Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

Forecasting Cryptocurrency Prices

Author: Xinyu Yan (CID: 01804622)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2019-2020

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Acknowledgements

I have to start by thanking my supervisor, Paul. From giving me early constructive remarks to giving me advice on the next steps, he was important to this thesis as well as my master's study at Imperial.

I was really lucky to be admitted by Imperial College, where I met many excellent classmates and had unforgettable time with them. I need to express my deepest appreciation to all of my classmates, who gave advice on this thesis, and supported me in this hard self-isolation time.

None of us predicted the changes with the pandemic. Accommodating such a radical shift in such a short time is not simple. I would like to thank my parents, who care about me all the time, the NHS, the government, and my courage to face it all.

Abstract

Statistical models combined with machine learning have gained importance when they have become ubiquitous in modern life. In this paper we aim to use some machine learning models such as linear regression, gradient boosting and random forest to predict the high-frequency time series of prices of BitCoin (BTC), one of the most popular crypto-currencies in the market. The models are created by taking existing BTC's high frequency historical limit order book market data, extracting the best features and finding a mathematical representation that has acceptable fidelity to the data. Moreover, we will explain the end-to-end process of performing forecasting, such as explanatory data analysis, feature engineering, feature selection, model training techniques and model performance comparison in this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

In this thesis we review the various methods of forecasting cryptocurrency prices, notably bitcoin (BTC) [Nak09]. To the best of our knowledge, previous work in this area was undertaken by Silantyev [Sil19]. While contemporaneous and one-step-ahead prediction are of academic interest, we seek to obtain tradeable results. To this end we attempt to predict the mid-price up to one hundred ticks ahead.

We will perform the prediction on the time series data downloaded from BitMex, a crypto-currency exchange. Before we start to train the models on the dataset, we perform exploratory data analysis on the features of the trade data and market quotes from the dataset. Then we use some feature engineering techniques to select the best derived features for the purpose of training. Some machine learning models such as linear regression, random forest and gradient boosting will be used to predict the future tick data. Next, we print each combined out of sample prediction evaluation line to compare the results.

For the outline of the thesis, we start introducing the concepts of the linear regression and ensemble methods in chapter 2. In this chapter, random forest and gradient boosting tree are explained. From chapter 3 to chapter 6, detailed steps are specified on how to organise the Bitcoin time-series data to perform the prediction using the models. We carry out some statistical analysis on the Bitcoin time series data such as bid-ask spread, bid and ask size, bid and ask price and etc in Chapter 3. In chapter 4, we explain the methodologies on feature augmentation based on the original features. We then extract more than 2000 features from historical order book data. In chapter 5, we specify how we use the bespoke method to select the features from large feature set and rank them. In that way, we try to get the most useful data for forecasting. In chapter 6, we use grid search and differential evolution to optimise the parameter space for the random forest and gradient boosting tree forecasting respectively. Then we evaluate the performance of the models when making decision for next 100-day prediction and make comparison. In chapter 7, we apply the feature selection by using random forest and data compression. Also, to speed up the computational time, we use the map-reduce method to do the feature selection for large amount of feature set. In chapter 8, we propose future directions of research. In chapter 9, we

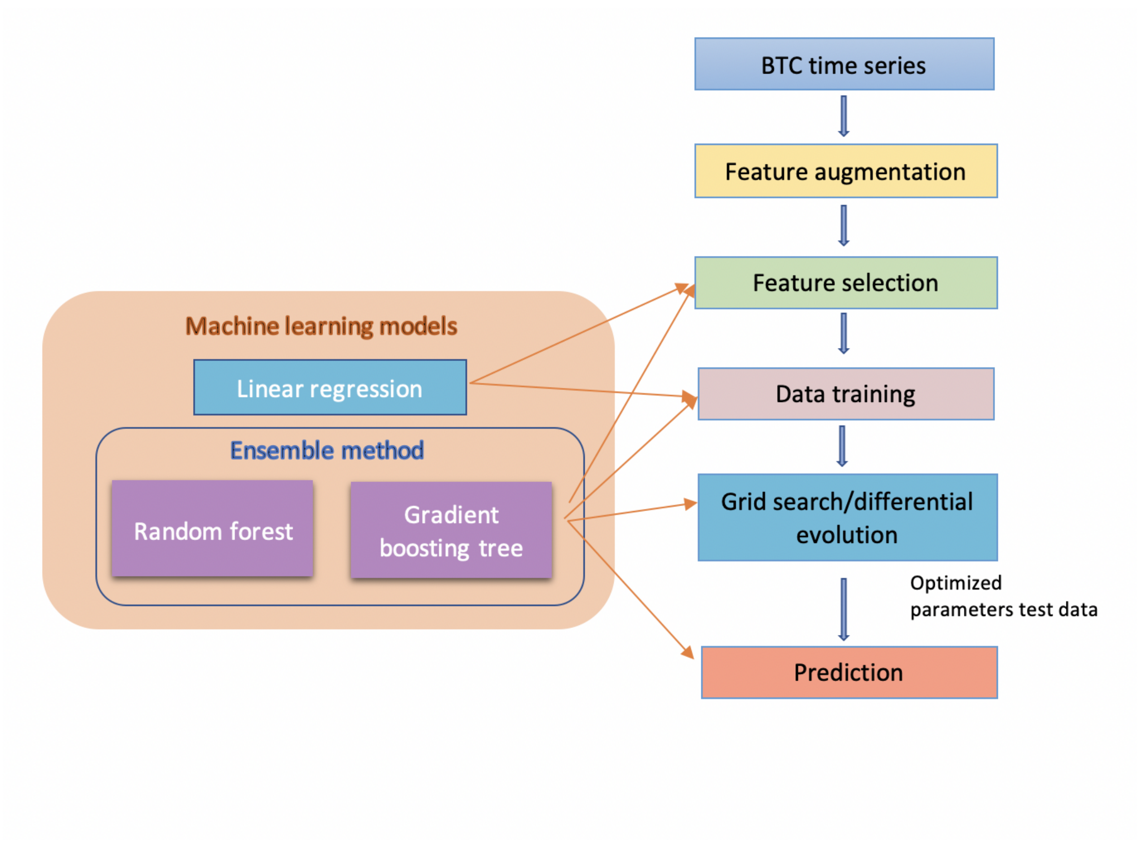


Figure 1.1: flow chart

make the conclusion of how the feature selection by using linear regression (LR) and random forest (RF) can affect the forecasting results by using the models like linear regression, random forest and gradient boosting.

The whole process can be shown as figure 1.1

1.2 Forecastability

The **efficient-market hypothesis (EMH)** [Fam70] states that asset prices reflect all available information. A direct implication is that it is impossible to 'beat the market' consistently on a risk-adjusted basis since market prices should only react to new information.

In opposition to EMH, the late Mark Joshi cited [Jos08] "the apocryphal story of the two economists who see a ten dollar bill lying in the gutter. The first one goes to pick it up but the second one tells him not to be silly as if it were real, someone else would have already picked it up." He states further that "the true moral of this story is that market efficiency only works if someone does not believe in it—the first person who does not believe in market efficiency picks up the ten dollar bill and thereafter it is no longer lying in the gutter."

One should question to what degree the cryptocurrency markets are efficient. They differ from the more traditional markets, such as equities and foreign exchange, in the high degree of fragmentation, stemming from the proliferation of independent spot and derivatives exchanges, such as Binance, BitMEX, Bybit, Coinbase, Deribit, Huobi, Kraken, OKEx.

1.3 BitMEX

In this study we focus on the BitMEX exchange, which is advertised as "The Next Generation of Bitcoin Trading Products. Up to 100x leverage. Trading without expiry dates. Industry-leading security. Welcome to Bitcoin's most advanced trading platform."

BitMEX accepts only bitcoin, and settle contracts in bitcoin.

Rather than trading bitcoin, BitMEX trades its derivative—a perpetual contract XBTUSD. Each contract is worth 1 USD, so when you buy one contract at 7259, it means you have to pay 1 USD worth of bitcoin at the price you specified. When you sell the contract instead you receive 1 USD worth of bitcoin at the price you sold at. You are buying contracts and not USD or BTC.

How does BitMEX guarantee that the price of the contract will follow the exchange rate of BTCUSD? This is implemented with the **funding rate**. The price of the contract might differ from BTCUSD. BTCUSD is distilled into an index (.BXBT), which is the average of BTCUSD from other exchanges. This is the 'price of BTC' and is usually slightly different from the 'price of the contract'. When the price of the contract is less than the price of BTC, users with a long position get paid the funding rate so you have an incentive to buy the contracts. This pushes up the price of the contract thus realigning it with BTCUSD. At the same time users with a short position will pay the funding rate, so they have an incentive to reduce their position by buying contracts. This also pushes up the price of the contract to match BTCUSD. When the price of the contract is more than the price of BTC, the opposite happens: users with a long position pay users with a short position; this incentivizes one to sell the contract, pushing down its price to be closer to BTCUSD.

The funding rate is a zero sum game: longs pay shorts (or vice versa); BitMEX does not get any fees out of the funding rate.

1.4 Literature Review

Time series forecasting has been a very popular research topic because it has wide applications in many different areas, especially in high frequency trading. As a high frequency trader, he or she will gain an edge in trading if he or she has some good forecasting models. The most popular feature to predict is the mid-price (average of bid-ask) of the assets, as happens in market-making activities. We examine a few papers which propose using some innovative forecasting methods to predict the financial time series.

Linear regression model has various forms of classical models such as auto regressive model (AR), moving average model (MA), then the auto-regressive moving average model (ARMA) and auto-regressive integrated moving average model (ARIMA) [Ham94]. It was found that ARIMA model had a good fitness for the forecasting of time series such as sales of retail products in one step and multiple steps models [PR15].

To handle the non-linear data, some authors use hybrid models to do the predictions on those data and they manage to show some improvement in accuracy. Inspired by the linear error correction model, Giot, Laurent, and Petitjean proposed the ARIMA model combined with the support vector machine (SVM), i.e. the ARIMA-SVM model [PGP10] to improve the accuracy of the prediction for historical time series data which are non-linear. Zhang [Zha01] proposed a hybrid ARIMA and neural network (ARIMA-NN) model that is useful in forecasting real-world time series which are linear and nonlinear. The paper shows that the hybrid ARIMA-NN model improves the accuracy in forecasting the real-world time series which have both linear and non-linear patterns. Another authors, Xu *et al.* [WX19] use a novel hybrid LR-DBN model which combines linear regression (LR) and deep belief network (DBN). This hybrid model shows good result in predicting the cyclical time series like sunspot data.

The above articles mentioned about the prediction on non-financial data. Lai [RKL08] established a novel forecasting model by using evolving and clustering fuzzy decision tree for stock market data in Taiwan Stock Exchange Corporation (TSEC). The model uses technical indexes and historical data as input features and then perform Fuzzy Decision Tree (FDT) and Genetic Algorithms (GA) for prediction. This FDT-GA model looks promising when predicting the TSEC market data.

The models mentioned above are shown to be able to handle non-linear data but they are not suitable to high frequency trading which has more stringent requirements on the performance of the models. Also, as opposed to lower frequency data, high frequency data provides more market information which leads to more accurate forecast [Han12]. With the aim to extract as much information as possible while making the prediction over the next 100 steps of BTC's mid-prices within time constraint, we decide to use the classical machine learning methods such as linear regression (LR), random forest (RF) and gradient boosting tree (GB). However we put more attention to feature engineering to improve the prediction of the BTC prices.

Chapter 2

Background Reading

2.1 Gauss-Markov Theorem

Gauss-Markov Theorem (GMT) [GMT] tells us if the function is linear, incorporates all k variables, and assume e_n is white noise, and uncorrelated to x_n, then the ordinary least squares (OLS) estimators has the lowest variance among all linear unbiased estimators, which has a direct ratio of $\frac{\sigma_\epsilon^2}{N}k$. σ_ϵ^2 is the homoscedastic finite variance of the error term. N is the number of observations and k is the number of features. So GMT is the fundamental reason why OLS is the statistical workhorse in econometrics.

Ordinary least squares (OLS) is a linear least squares method to estimate unknown parameters in a linear regression model. OLS chooses linear function parameters of a set of explanatory variables by the principle of least squares. This thesis uses linear regression and related methods to determine the best features for forecasting bitcoin prices.

2.1.1 Linear Regression (LR)

There are five key assumptions for the linear regression model. However, in practice, on our data, these assumptions are likely to be violated.

- **Linearity:** The expected value of dependent variable (y) is a linear function of each independent variable, if holding the other independent variable fixed.In our model, we enrich features with their nonlinear transformations to better satisfy the linearity assumption.

- **Independence:** The residuals of the fitted model are independent from each other.

- **Homoscedasticity:** The variance of the residuals is constant with respect to the dependent variables.

- **Normality:** The errors are from the normal distribution of unknown mean and variance which can be estimated from data. This assumption is used to calculate the 'confidence' intervals as the well-known analytical expressions.

- **Multicollinearity:** For multiple linear regression, it is critical for this assumption assumes minimal or no linear dependence between the prediction variables.

The equation of linear regression can be represented by

$$Y = \theta^T x$$

where θ is the model's parameter vector including the intercept term θ_0 and x is the feature vector with $x_o = 1$. Since LR solved with OLS method has closed-form solution, parameter vector θ with small dimension can be computed easily in very short time.

To find the model parameter θ for related but more complicated models, gradient descent method can be used. Gradient descent is a generic optimization algorithm used in many machine learning algorithm. It changes the parameters of the model to minimize the cost function iteratively. The steps of gradient descent are outlined below.

1. Firstly, we initialize the model parameters with random values. This is what we called **random initialization**.

2. Secondly, we need to measure the changes of the cost function when its parameters change. Therefore we compute the partial derivatives of the cost function with respect to the parameters $\theta_0, \theta_1, ...\theta_n$

$$\begin{aligned} \tfrac{\partial J(\theta)}{\partial \theta_0} &= \tfrac{1}{m} \textstyle\sum_{i=1}^{m} \left(h\left(x^i\right) - y^i\right) \\ \tfrac{\partial J(\theta)}{\partial \theta_1} &= \tfrac{1}{m} \textstyle\sum_{i=1}^{m} \left(h\left(x^i\right) - y^i\right) x_1^i \end{aligned}$$

   Similarly, the partial derivative of the cost function with respect to any parameter can be denoted by

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h\left(x^i\right) - y^i\right) x_j^i$$

   We can then compute the partial derivatives for all parameters at once using

$$\begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} = \frac{1}{m} x^T (h(x) - y)$$

   where $h(x)$ is

$$h(x) = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$$

3. After computing the derivatives we then update the parameters as given below

$$\begin{aligned} \theta_0 &= \theta_0 - \tfrac{\alpha}{m} \textstyle\sum_{i=1}^{m} \left(h\left(x^i\right) - y^i\right) \\ \theta_1 &= \theta_1 - \tfrac{\alpha}{m} \textstyle\sum_{i=1}^{m} \left(h\left(x^i\right) - y^i\right) x_1^i \end{aligned}$$

where α is the learning rate parameter. We could update all the parameters

$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

4. Finally, we repeat the steps 2 and 3 until the cost function converges to the minimum. We need to be careful with the α adjustment since the cost function will take longer to converge if α is too small, or overshoot the minimum values and fail to converge if α is too large.

