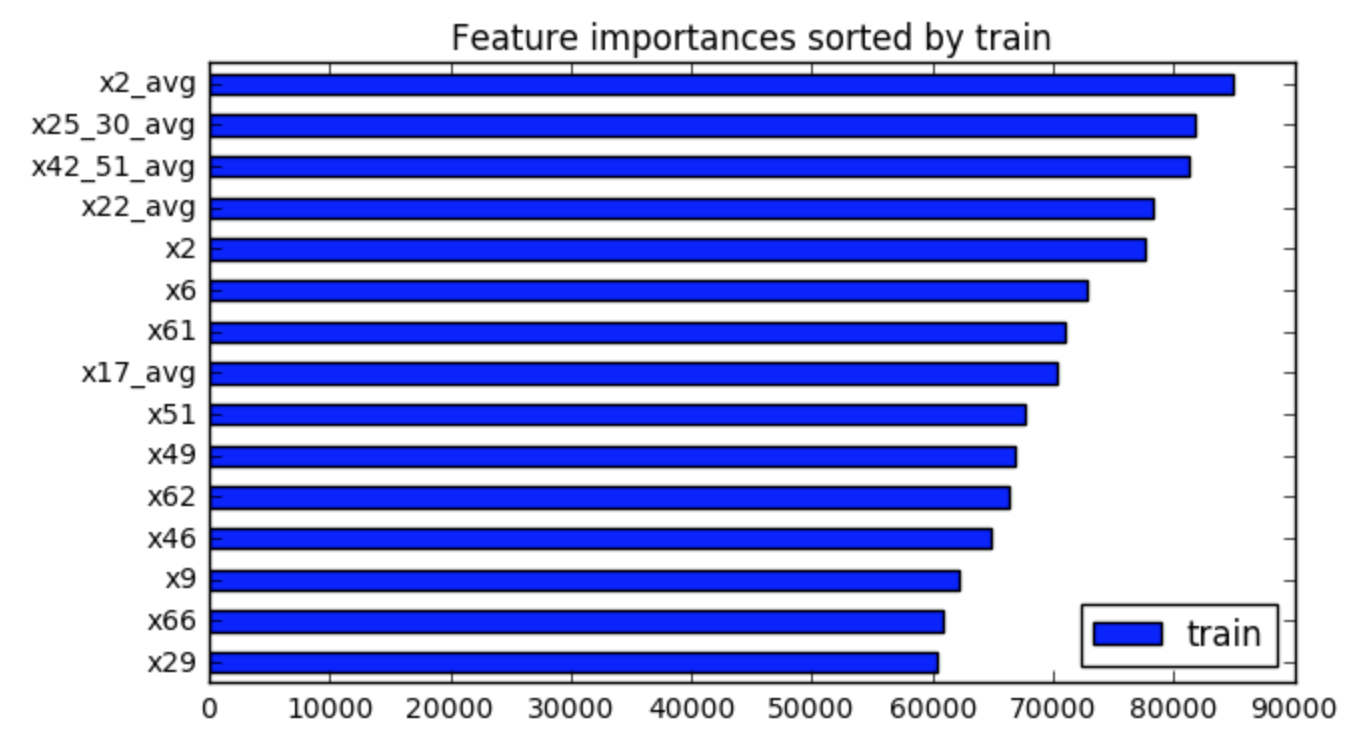


Obviously, the highest weighted r-squared is 0.0021 which is 3-4 basis pointer better than depth of 7,8,9. However, the pick of parameters should be parsimonious considering the extremely low SNR of our data. So they can also be a candidate since they are less than 5 basis off 0.0021. To judge which one is the optimal depth, we used forward chaining cross validation (FCCV) to test their robustness when data are more flexible. In FCCV, we divide the dataset into 5 parts with respect to timestamp, e.g. [0,99], [100,199], [200,299], [300,399] and [400,498]. Their validating scores are the averaged weighted r-squares when we train and test datas in a rolling style. More explicitly, we train models with [0,99] data and test it against [100, 199] data etc. Since the holdout data is only 25% of the training data, we decide to skip the first CV and start from [0,199] data. The validating score is listed in the following table. The validating score is increasing as the depth increases which proves that 10 is the optimal parameter. The corresponding score is 0.00067.



## Importance of synthesized features

The average feature importance is shown in the above plots. Top 4 important features are synthesized features which clearly proves the values of these synthesized features.



# Extra Trees

The tuning of ExtraTree is totally different from XGBoost since it’s much more time-consuming. But its robustness even compared with XGBoost gives us more space to do feature engineering. Here, we decided to choose the following features to use.

x16,x53,x63,x49,x21,x66,x9,x64,x62,x39,x37,x0,x61,x65,x5,

x22\_avg,x17\_avg,x2\_avg,x25\_30\_avg,x42\_51\_avg,x29\_1,x29\_2,x29\_3,x6,x46

They include all continuous features except that x22 and x17 are replaced with their 5 day moving average. We also include 5 day moving average of x2, (x25+x30), (x42+x51). We use one-hot-encoding for x29, i.e. x29\_1, x29\_2, x29\_3 represent x29==1, x29==2, x29==3 respectively. x6 and x46 are also chosen.

After manually playing with Extra Trees, we chose the following hyper parameters

n\_estimators=1000

max\_depth=10

min\_sample\_leaf=250

Data before timestamp 400 is used for training and data after is using as out-of-sample. The out-of-sample R-Squared is about 12~13 bps.

# Model ensemble

A random forest classifier is used to ensemble the ExtraTree and XGBoost estimators. More explicitly, we train the ExtraTree and XGBoost estimator with training data and calculate their absolute residual respectively. A binary variable which indicates the estimator with lower absolute residual is synthesized. For instance, if this value is 1, the absolute residual of XGBoost is lower and it should be the better model to be selected. A random forest binary classifier is trained on this variable against all the features of training set. It is expected to tell us which estimators should be used accordingly. Again, the weighted r-square of ensembled model is evaluated through CV. When training/validating is set to 4:1, the weighted r-square is raised by 1-2 basis points. More details can be found in the submitted script.

Note that, the submitted version is based on a linear regression ensembling scheme. It seems that the xgboost doesn’t play an import role for some reasons. We can’t locate the error without holdout data.