2.1.2 Limitations of the Gauss–Markov Theorem (GMT)

However, even if all the assumptions are true, OLS does not necessarily yield the minimum mean squared error [JS08]. First of all, there can be some non-linear biased estimators with lower variance such as James-Stein estimator ([JS08]), which is a member of a type of Bayesian estimators that dominate the maximum-likelihood estimator. Secondly, there are also linear biased estimators with lower mean squared error such as Ridge or Lasso.

The common case in financial datasets is that the assumptions for Gauss-Markov Theorem are not true. When the assumptions are false, the situation will be of course more problematic. The most common error is specification error. The true model may not be linear in its parameters, and the statistical model can omit variables or incorporate spurious variables. There are also errors such as stochastic regressors, which means the explanatory variables are measured with error, are endogenous, are mutually dependent; and non-spherical error, which means the errors are heteroskedastic or auto-correlated. As a result, econometric models are often inefficient, which have either high bias or high variance.

We all know that the financial system is very complex. Even experienced researchers may miss important variables. The use of classical statistics is somewhat challenging. One motivation for using machine learning in finance is that it relies on more realistic assumptions.

In finance, there is no strong theoretical reason to discard biased models, since financial systems are too complex for expecting fully well-specified (unbiased) models. Minimizing variance subject to zero bias (MVUE) is the wrong objective. Machine learning methods, though, can achieve lower mean squared error than econometric models through cross-validation; complexity reduction such as regularization; and ensemble methods such as bootstrap aggregation, boosting, and stacking.

2.2 Ensemble Methods

There are three ways to build ensemble estimators, bagging, boosting, and stacking. "Bias-variance decomposition" is an important tool from the angle of bias and variance to explain the generalization performance of the learning algorithm. Bias measures the learning ability of a single model, while variance measures the stability of the same model over different data sets. In general, models with lower variance exhibit higher bias, and models with lower bias exhibit higher variance. This is called Bias-Variance Tradeoff. This is not always the case, but is common. We attempt to mitigate the Bias-Variance Tradeoff by fitting a large number of predictors, which have a relatively low bias, and then generate an ensemble forecast.

2.2.1 Bagging Model - Random Forest

Bagging means we need to train several models independently on the bootstrap of the data-set. These several weak learning models are trained in parallel and are aggregated using a deterministic procedure. For a sufficiently large number of uncorrelated learners, bagging algorithms effectively reduce the variance,so it is a general procedure that be used for these algorithm that have a high variance, typically decision trees. It is extremely useful for high-frequency trading including in bitcoin because the data-set is extremely large.

The basic building block for the random forest is a decision tree. Decision trees are a series of questions, with each question narrowing our possible values until the answer is narrow down enough to make a single prediction.

Formally speaking, for a $p-$dimensional random vector $X = (X_1, X_2, ..., X_p)^T$ which represents an input (or predictor variables) of real value, with a random variable Y representing the the real-valued output (or prediction results), an unknown joint distribution $P_{XY}(X, Y)$ is assumed given. The goal for predicting Y is to find a prediction function $f(X)$. The prediction function is constructed by a loss function $L(Y, f(X))$ and the objective is to minimize the expected value of the loss.

$$E_{XY}(L(Y, f(X)))$$

The subscripts denote expectation with respect to the joint distribution of X and Y.

It is easy to understand that $L(Y, f(X))$ is a measure of the distance between $f(X)$ and Y. The loss function penalizes the value of $f(X)$ that are a long way from Y. Normally, we choose loss function L squared error loss $L(Y, f(X)) = (Y - f(X))^2$ for regression and zero-one loss for classification (which is not included in our method below).

And it turns out that minimizing $E_{XY}(L(Y, f(X)))$ for squared error loss will give the conditional expectation

$$f(x) = E(Y \mid X = x)$$

So it is also known as the regression function. In the classification situation which this thesis doesn't include, minimizing $E_{XY}(L(Y, f(X)))$ can also give the probability of belonging to a particular class.

Ensembles construct f with respect of a collection of 'base learners' $h_1(x), h_2(x), ..., h_N(x)$ and these base learners are combined to give the final 'ensemble predictor' $f(x)$as in [CZ12].The ensemble predictor averages the predictions of the base learners.

$$f(x) = \frac{1}{N} \sum_{N=1}^{N} h_N(x)$$

In random forests, let us denote by the jth base learner a tree $h_j(X, \Theta_j)$, Θ_j here is a collection of random variables and are independent from each other for $j = 1, 2, ..., J$. The training data are then set as $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $x_i = (x_{i,1}, ..., x_{i,p})^T$ denotes the p input variables and y_i denotes the response, and a particular realization θ_j of Θ_j, the fitted tree is denoted by $\hat{h}_j(x, \theta_j, D)$. This is the original given by Breiman [Bre01], while in practice the random compunent θ_j is implicitly instead of explicitly to inject randomness in these two ways. Firstly, as for a method of bagging, each tree is fit into an independent bootstrap sample from the original dataset. The bootstrap sampling is random which gives a part of θ_j. Secondly, when splitting the nodes, the best split is found over a randomly selected subset of m input variables instead of all p variables,

The whole algorithm can be described as below.

Algorithm 1: Random Forest

```
Let $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ denotes the training data, with $x_i = (x_{i,1}, ..., x_{i,p}^T)$.
for j in range(1, J) do
    Take a bootsstrap sample $D_j$ of size $N$ from $D$;
    With the bootstrap sample $D_j$ as training set, fit a tree by using binary recursive
      partitioning;
    Starting with all observations in a single node;
    while Stopping criterion has not met do
        Select $m$ predictors randomly from the $p$ available input variables;
        Find the prediction result among all functions on the $m$ predictors from last step;
        Split the node into two descendant nodes using the split from last step.
    end
end
```

Random Forest is a typical method to use bagging, which is to train decision trees in parallel and aggregate. Decision trees are sensitive to the data on which they are trained, resulting quite different when training data is changed. So the predictions are always quite different. What's more, decision trees are computationally expensive to train. In that way, it carries a huge risk of overfitting, and tend to find local optimisation instead of global optimisation because they can't go backward after they have made a split.

Random Forest is instead to address these weaknesses. It illustrates the power of bagging, combining these decision trees into one model. The trees are run in parallel and there is no interaction between each decision tree while building the trees. Random Forest runs by constructing a multiple of decision trees at training set and outputting the mean prediction of the individual trees.

There are also some useful modifications for random forest when aggregating decision trees. Firstly, the number of features that can split on at each node is limited on the percentage of total hyperparameter, which ensures the random forest does not rely heavily on any individual feature and makes good use of all potential predictive features. Secondly, each tree uses a random sample from the initial data-set, adding further randomness that prevents overfitting.

2.2.2 Boosting Model - Gradient Boosting Tree

Boosting is very different with bagging. In contrary to learn in parallel like bagging, boosting learn iteratively. One classifier is going to produce some error, and then this error is going to be adapted by next model. It is a sequence of weak classifier, that one weak classifier is going to improve over the previous one. Boosting algorithm forms a strong learner by training several weak learners iteratively, where subsequent models learn from prior errors, and aggregating their forecasts using a deterministic procedure. Because of its adaptive nature, boosting is generally more effective than bagging at reducing bias. And it can also be effective at reducing variance.

However, the shortness for the boosting is that since the model needs to be fitted iteratively, it is extremely complicating. Unlike bagging, boosted learners cannot be fit in parallel, consequently, boosting often takes longer to fit. Another reason we don't often use boosting is that, in finance, we care more about overfitting than biased. Bagging is a good compromise because this effective model can be fitted very fast, very intuitive, and relies on very few parameters. There are two main types boosting. One is "Adaboost", which adapts by updating the sample weights. And the other is "Gradient boosting" which adapts by updating the observations.

In the gradient boosting function estimation problem, we have a system consisting of a random response variable y and a set of random explanatory variables $x = \{x_1, x_2, ..., x_n\}$. When giving a training sample $\{y_i, \mathbf{x_i}\}_\mathbf{1}^\mathbf{N}$ of known (y, x) values. The goal, like all algorithm is to find a function $F^*(x)$ that maps x to y. For example, over a certain joint distribution of all $(y, x$, the loss expectation of some certain loss function is minimized

$$F^*(\mathbf{x}) = \arg\min_{F(\mathbf{x})} E_{y,\mathbf{x}} \Psi(y, F(\mathbf{x})) = \arg\min_{F(\mathbf{x})} E_\mathbf{x} \left[E_y(\Psi(y, F(\mathbf{x})) \mid \mathbf{x}\right]$$

Like the other algorithm, frequently used loss function $\Psi(y, F)$ include squared-error $(y - F)^2$ and absolute error $|y - F|$.

The common procedure like is to take $F(\mathbf{x})$ as a member of a parameterized class of functions $F(x; P)$, where $\mathbf{P} = \{P_1, P_2, \cdots\}$ is a set of parameters. Then the parameterized functions can be expressed as

$$F(\mathbf{x}; \mathbf{P}) = \sum_{m=0}^{M} \beta_m h\left(\mathbf{x}; \mathbf{a}_m\right)$$

where $\mathbf{P} = \{\beta_m, \mathbf{a}_m\}_0^M$ The function $h(\mathbf{x}; \mathbf{a})$ in the last formula usually use the simple function of $\mathbf{x}$ with parameters $\mathbf{a} = \{a_1, a_2, ...\}$.

In general, this problem can be changed as choosing a parameterized model $F(\mathbf{x}; \mathbf{P})$ changing the function optimization problem.

$$\mathbf{P}^* = \arg\min_{\mathbf{P}} \Phi(\mathbf{P}) = \arg\min_{\mathbf{P}} E_{y,\mathbf{x}} \Psi(y, F(\mathbf{x}; \mathbf{P})); \quad F^*(\mathbf{x}) = F\left(\mathbf{x}; \mathbf{P}^*\right)$$

For most $F(\mathbf{x}; \mathbf{P})$ and Ψ, we need to use the numerical optimization method to solve the function.

This needs the solution for the parameters in the form below: [Fri99]

$$\mathbf{P}^* = \sum_{m=0}^{M} \mathbf{p}_m$$

$\mathbf{p}_0$ is the initial guess and $\{\mathbf{p}_m\}_1^M$ are successive increments defined by the optimization method.

For example, one of the simplest of the frequently used numerical minimization method to handle the gradient boosting tree is 'steepest-descent' method. The whole formula is as below.

$$\mathbf{g}_m = \{g_{jm}\} = \left[\frac{\partial \Phi(\mathbf{P})}{\partial P_j}\right]_{\mathbf{P}=\mathbf{P}_{m-1}}$$

where

$$\mathbf{P}_{m-1} = \sum_{i=0}^{m-1} \mathbf{p}_i$$

The step is taken to be

$$\mathbf{p}_m = -\rho_m \mathbf{g}_m$$

where

$$\rho_m = \arg\min_{\rho} \Phi\left(\mathbf{P}_{m-1} - \rho \mathbf{g}_m\right)$$

The negative gradient $-\mathbf{g}_m$ define the direction of descent and the last formula is the searching for best function along that descent direction.

Gradient boosting can work in credibly effective in practice. Perhaps the most popular using way is XGBoost. XGBoost employs a number of tricks which can make the whole process faster and more accurate than traditional gradient boosting. Gradient boosting method can be used as a classification, ranking, or regression model using different loss function for the algorithm to minimize. Common objective functions is cross entropy for classification and mean squared error for regression. In this paper, we use common mean squared error to regress and get the prediction.

2.2.3 Stacking

Stacking is also named stacked generalization. It uses a meta-learning algorithm which can learn how to best combine the predictions from some other more base machine learning algorithms. The advantage of stacking is that it is able to take advantage of the capabilities of a wide range of well-performing models on a classification or regression tasks, then make predictions that have better performance than any single model in other ensemble method.

2.3 Differential Evolution

Differential Evolution(DE) was designed to be a stochastic search method [Sto97]. It is a direct search method and has benefit which can be easily applied to experimental minimization where the cost value is from experiment instead of a computer simulation. DE can handle non-differentiable, nonlinear and multimodel cost function. Differential Evolution is a also parallel direct search method that uses NP D-dimensional parameter vectors

$$x_{i,G}, i = 1, 2, ..., NP$$

as a population for each generation G. NP does not change during the minimization process. The initial vector populations are chosen randomly and should cover the entire parameter space. When assuming all random decisions are uniform distributed and preliminary solution is available, the initial population might be found by adding normal random deviations to the nominal solution $x_{nom,x}$. DE generates new parameter vectors by adding the weighted difference between two population vectors to the third one. This is called the operation mutation. The parameters are then mixed with the those of the target vector, to yield trial vector. If the trial vector yields a lower cost function value than the target vector, the trial vector replaces the target vector in the following generation. Each population vector has to serve once as the target vector so that NP competitions take place in one generation.

- **Mutation**

  For each target vector $x_{i,G}, i = 1, 2, 3, ..., NP$, a mutant vector is generated according to

  $$v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_7,G} - x_{r_3,G})$$

  with random indexes $r_1, r_2, r_3 \in \{1, 2, ..., NP\}, F > 0$. The randomly chosen integers are also chosen to be different from the running index i, so that NP must be greater or equal to four for this condition. F is a real constant factor $\in [0, 2]$ which controls the amplification of the differential variantion $(x_{r2,G} - x_{r3,G})$. The figure below shows a two-dimensional example that illustrates the different vectors which plays a part in the generation of $v_{i,G+1}$

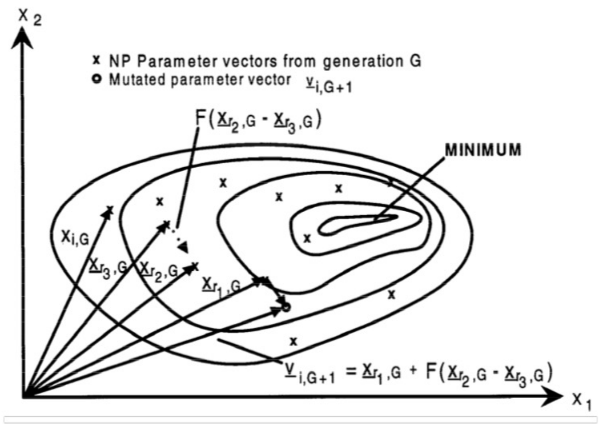

x 2
x NP Parameter vectors from generation G
o Mutated parameter vector v i,G+1
F(x r2,G - x r3,G)
MINIMUM
x i,G
x r3,G
x r2,G
x r1,G
v i,G+1 = x r1,G + F(x r2,G - x r3,G)
x 1

Figure 2.1: Two-dimensional cost function

Source: Differential Evolution, A Simple and Efficient Heuristic forGlobal Optimization over Continuous Space

- **Crossover**

  Crossover can increase the diversity of the pertubed parameter vectors.

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{if } (randb(j) \leq CR) \text{ or } j = rnbr(i) \\ x_{ji,G} & \text{if } (randb(j) > CR) \text{ and } j \neq rnbr(i) \end{cases}$$
$$j = 1, 2, \ldots, D$$

  where $randb(j)$ is the jth evaluation of a uniform random number generator from outcome $[0,1]$. CR is the crossover constant $\in [0,1]$ which has to be determined by the user.

- **Selection**

  To decide whether or not it will become a member of generation $G+1$, the trial vector $u_{i,G+1}$ compared to the target vector $x_{i,G}$ using greedy criterion. If vector $u_{i,G+1}$ yields a smaller cost function value than $u_{i,G}$, then $u_{i,G+1}$ is set to $u_{i,G+1}$; otherwise, the old value $u_{i,G}$ is retained.

Chapter 3

Exploratory Data Analysis

3.1 Raw Data

We have collected streaming data from BitMEX using the WebSocket API, the precise subscription being given by

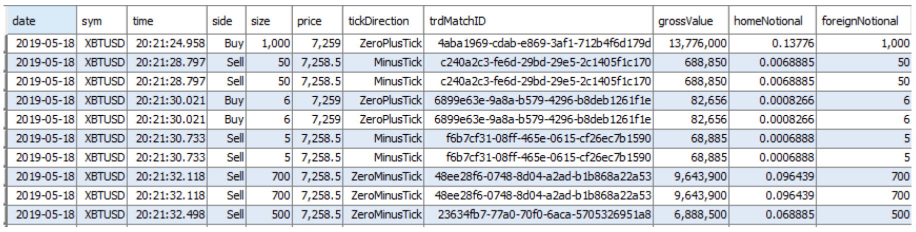
```
wss://www.bitmex.com/realtime?subscribe=instrument,orderBook10:XBTUSD,trade:XBTUSD
```

This data feed is not without issues:

- The order book updates often lag the trade updates. (This would matter in a production trading system, but has little effect on our analysis.)
- A small minority of trades arrive out of order.
- The trade updates do not always match the history of the trade updates, which can be requested using, for example,

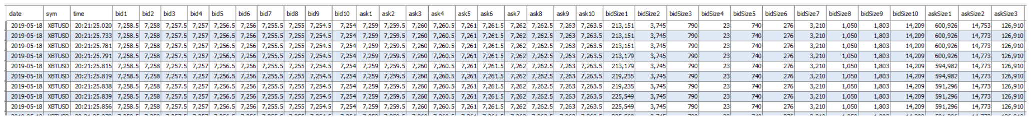
  ```
  https://www.bitmex.com/api/v1/trade?symbol=XBTUSD&reverse=false&startTime
      ↪ =2019-05-19T00%3A00%3A07&endTime=2019-05-19T00%3A00%3A08
  ```

We collected the real-time data using a **kdb+tick triplet** [NBGD19]. The date range covered by the dataset is 2019.05.21 (inclusive) to 2019.05.24 (inclusive) as training data and 2019.05.25 as test data. 3.1 and 3.2 are shown as example for kdb table structure.



date	sym	time	side	size	price	tickDirection	trdMatchID	grossValue	homeNotional	foreignNotional
2019-05-18	XBTUSD	20:21:24.958	Buy	1,000	7,259	ZeroPlusTick	4aba1969-cdab-e869-3af1-712b4f6d179d	13,776,000	0.13776	1,000
2019-05-18	XBTUSD	20:21:28.797	Sell	50	7,258.5	MinusTick	c240a2c3-fe6d-29bd-29e5-2c1405f1c170	688,850	0.0068885	50
2019-05-18	XBTUSD	20:21:28.797	Sell	50	7,258.5	MinusTick	c240a2c3-fe6d-29bd-29e5-2c1405f1c170	688,850	0.0068885	50
2019-05-18	XBTUSD	20:21:30.021	Buy	6	7,259	ZeroPlusTick	6899e63e-9a8a-b579-4296-b8deb1261f1e	82,656	0.0008266	6
2019-05-18	XBTUSD	20:21:30.021	Buy	6	7,259	ZeroPlusTick	6899e63e-9a8a-b579-4296-b8deb1261f1e	82,656	0.0008266	6
2019-05-18	XBTUSD	20:21:30.733	Sell	5	7,258.5	MinusTick	f6b7cf31-08ff-465e-0615-cf26ec7b1590	68,885	0.0006888	5
2019-05-18	XBTUSD	20:21:30.733	Sell	5	7,258.5	MinusTick	f6b7cf31-08ff-465e-0615-cf26ec7b1590	68,885	0.0006888	5
2019-05-18	XBTUSD	20:21:32.118	Sell	700	7,258.5	ZeroMinusTick	48ee28f6-0748-8d04-a2ad-b1b868a22a53	9,643,900	0.096439	700
2019-05-18	XBTUSD	20:21:32.118	Sell	700	7,258.5	ZeroMinusTick	48ee28f6-0748-8d04-a2ad-b1b868a22a53	9,643,900	0.096439	700
2019-05-18	XBTUSD	20:21:32.498	Sell	500	7,258.5	ZeroMinusTick	23634fb7-77a0-70f0-6aca-570532695 1a8	6,888,500	0.068885	500

Figure 3.1: table structure for bitcoin transactions



date	sym	time	bid1	bid2	bid3	bid4	bid5	bid6	bid7	bid8	bid9	bid10	ask1	ask2	ask3	ask4	ask5	ask6	ask7	ask8	ask9	ask10
2019-05-18	XBTUSD	20:21:25.020	7,258.5	7,258	7,257.5	7,257	7,256.5	7,256	7,255.5	7,255	7,254.5	7,254	7,259	7,259.5	7,260	7,260.5	7,261	7,261.5	7,262	7,262.5	7,263	7,263.5
2019-05-18	XBTUSD	20:21:25.733	7,258.5	7,258	7,257.5	7,257	7,256.5	7,256	7,255.5	7,255	7,254.5	7,254	7,259	7,259.5	7,260	7,260.5	7,261	7,261.5	7,262	7,262.5	7,263	7,263.5
2019-05-18	XBTUSD	20:21:25.781	7,258.5	7,258	7,257.5	7,257	7,256.5	7,256	7,255.5	7,255	7,254.5	7,254	7,259	7,259.5	7,260	7,260.5	7,261	7,261.5	7,262	7,262.5	7,263	7,263.5
2019-05-18	XBTUSD	20:21:25.791	7,258.5	7,258	7,257.5	7,257	7,256.5	7,256	7,255.5	7,255	7,254.5	7,254	7,259	7,259.5	7,260	7,260.5	7,261	7,261.5	7,262	7,262.5	7,263	7,263.5
2019-05-18	XBTUSD	20:21:25.815	7,258.5	7,258	7,257.5	7,257	7,256.5	7,256	7,255.5	7,255	7,254.5	7,254	7,259	7,259.5	7,260	7,260.5	7,261	7,261.5	7,262	7,262.5	7,263	7,263.5
2019-05-18	XBTUSD	20:21:25.819	7,258.5	7,258	7,257.5	7,257	7,256.5	7,256	7,255.5	7,255	7,254.5	7,254	7,259	7,259.5	7,260	7,260.5	7,261	7,261.5	7,262	7,262.5	7,263	7,263.5
2019-05-18	XBTUSD	20:21:25.838	7,258.5	7,258	7,257.5	7,257	7,256.5	7,256	7,255.5	7,255	7,254.5	7,254	7,259	7,259.5	7,260	7,260.5	7,261	7,261.5	7,262	7,262.5	7,263	7,263.5
2019-05-18	XBTUSD	20:21:25.839	7,258.5	7,258	7,257.5	7,257	7,256.5	7,256	7,255.5	7,255	7,254.5	7,254	7,259	7,259.5	7,260	7,260.5	7,261	7,261.5	7,262	7,262.5	7,263	7,263.5
2019-05-18	XBTUSD	20:21:25.856	7,258.5	7,258	7,257.5	7,257	7,256.5	7,256	7,255.5	7,255	7,254.5	7,254	7,259	7,259.5	7,260	7,260.5	7,261	7,261.5	7,262	7,262.5	7,263	7,263.5

time	bidSize1	bidSize2	bidSize3	bidSize4	bidSize5	bidSize6	bidSize7	bidSize8	bidSize9	bidSize10	askSize1	askSize2	askSize3
20:21:25.020	213,151	3,745	790	23	740	276	3,210	1,050	1,803	14,209	600,926	14,753	126,910
20:21:25.733	213,151	3,745	790	23	740	276	3,210	1,050	1,803	14,209	600,926	14,773	126,910
20:21:25.781	213,151	3,745	790	23	740	276	3,210	1,050	1,803	14,209	600,926	14,773	126,910
20:21:25.791	213,179	3,745	790	23	740	276	3,210	1,050	1,803	14,209	600,926	14,773	126,910
20:21:25.815	213,179	3,745	790	23	740	276	3,210	1,050	1,803	14,209	594,982	14,773	126,910
20:21:25.819	219,235	3,745	790	23	740	276	3,210	1,050	1,803	14,209	594,982	14,773	126,910
20:21:25.838	219,235	3,745	790	23	740	276	3,210	1,050	1,803	14,209	591,296	14,773	126,910
20:21:25.839	225,549	3,745	790	23	740	276	3,210	1,050	1,803	14,209	591,296	14,773	126,910
20:21:25.856	225,549	3,745	790	23	740	276	3,210	1,050	1,803	14,209	591,296	14,773	126,910

Figure 3.2: table structure for bitcoin quote

We merge the two of table using the as-of-join Q command. By using this merged table we explore some time-series statistics of the raw features, then we augment and enrich these raw features for feature selection stage.

3.2 Data Statistical Analysis

We performed exploratory data analysis on our dataset. Table 3.1 summarizes and Figure 3.3 visualizes the time durations between the trades and trade sizes. The entire dataset was used to produce these tables and figures. The histograms in Figure 3.4 were produced after removing the outliers to improve their readability. The spikes in the trade sizes histogram indicate that the traders tend to gravitate towards round lots, such as 1,000, 5,000, 10,000, and 20,000 USD. There is also a very large number of small trades, as the maximum trade size is 1 USD.

	Time between trades (seconds)	Trade size (USD)
Minimum	0.003	1.00
Maximum	161.986	2,132,104.00
Mean	0.382	5,564.33
Median	0.187	1,000.00
Standard deviation	0.489	19,939.53

Table 3.1: Statistics of the time between the trades and trade sizes.

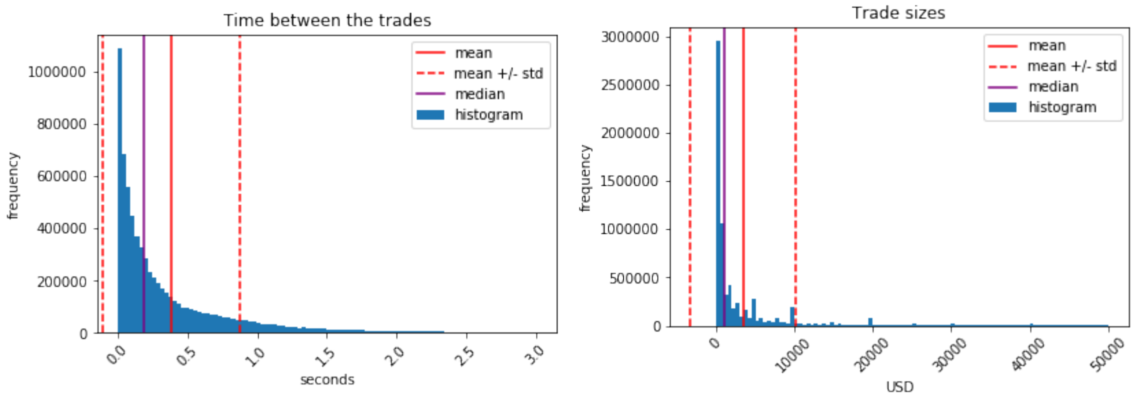


Figure 3.3: Histograms of the time between the trades and trade sizes.

The exploratory data analysis on order book data was restricted to a single day (the first one, 2019.05.21) due to time and memory constraints. Table 3.2 summarises our findings for the top of book. Table 3.3 shows the same statistics for the second level of the order book. The histograms in Figure 3.3 were produced after removing outliers for readability and normalising the ordinates to facilitate comparison.

The bid–ask spread was at 0.5 USD for 98.4% of order book updates. At the remaining times it took on other values, namely 1., 1.5, 2., 2.5, 3., 3.5, 4., 4.5, 5., 5.5, 6., 6.5, 7., 7.5, 8., 8.5, 9., 9.5, 10., 10.5, 11., 11.5, 12., 12.5, 13., 13.5, 14., 14.5, 15., 15.5, 16., 16.5, 17., 17.5, 18., 18.5, 19., 19.5, 20., 20.5, 21., 21.5, 22., 22.5, 23., 23.5, 24., 24.5, 25., 25.5, 26., 26.5, 27., 27.5, 28., 28.5, 29., 29.5, 30., 30.5, 32., 34., and 37.5.

	Bid–ask spread (USD/BTC)	Level 1 bid size (USD)	Level 1 ask size (USD)
Minimum	0.003	1.00	1.00
Maximum	161.986	21,358,456.00	7,734,094.00
Mean	0.382	479,731.10	420,949.27
Median	0.187	346,155.00	282.032.00
Standard deviation	0.489	549,476.82	456,613.77

Table 3.2: Top of book statistics.

	Level 2 bid size (USD)	Level 2 ask size (USD)
Minimum	1.00	1.0
Maximum	21,230,335.00	7,734,094.00
Mean	147,245.67	125,111.31
Median	27,161.00	24,755.00
Standard deviation	308,731.97	267,666.41

Table 3.3: Order book level 2 statistics.

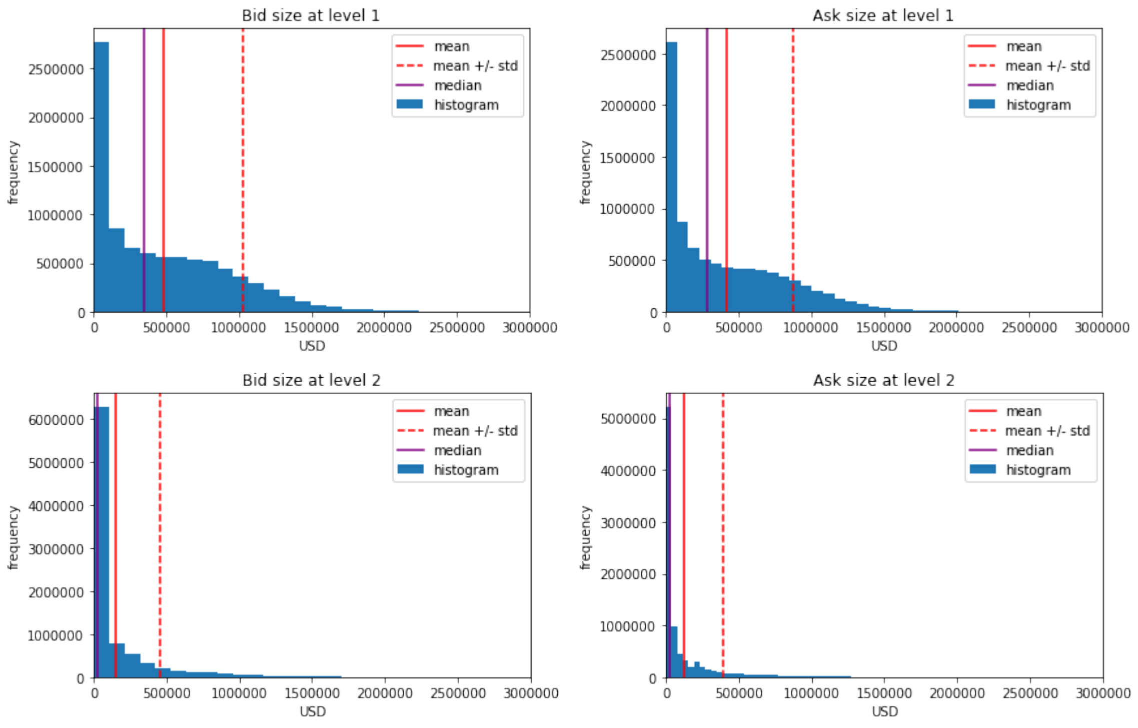


Figure 3.4: Histograms of the bid and ask sizes at levels 1 and 2 of the order book.

We also make some other explanatory data exposure. We transpose these tick data to minute data to make a clear visualization. This paper shows the 2019-05-22 minute data as figure 3.5.

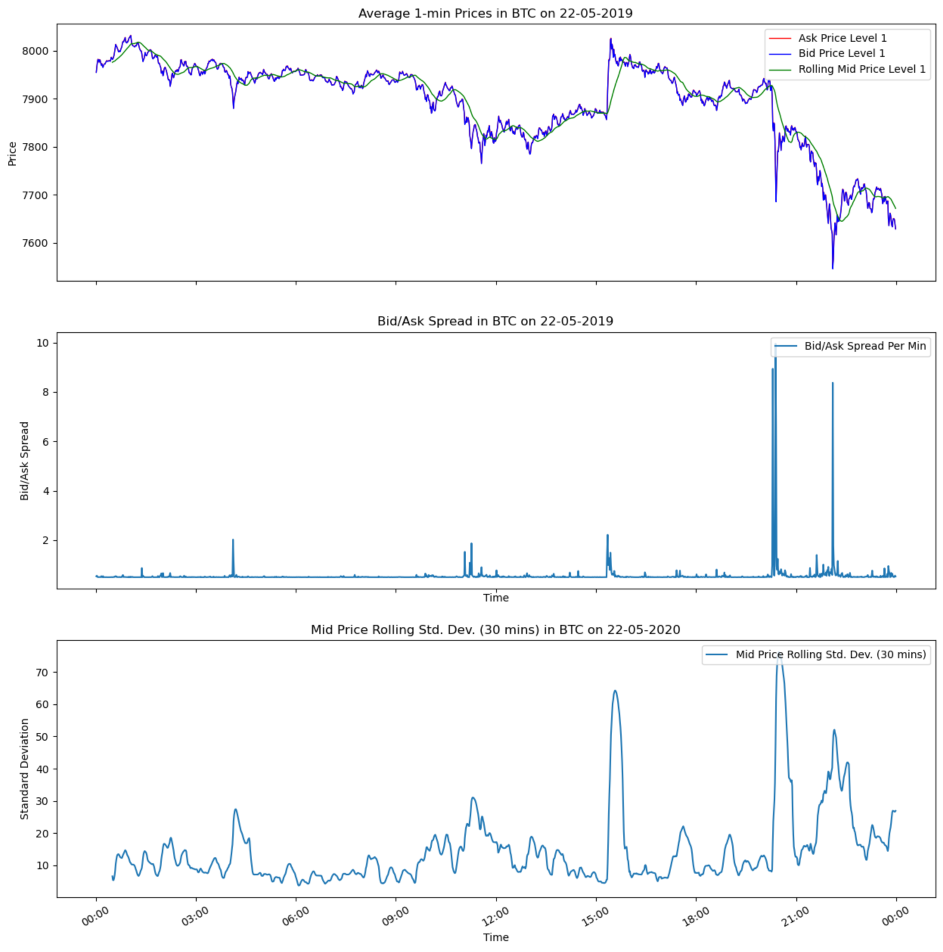


Figure 3.5: other explanatory data

The first figure shows the average bid, ask and mid price. We notice that the BTC price fluctuates significantly at night. The second figure shows the statistics data of bid-ask spread. There are five large bid-ask spread during the day, indicating five periods of poor liquidity. The third picture shows there are three violent volatility during the day. The first is at around 15:00 o'clock, the second is at around 21:00, the third is at around 23:00.

Chapter 4

Feature Augmentation

4.1 Methodology

In our parlance, a tick corresponds to a trade reported by an exchange. We augment the data from the trade feed, which contains

- the **exchange timestamp**;
- the instrument **symbol**, e.g. XBTUSD;
- the **side**—Buy (on the ask side of the order book) or Sell (on the bid side of the order book);
- the **size**—the number of XBTUSD contracts; each contract has 1 USD notional;
- the **price**—the price of the contract in USD per one BTC;
- the **tick direction**—the direction that the price moves as a result of the trade: down (MinusTick); up (PlusTick); stays constant, but depletes part of a level on the asks side of the order book (ZeroPlusTick); stays constant, but depletes part of a level on the bids side of the order book (ZeroMinusTick);
- the **trade match identifier**—a unique trade id;
- the **gross value**—the number of satoshi[1] units that were exchanged;
- the **home notional**—trade value in BTC; this is equal to $\frac{\text{foreign notional (USD)}}{\text{price (USD/BTC)}}$;
- the **foreign notional**—trade value in USD for XBTUSD contracts

[1]A **satoshi** is the smallest unit of a bitcoin, equivalent to 100 millionth of a bitcoin. Bitcoins can be split into smaller units to ease and facilitate smaller transactions. The satoshi was named after the founder, or founders, of bitcoin, known as Satoshi Nakamoto.

with order book data, which contains

- the exchange timestamp;
- the instrument **symbol**, e.g. XBTUSD;
- the **bids**, $P_t^{b;1}$, $P_t^{b;2}$, $\ldots$, $P_t^{b;10}$ in USD per one BTC; $P_t^{b;1}$ being the best (largest) bid, $P_t^{b;2}$ the next best, etc.;
- the **asks**, $P_t^{a;1}$, $P_t^{a;2}$, $\ldots$, $P_t^{a;10}$ in USD per one BTC; $P_t^{a;1}$ being the best (smallest) ask, $P_t^{a;2}$ the next best, etc.;
- the **bid sizes**, $V_t^{b;1}$, $V_t^{b;2}$, $\ldots$, $V_t^{b;10}$ in USD;
- the **ask sizes**, $V_t^{a;1}$, $V_t^{a;2}$, $\ldots$, $V_t^{a;10}$ in USD.

The actual order book has more levels, but in this particular data feed only the top ten levels are reported.

Order book updates are more frequent than the trades. For our purpose it is sufficient to operate at the frequency of the trades rather than the order book updates, as trades can be thought of as 'maximum information events', which drive the evolution of the order book. Therefore we perform an **asof-join** [NBGD19, Chapter 7] of the order book updates on the trade times. Thus we use the **trade time** instead of the wall-clock time.

The backbone of our work is the linear regression analysis. We regress the difference between the future $\text{Midprice}_{t+\tau}$ and the current Midprice_t, where τ, measured in trade-ticks, is a certain time horizon. In our research, we take $\tau = 1, 2, \ldots, 100$.

We divide each trading day into two halves (by the number of trades). The first half is used as a training set, the second as a test set. We calibrate the linear regression model on the training set and assess its performance, by means of an **out-of-sample** R^2, on the test set.

We produce a plot of out-of-sample R^2 versus τ in order to assess the forecastability at different time horizons.

4.1.1 Midprice variance

We define the **midprice variance** with window size w at time t, MV_t^w, as the sample variance of the mids in the window $[t - w + 1, t]$.

4.1.2 Signed trade size variance

Similarly, we define the **signed trade size variance** with window size w at time t, $STSV_t^w$, as the sample variance of the trade sizes multiplied by 1 for Buys and -1 for sells.

4.2 Features

4.2.1 Bid–ask imbalance

The **bid–ask imbalance**[2] was defined in [LPS14] as

$$I_t^{ba} = \frac{V^{b;1} - V^{a;1}}{V^{b;1} + V^{a;1}}.$$

A positive (respectively, negative) bid–ask imbalance indicates an order book that is heavier on the bid (respectively, ask) side.

We can generalise this definition in two ways. First, we consider the **level bid–ask imbalances** at other levels than the top of book:

$$I_t^{ba;i} = \frac{V^{b;i} - V^{a;i}}{V^{b;i} + V^{a;i}}.$$

In this notation, I_t^{ba} becomes $I_t^{ba;1}$. The economic significance of this generalisation is somewhat less clear than that of the following generalisation—the **cumulative bid–ask imbalances**:

$$I_t^{ba;1,2,\ldots,i} = \frac{\sum_{k=1}^{i} V^{b;k} - \sum_{k=1}^{i} V^{a;k}}{\sum_{k=1}^{i} V^{b;k} + \sum_{k=1}^{i} V^{a;k}}.$$

We hope that the bid–ask imbalance calculated across multiple order book levels may contain more information as to the future price movements than just the top-of-book bid–ask imbalance, which in this notation becomes $I_t^{ba;1}$.

4.2.2 Bid And Ask Imbalance

We further generalise the bid–ask imbalance and compute the **bid imbalance**

$$I_t^{b;1,2,\ldots,i;i+1,i+2;\ldots;j} = \frac{\sum_{k=1}^{i} V^{b;k} - \sum_{k=i+1}^{j} V^{b;k}}{\sum_{k=1}^{j} V^{b;k}}$$

and the **ask imbalance**

$$I_t^{a;1,2,\ldots,i;i+1,i+2;\ldots;j} = \frac{\sum_{k=1}^{i} V^{a;k} - \sum_{k=i+1}^{j} V^{a;k}}{\sum_{k=1}^{j} V^{a;k}}.$$

These imbalances indicate whether the order book is top heavy on the bid (respectively, the ask) side, or on the contrary, most of the liquidity is contained in deeper levels of the order book.

4.2.3 Volume Order Imbalance

Volume order imbalance (VOI) was proposed in [She15] as a proxy to **order flow imbalance (OFI)** defined in [CKS14]. Shen [She15] defines the VOI as follows:

$$VOI_t = \delta V_t^{b;1} - \delta V_t^{a;1},$$

[2]In [She15] this is referred to as the **order imbalance ratio (OIR)**.

where

$$\delta V_t^{b;1} = \begin{cases} 0, & P_t^{b;1} < P_{t-1}^{b;1}, \\ V_t^{b;1} - V_{t-1}^{b;1}, & P_t^{b;1} = P_{t-1}^{b;1}, \\ V_t^{b;1}, & P_t^{b;1} > P_{t-1}^{b;1}; \end{cases} \qquad \delta V_t^{a;1} = \begin{cases} V_t^{a;1}, & P_t^{a;1} < P_{t-1}^{a;1}, \\ V_t^{a;1} - V_{t-1}^{a;1}, & P_t^{a;1} = P_{t-1}^{a;1}, \\ 0, & P_t^{a;1} > P_{t-1}^{a;1}. \end{cases}$$

If the current bid price is lower than the previous bid price, that implies that either the trader cancelled his buy limit order or an order was filled at $P_{t-1}^{b;1}$. As we do not have a more granular order or message book, we cannot be certain of the trader intent, hence we conservatively set $\delta V_t^{b;1} = 0$. If the current bid price is the same as the previous price, we take the difference of the bid volume to represent incremental buying pressure from the last period. Lastly, if the current bid price is greater than the previous price, we can interpret this as upward price momentum due to the trader's intent to buy at a higher price. Downward price momentum and sell pressure can be interpreted analogously from the current and previous ask prices.

By analogy with our generalisations for the bid–ask imbalance, we generalise the VOI to **level VOI** as follows,

$$VOI_t^i = \delta V_t^{b;i} - \delta V_t^{a;i},$$

where

$$\delta V_t^{b;i} = \begin{cases} 0, & P_t^{b;i} < P_{t-1}^{b;i}, \\ V_t^{b;i} - V_{t-1}^{b;i}, & P_t^{b;i} = P_{t-1}^{b;i}, \\ V_t^{b;i}, & P_t^{b;i} > P_{t-1}^{b;i}; \end{cases} \qquad \delta V_t^{a;i} = \begin{cases} V_t^{a;i}, & P_t^{a;i} < P_{t-1}^{a;i}, \\ V_t^{a;i} - V_{t-1}^{a;i}, & P_t^{a;i} = P_{t-1}^{a;i}, \\ 0, & P_t^{a;i} > P_{t-1}^{a;i}. \end{cases}$$

In addition, we define the **cumulative VOI** as

$$VOI_t^{1,2,\ldots,i} = \delta V_t^{b;1,2,\ldots,i} - \delta V_t^{a;1,2,\ldots,i},$$

where

$$\delta V_t^{b;1,2,\ldots,i} = \begin{cases} 0, & \sum_{k=1}^i P_t^{b;k} < \sum_{k=1}^i P_{t-1}^{b;k}, \\ \sum_{k=1}^i V_t^{b;k} - \sum_{k=1}^i V_{t-1}^{b;k}, & \sum_{k=1}^i P_t^{b;k} = \sum_{k=1}^i P_{t-1}^{b;k}, \\ \sum_{k=1}^i V_t^{b;k}, & \sum_{k=1}^i P_t^{b;k} > \sum_{k=1}^i P_{t-1}^{b;k}; \end{cases}$$

$$\delta V_t^{a;1,2,\ldots,i} = \begin{cases} \sum_{k=1}^i V_t^{a;k}, & \sum_{k=1}^i P_t^{a;k} < \sum_{k=1}^i P_{t-1}^{a;k}, \\ \sum_{k=1}^i V_t^{a;k} - \sum_{k=1}^i V_{t-1}^{a;k}, & \sum_{k=1}^i P_t^{a;k} = \sum_{k=1}^i P_{t-1}^{a;k}, \\ 0, & \sum_{k=1}^i P_t^{a;k} > \sum_{k=1}^i P_{t-1}^{a;k}. \end{cases}$$

4.2.4 Trade Flow Imbalance

We modify the definition of **trade flow imbalance (TFI)** from [Sil19] and define it, by analogy with the bid–ask imbalance, as

$$TFI_t^w = \frac{\sum_{k=0}^{w-1} \mathbb{1}_{S_{t-k}=\text{Buy}} \cdot V_{t-k} - \sum_{k=0}^{w-1} \mathbb{1}_{S_{t-k}=\text{Sell}} \cdot V_{t-k}}{\sum_{k=0}^{w-1} V_{t-k}},$$

where w is the window size, $\mathbb{1}$ is the indicator function, which takes on the value 0 if the condition (subscript) is false and 1 if that condition is true, S_t is the side of the trade at time t (Buy or Sell), and V_t is the size of that trade. This definition of trade flow imbalance, under the name trade imbalance, can be found in [Roy].

4.2.5 Corwin-Schultz Bid-ask Spread Estimator

The **Corwin-Schultz bid-ask spread(CSS)** estimator was developed in [Rip16] to measure the bid-ask spread of an asset class from daily high and low prices based on two assumptions. The first assumption is that the daily high prices are typically buyer initiated and low prices are to the contrary, seller initiated. Therefore, the ratio of high-to-low prices for each day reflects both the fundamental volatility of stocks and its bid-ask spread. The second assumption is that the high-to-low price ratio's volatility increases proportionately with the length of trading interval. The equation of Corwin-Schultz bid-ask spread estimator is presented as below.

$$
\begin{aligned}
S_t &= \tfrac{2(e^{\alpha_t}-1)}{1+e^{\alpha_t}} \\
\alpha_t &= \tfrac{\sqrt{2\beta_t}-\sqrt{\beta_t}}{3-2\sqrt{2}} - \sqrt{\tfrac{\gamma_t}{3-2\sqrt{2}}} \\
\beta_t &= E\left\{ \textstyle\sum_{j=0}^{1} \left[\ln\left(\tfrac{H^1_{t+j,t+j+1}}{L^1_{t+j,t+j+1}}\right)\right]^2 \right\} \\
\gamma_t &= E\left\{ \textstyle\sum_{j=0}^{1} \left[\ln\left(\tfrac{H^1_{t+j,t+j+2}}{L^1_{t+j,t+j+2}}\right)\right]^2 \right\}
\end{aligned}
$$

In our case, We apply the equation to Bitcoin (BTC) asset with some modification. S is the spread of BTC bid-ask prices, we denote $H^1_{t,t+T}$ and $L^1_{t,t+T}$ as the observed high and low of the mean of level-1 bid and ask size within time interval T, respectively.

α_t represents the difference between the adjustments of a single window-length and double window-length at time t. β_t represents the high and low bid-ask size average adjustments to single rolling window-length. γ_t represents a double rolling window-length of high and low bid-ask size average adjustments.

4.2.6 Relative Strength Index

Relative strength index(RSI) is a momentum indicator used in analysis which can measure the recent price change level and evaluate overbought or oversold conditions. RSI can be displayed as an oscillator moved between two extremes and read from 0 to 100.

$$RSI = 100 - \left[\frac{100}{1+\frac{G_t}{L_t}}\right]$$

where

$$
\begin{aligned}
G_t &= E\left[\sum_{l=t-w}^{t} \mathbb{1}_{\{R_l>0\}}\right], w = \{10, 20, \ldots, 50\} \\
L_t &= E\left[\sum_{l=t-w}^{t} \mathbb{1}_{\{R_l<0\}}\right], w = \{10, 20, \ldots, 50\}
\end{aligned}
$$

Both the average gain and the average loss are positive value. R_i could represent different type of features. One of features we used is **Volume-Weighted Average Price (VWAP)** for Ask price where VWAP is defined as

$$VWAP_t = \frac{\sum_{i=t-w}^{t} V_i * p_i}{\sum_{i=t-\omega}^{t} V_i}$$

4.2.7 Aggregate Impact Index

We consider all transactions that happen in the given time interval $[t, t+T)$, for some $T > 0$ and where t and T are expressed in either event-time or calendar-time. Throughout this process, we choose to work in market order time, whereby we advance t by 1 for each market order arrival. For each $n \in [t, t+T)$, let ϵ_n be the signal of the n^{th} market order and let v_n be the volume. The signed **order-flow imbalance** is:

$$\Delta V = \sum_{n \in [t,t+T)} \varepsilon_n v_n$$

If $\Delta V > 0$, then more bid volume arrives in the given interval than does ask volume. That is the reason price to rise.

Taking the idea from [JPB18], we define the **aggregate impact index (AI)** as the change in the average of bid and ask size (level-1), A_t^1, over the interval $[t, t+T)$, conditioned to a certain order-flow imbalance:

$$AI(\Delta V, T) := \mathbb{E}\left[A_{t+T}^1 - A_t^1 \mid \sum_{n \in [t,t+T)} \varepsilon_n v_n = \Delta V\right]$$

4.2.8 Return

Let the **midprice** be defined as

$$\text{Midprice}_t = \frac{P_t^{a;1} + P_t^{b;1}}{2}.$$

We define the return as

$$R_t^w = \ln(\text{Midprice}_t) - \ln(\text{Midprice}_{t-w+1}).$$

We seek to determine whether past returns influence the future price movements.

4.2.9 Midprice Variance

Midprice variance features are sample variances of the midprice computed over a number of rolling windows.

4.2.10 Signed Trade Size Variance

Signed trade size variance features are sample variances of the signed trade size (positive for buys, negative for sells) computed over a number of rolling windows.

4.3 Feature Enrichment

We also enrich all the feature above by adding the ln function, moving average(ma) and difference($diff$).

$$\begin{aligned} ln(\text{feature}) &= log_e(\text{feature}) \\ diff_t(n, \text{feature}) &= \text{feature}_{t+n} - \text{feature}_t \\ MA_t(n, \text{feature}) &= \frac{\sum_{t=i-n+1}^{t=i} \text{feature}_i}{n}, (i > n-1) \end{aligned}$$

where t is the t^{th} row for each column.

The whole enrichment algorithm is shown as below(Algorithm 2).

Algorithm 2: Feature Enrichment

```
All_Features = [];
Win_Size = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 50, 100, 300, 500]
for f in feature set do
    enrich f with the functions $diff(1)$, $ln(1)$ ;
    store enriched f into AllFeatures
end
for f in All_Features do
    for w in Win_Size do
        ma_w = apply moving average with w for f;
        store ma_w into AllFeatures
    end
end
```

4.4 Assessing The Features

4.4.1 Individual Feature Visualization

We use the out-of-sample R^2 metric to assess the quality of each group of features. To this end we fit multiple linear regression models—one per feature—with that feature as a regressor. We regress the difference between the future price at each time horizon (from 1 inclusive to 100 inclusive) on the feature. We then repeat this exercise but with all features from a given group as regressors in a single, 'combined', regression model.

We divide each trading day into two equal (in terms of the number of trade-ticks) parts. The first part serves as a training set, the second as a test set. We fit the linear regression models on the training set, and assess their performance (compute the out-of-sample R^2) on the test set.

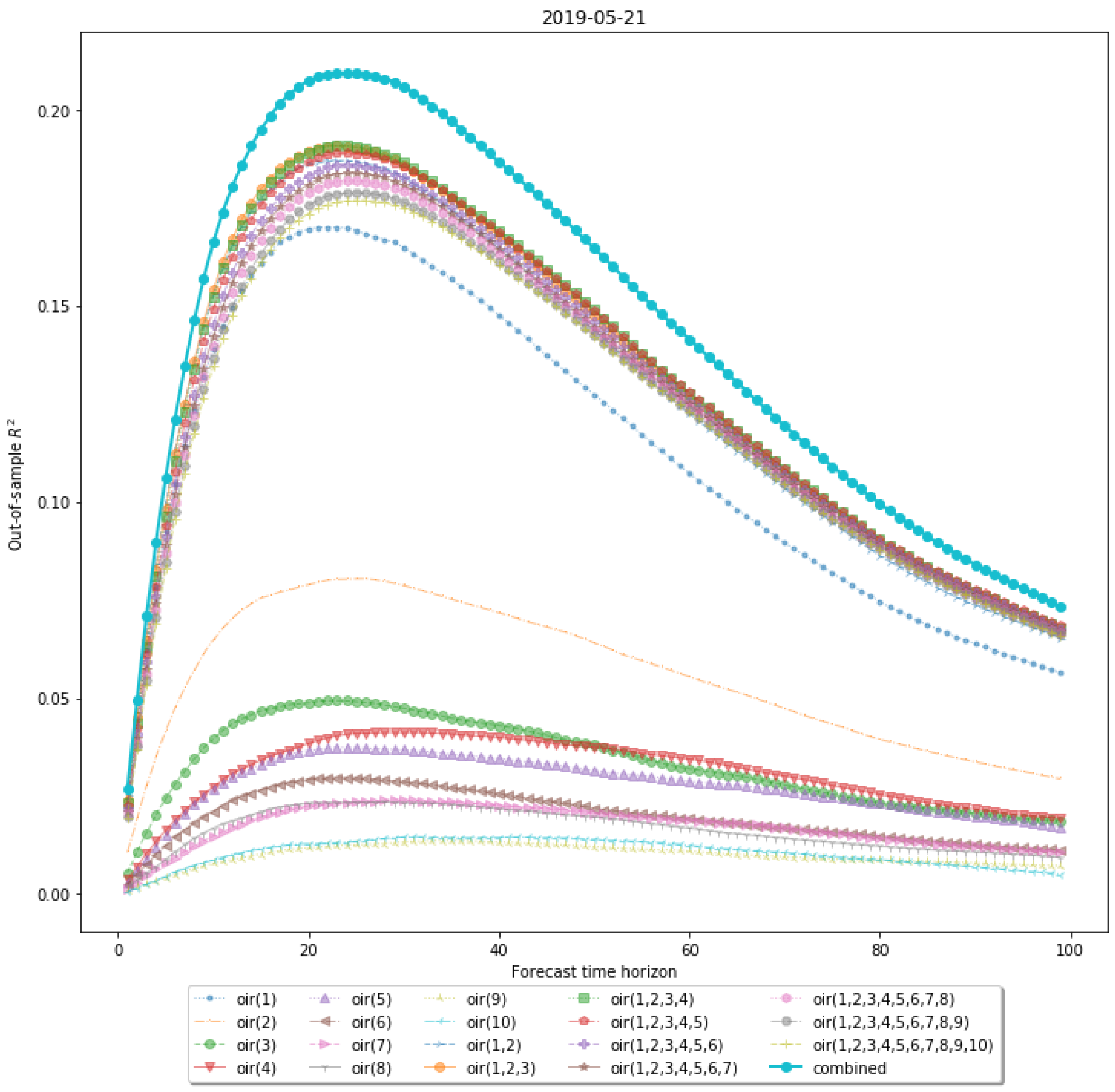


Figure 4.1: The performance of the OIR (bid–ask imbalance) features on 2019.05.21.

The results are presented on a plot of out-of-sample R^2 (on the y-axis) versus the forecast time horizon (on the x-axis) presented.

Figure 4.1 demonstrates the strong performance of the OIR (bid–ask imbalance) features. We notice that OIR has the best R^2 performance among all of the features. The highest out-of-sample R^2 could reach as high as 0.25 in between steps 20 and 30. OIR(1,2,3,4) has the best R^2 result for the individual OIR feature. The cumulative bid-ask imbalance has a better out of sample R^2 than single level bid-ask imbalance.

The bid and ask imbalance features (Figure 4.2) individually perform much worse than the bid–ask imbalance features, yielding the out-of-sample R^2 of less than 1%. The contribution for each out-of-sample R^2 is small, notably in between -0.01 to 0.01. However, when combined, they produce the out-of-sample R^2 of over 4.5%, which is quite significant in between stemps 15 to 30.

The combination of the VOI features (Figure 4.3) performs significantly worse than the combination of the bid and ask imbalance features. The VOIs with more combined order levels tend to

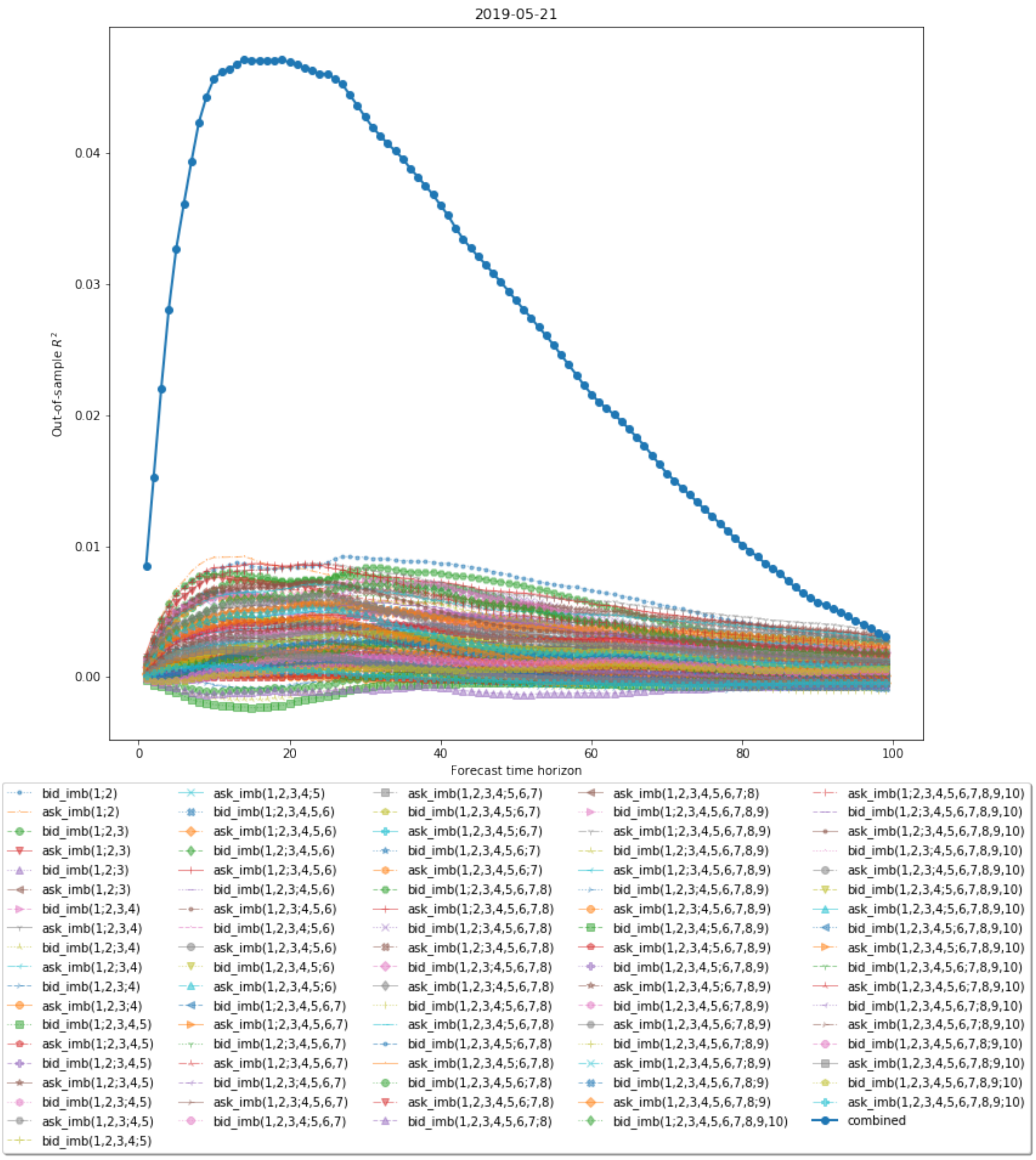


Figure 4.2: The performance of the bid and ask imbalance features on 2019.05.21.

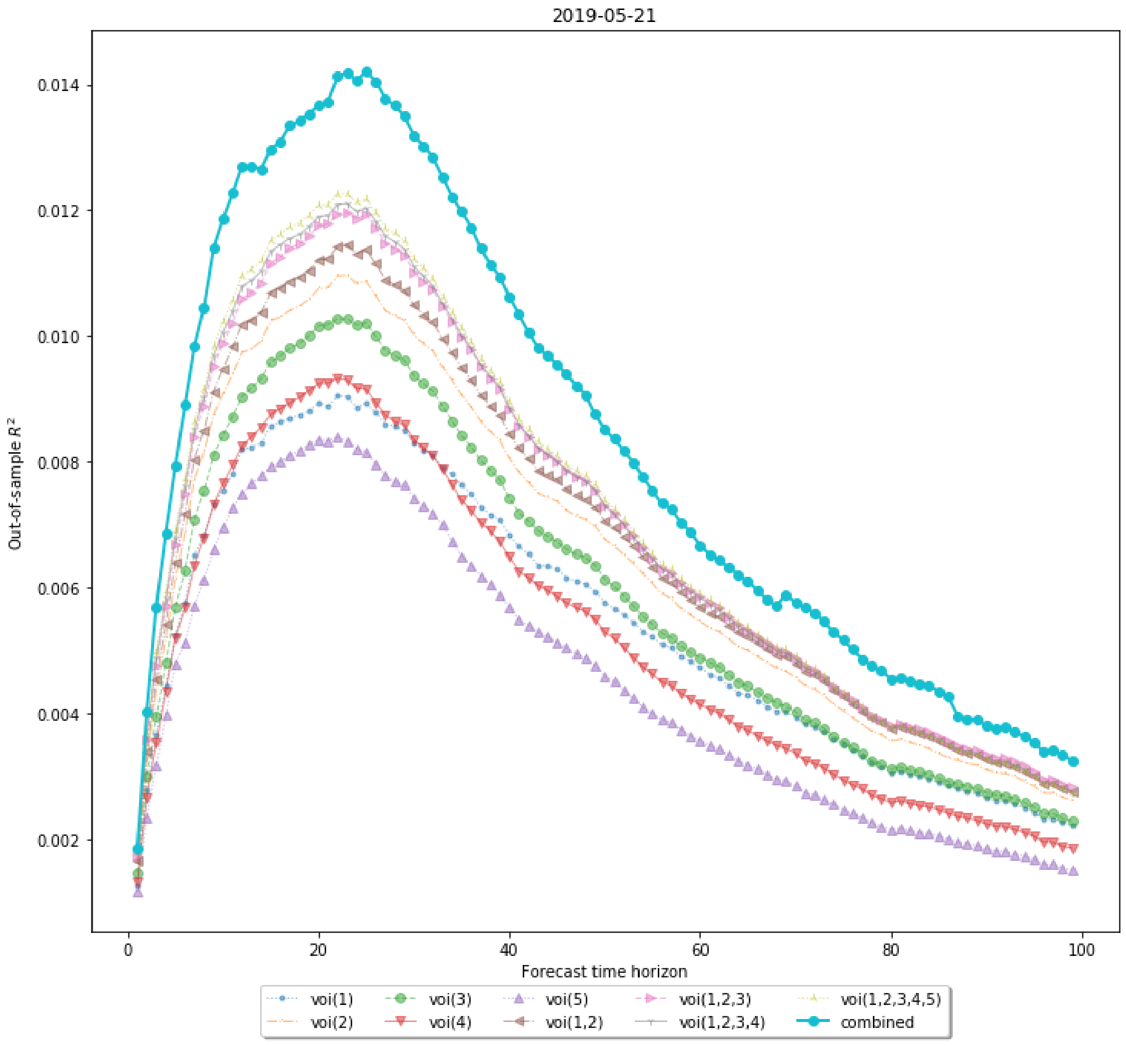


Figure 4.3: The performance of the VOI features on 2019.05.21.

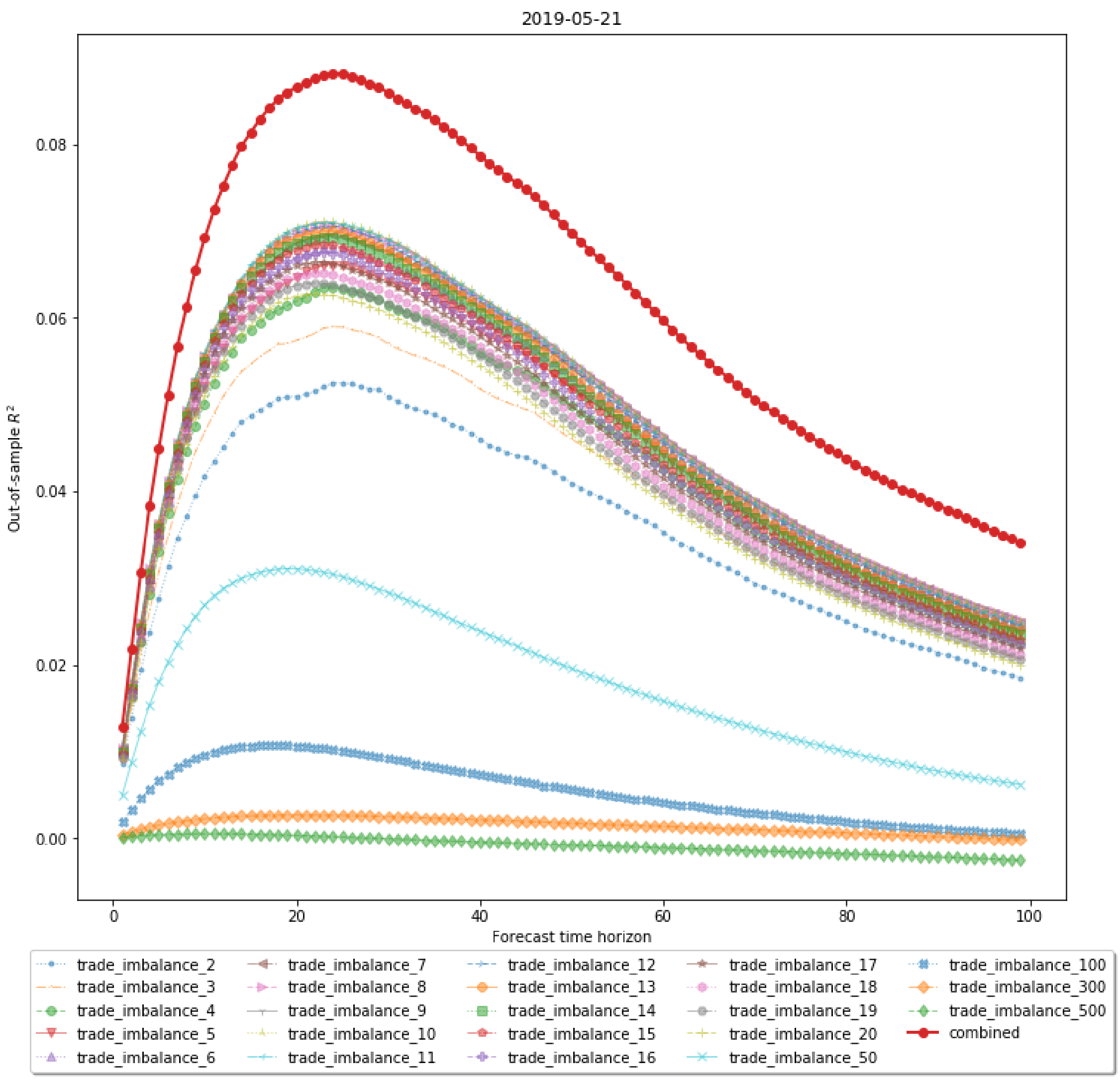


Figure 4.4: The performance of the trade flow imbalance features on 2019.05.21.

perform better than VOI with single order level. For instance, voi(1,2,3,4,5) is higher than voi(1), voi(2), voi(3), voi(4) and voi(5).

From the graph for the trade flow imbalance features (Figure 4.4) we can learn about the window size which yields the maximum information about the future price movement. The trade imbalance features demonstrate the best performance after the OIR. The overall out-of sample result for TFI looks promising. It has R^2 value exceeds 0.08 in between forecasting steps 18 to 40. For individual TFI features, TFI with rolling windows 11 and 12 perfrom the best. However the higher the rolling window length, the R^2 will be worse, i.e. TFI with window 500 has the worst predicting power which has 0 value.

Feature CSS (Figure 4.5) is one of the worst prediction features in our example. It shows negative out of sample results for prediction. It doesn't show a timely feedback for the results.

From the figure 4.6, we could notice that the R^2 of the combined RSI is slightly higher than 5% for prediction in between 15-20 steps. VWAP-Ask with rolling window of 20 steps performs the best among those RSI features. The R^2 of VWAP-Ask for all rolling windows are larger than zero,

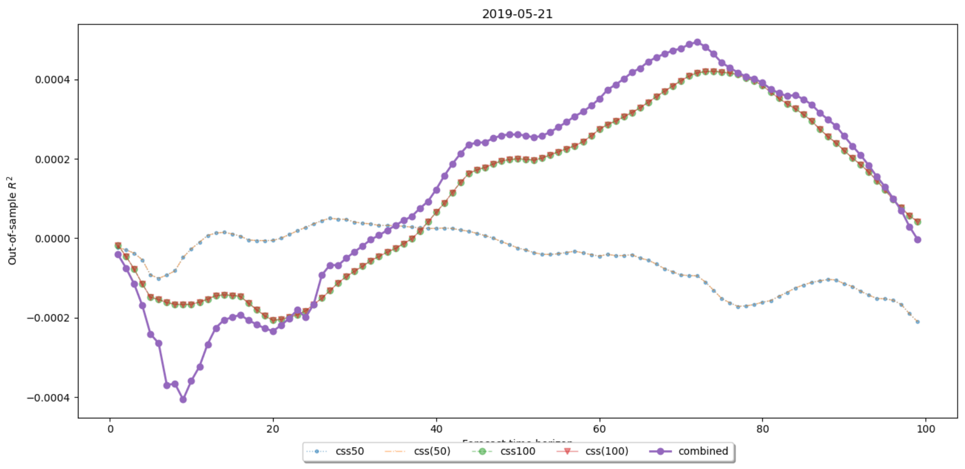


Figure 4.5: The performance of the CSS(Corwin-Schultz Bid-ask Spread) features on 2019.05.21.

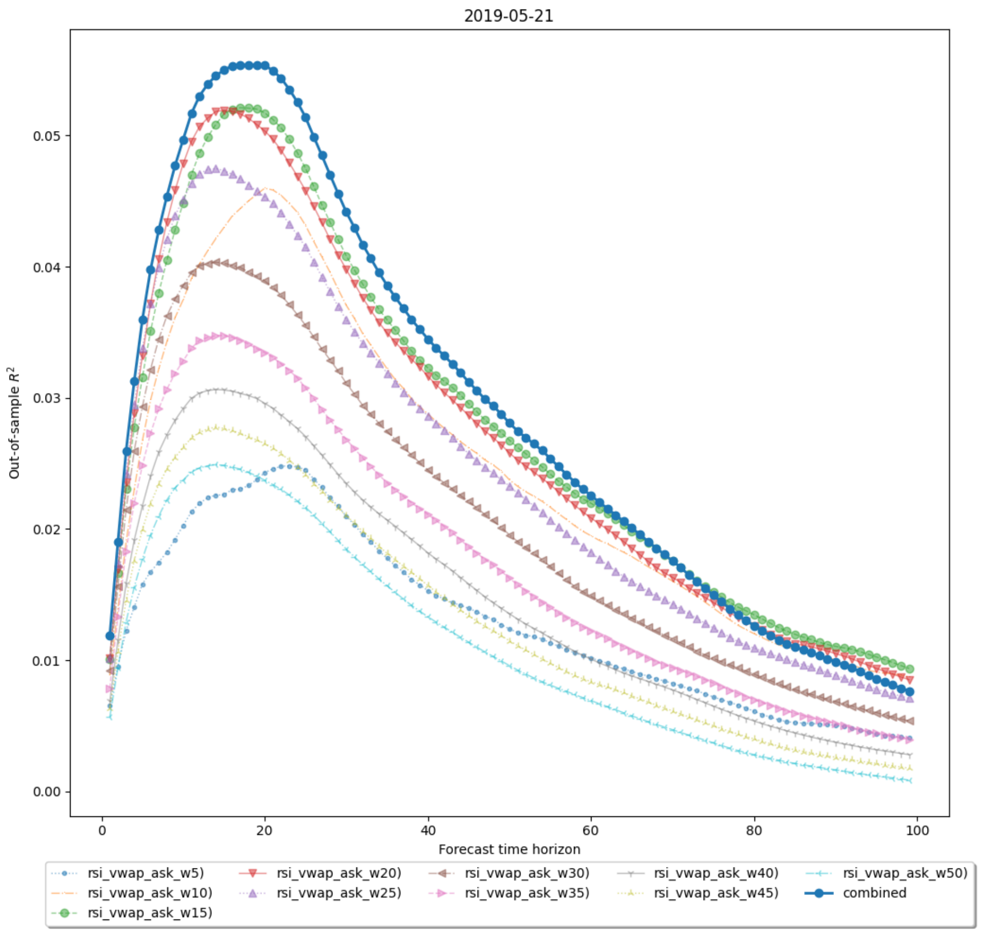


Figure 4.6: The performance of the RSI(relative strength index) features on 2019.05.21.

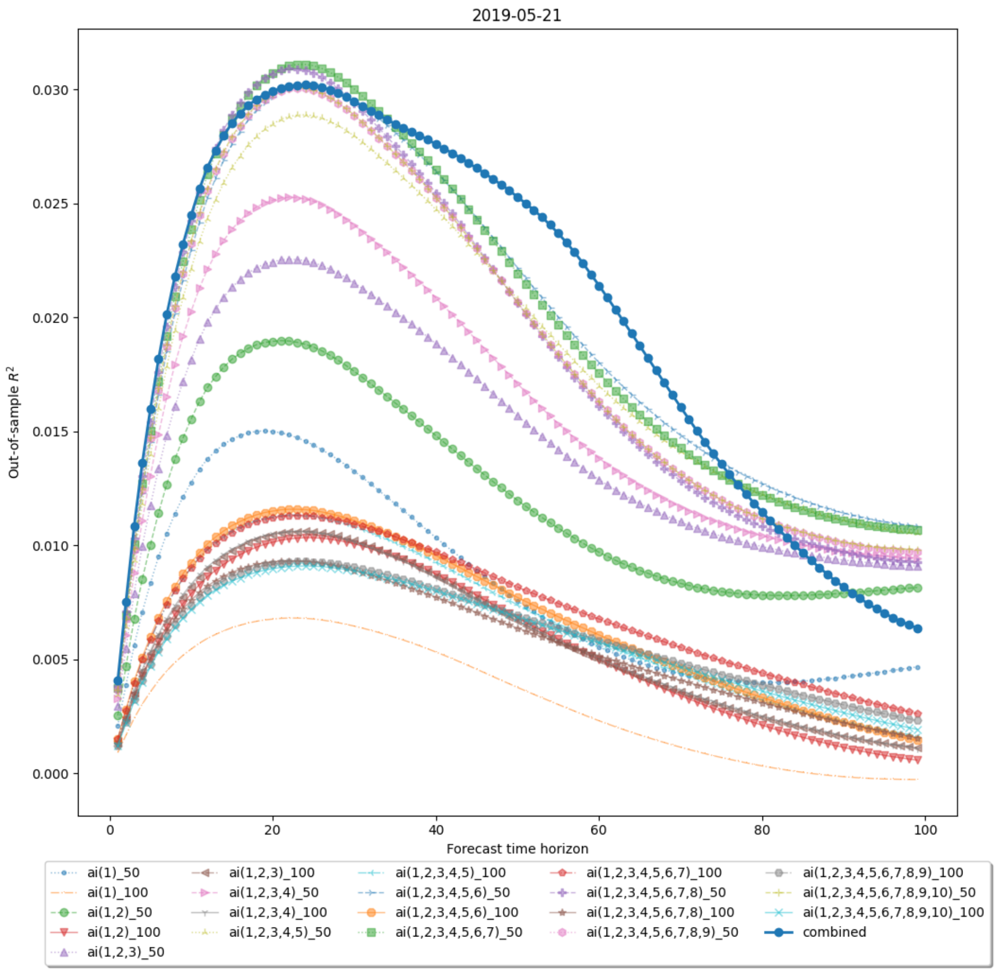


Figure 4.7: The performance of the AI(aggregate impact) features on 2019.05.21.

which indicates that RSI is a significant feature for prediction.

From the figure 4.7, we notice that the combined feature AI is slightly higher than 3% for the prediction in around 20, like the features above. Cumulative aggregate impact has a better result than individual aggregate impact. Note that the combined result does not perform the best since there are some negative results which haven't shown in this figure.

The return features (Figure 4.8) yield information about shorter-term price movements compared to the features that we have considered above—the peak is located further to the left. These features, when combined, produce an out-of-sample R^2 of just over 3%.Among individual return features, returns with rolling windows 14, 15, 16, 17, 18 perform the best, whereas the returns with very large rolling windows such as 100, 300, 500 perform the worst.

The result of combined Mid-Variance(Figure 4.9) reaches the highest point 0.0002 in between steps 40 to 50. However, all but one individual mid-variance features has negative R^2 which means

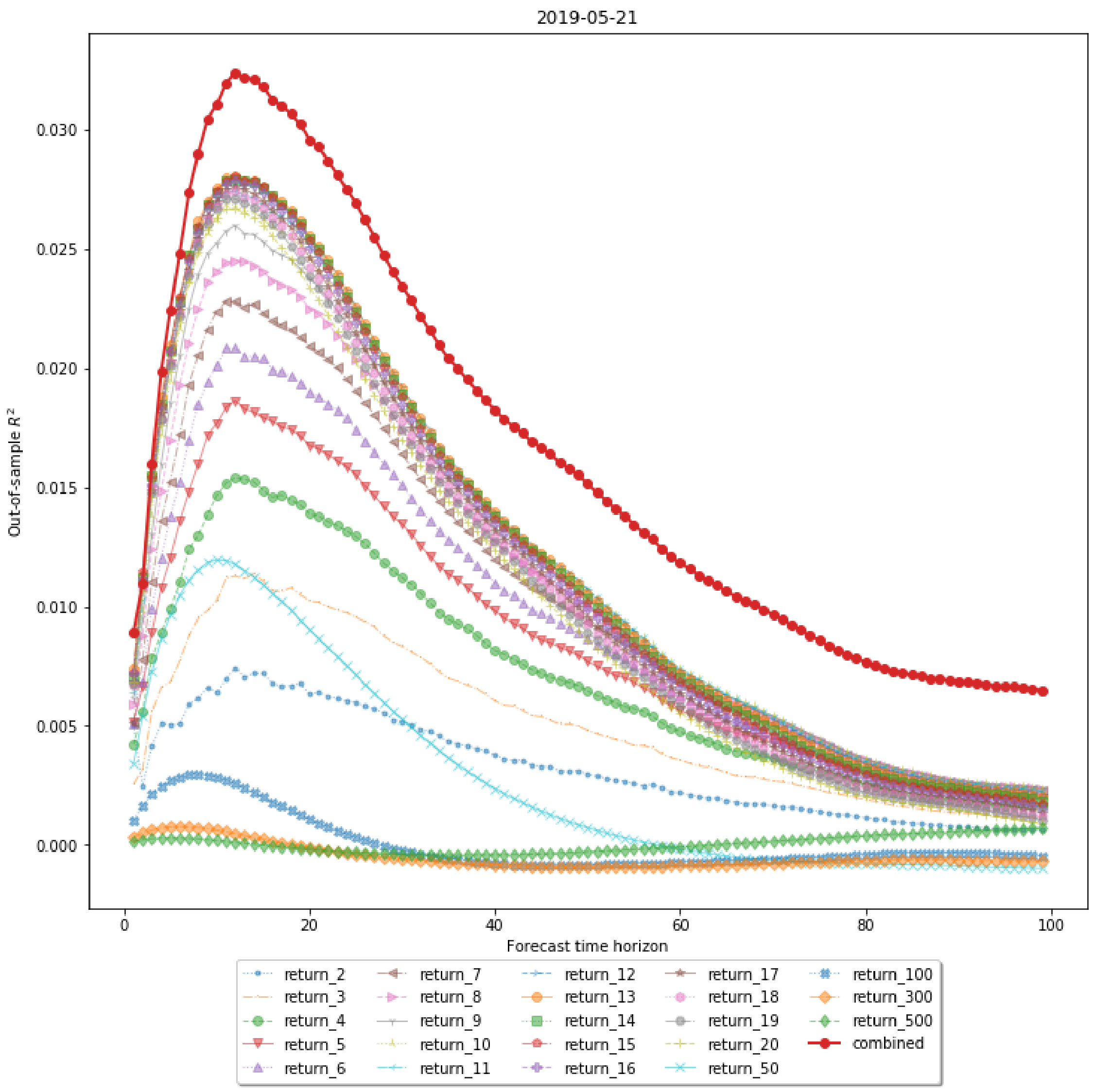


Figure 4.8: The performance of the return features on 2019.05.21.

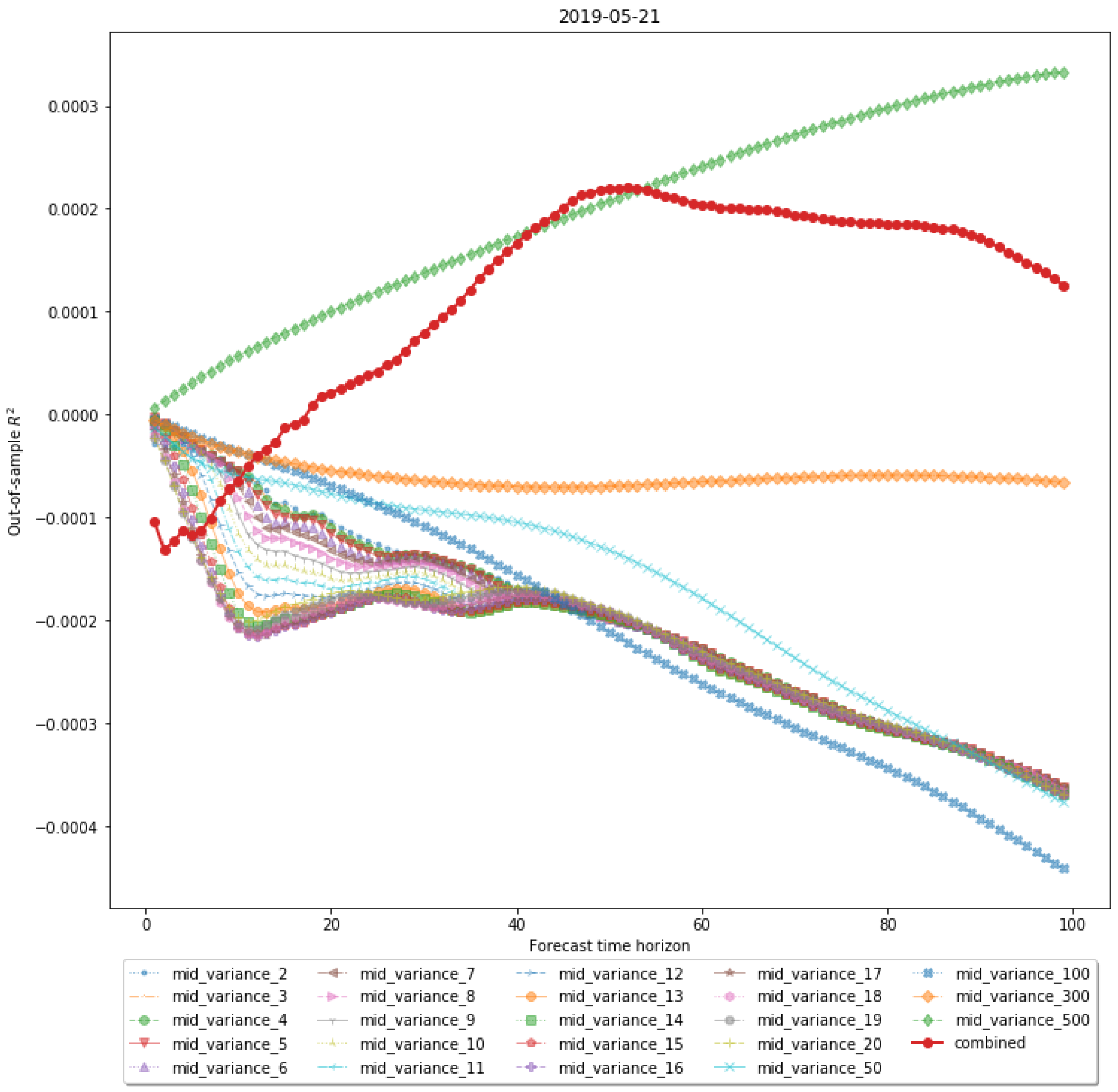


Figure 4.9: The performance of the midprice variance features on 2019.05.21.

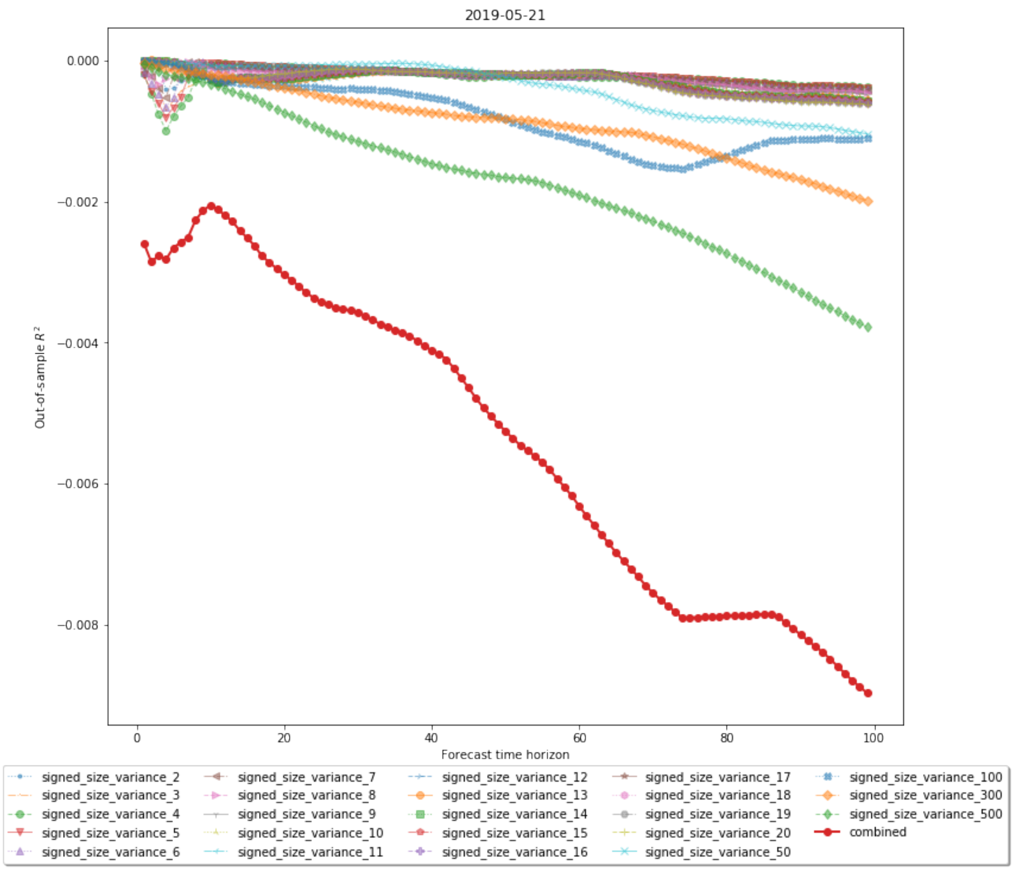


Figure 4.10: The performance of the signed trade size variance features on 2019.05.21.

that they are considered as insignificant features for the predictions. The only outlier is the mid-variance with window 14 which has increment R^2 for all predicting steps. As for the signed trade size variance, they are all below zero for out of sample predictions.

We see that neither the midprice variance features , nor the signed trade size variance features (Figure 4.10) add any value: the figures demonstrate that, as predictors of the future price moves, they amount to noise. However, we are not discounting them completely, as we hypothesize that other features may work differently (have different weights), depending on the volatility regime.

4.4.2 Combined Feature Visualization

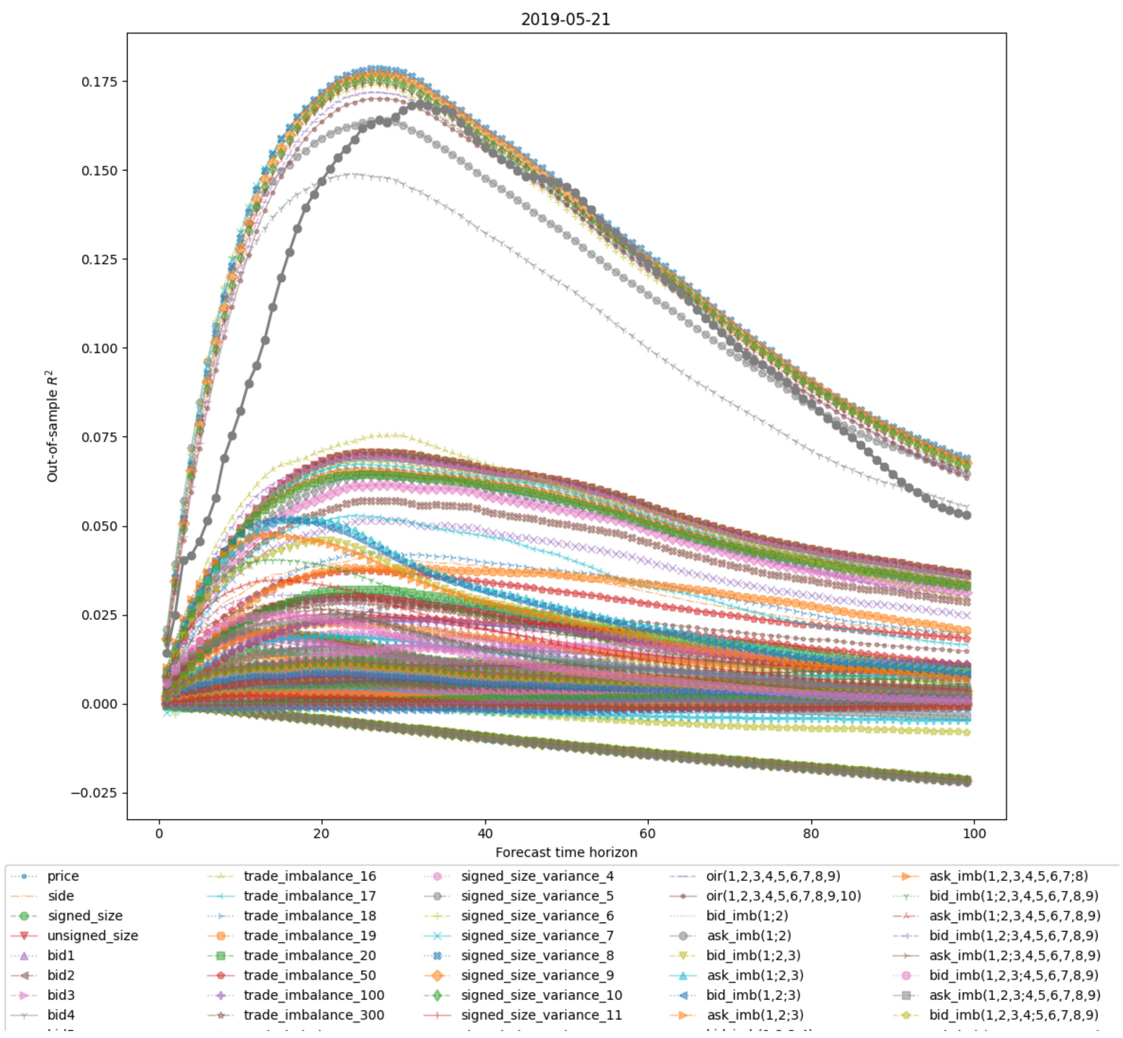


Figure 4.11: All features without moving average

We also print the out of sample results for all the features before moving average on figure 4.11. This figure shows impacts for each individual feature on the out of sample time series data. This picture describes in detail the general effect of each feature. It is shown that some of the features can achieve as good as 17.5% R^2. Most of the features can achieve positive results to regression and the predictive influence of each feature on future results increases first and then decreases, peaking at around 25 steps.

Chapter 5

Feature Selection

5.1 Big Data Storage And Computation

If we store all of the features which include the augmented features as shown in Chapter 4 into memory and perform the feature selection, we will have memory and performance issues for any system. To deal with the large data set in a more scalable way, we design a bespoke method to store the data into the file system.

The time-series data of these features are stored in Apache Parquet (.parquet) format. It is a standardized columnar-based storage, which allows compression schemes to be specified on a per-column level, and is future-proof to allow adding more encodings as they are invented and implemented. The reading and writing of the data from the parquet files are done by using Apache Arrow, which is an in-memory transport layer included in "PyArrow" Python package that can load the data from parquet into Pandas data frames. [PA-]

First, we decide to store the time-series data under all of the features excluding moving-average augmented features into a main parquet file named as "btc_features_with_ln_diff.parquet". For the MA augmented features, we name the file as "ma_feature X.parquet" conventionally and it will store all the feature X's moving average enrichment with 23 window sizes (i.e. MA(2, feature X), MA(3, feature X), ... , MA(300, feature X), MA(500, feature X) into that particular file. We use figure 5.1 as the example and we define the "feature X" as "ai(1)_50".

High performance computer (HPC) is a computer cluster consists of a set of connected computers that work together so that, they can be viewed as a single system so that one can harness the power of the CPUs, the memory and all the storage. We use the cluster to offload code execution from our laptop because the data needs too much memory and runs too long. What's more, clusters are particularly useful for executing parallel code so that when using the later bagging or boosting method, it is extremely useful for these parallel code execution.

Figure 5.2 shows the example for all moving average AI(Aggregate Impact) feature's parquet files.

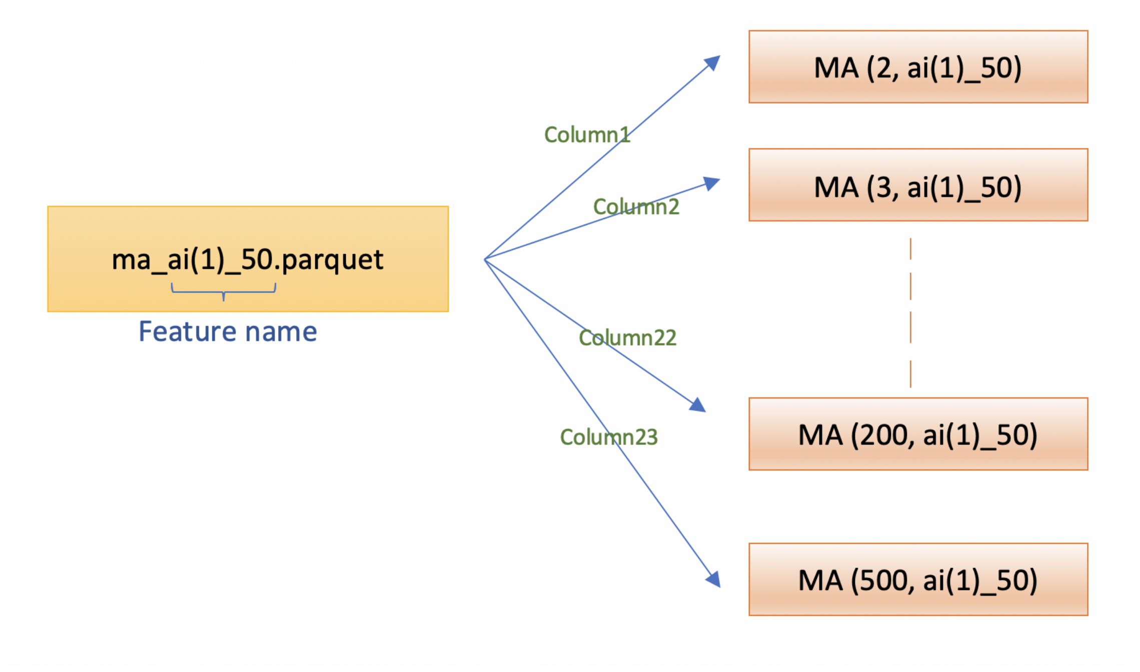
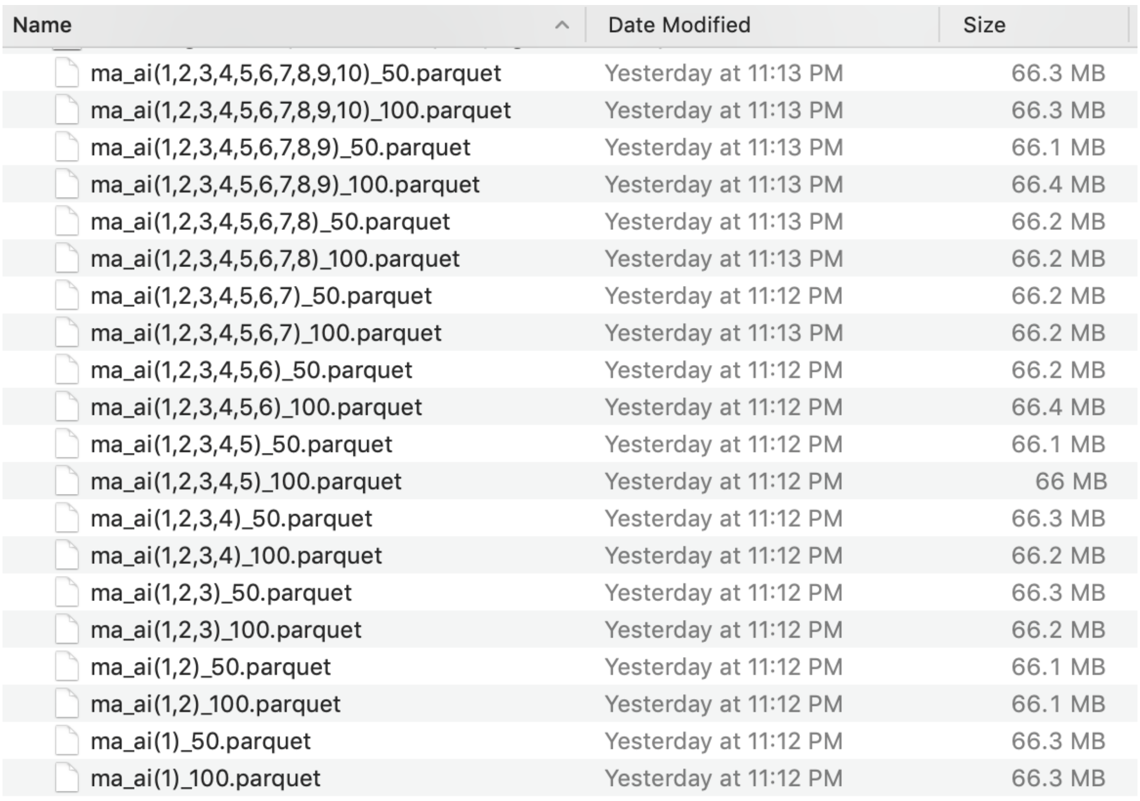


Figure 5.1: algorithm for parquet method



Name	Date Modified	Size
ma_ai(1,2,3,4,5,6,7,8,9,10)_50.parquet	Yesterday at 11:13 PM	66.3 MB
ma_ai(1,2,3,4,5,6,7,8,9,10)_100.parquet	Yesterday at 11:13 PM	66.3 MB
ma_ai(1,2,3,4,5,6,7,8,9)_50.parquet	Yesterday at 11:13 PM	66.1 MB
ma_ai(1,2,3,4,5,6,7,8,9)_100.parquet	Yesterday at 11:13 PM	66.4 MB
ma_ai(1,2,3,4,5,6,7,8)_50.parquet	Yesterday at 11:13 PM	66.2 MB
ma_ai(1,2,3,4,5,6,7,8)_100.parquet	Yesterday at 11:13 PM	66.2 MB
ma_ai(1,2,3,4,5,6,7)_50.parquet	Yesterday at 11:12 PM	66.2 MB
ma_ai(1,2,3,4,5,6,7)_100.parquet	Yesterday at 11:13 PM	66.2 MB
ma_ai(1,2,3,4,5,6)_50.parquet	Yesterday at 11:12 PM	66.2 MB
ma_ai(1,2,3,4,5,6)_100.parquet	Yesterday at 11:12 PM	66.4 MB
ma_ai(1,2,3,4,5)_50.parquet	Yesterday at 11:12 PM	66.1 MB
ma_ai(1,2,3,4,5)_100.parquet	Yesterday at 11:12 PM	66 MB
ma_ai(1,2,3,4)_50.parquet	Yesterday at 11:12 PM	66.3 MB
ma_ai(1,2,3,4)_100.parquet	Yesterday at 11:12 PM	66.2 MB
ma_ai(1,2,3)_50.parquet	Yesterday at 11:12 PM	66.3 MB
ma_ai(1,2,3)_100.parquet	Yesterday at 11:12 PM	66.2 MB
ma_ai(1,2)_50.parquet	Yesterday at 11:12 PM	66.1 MB
ma_ai(1,2)_100.parquet	Yesterday at 11:12 PM	66.1 MB
ma_ai(1)_50.parquet	Yesterday at 11:12 PM	66.3 MB
ma_ai(1)_100.parquet	Yesterday at 11:12 PM	66.3 MB

Figure 5.2: Parquet example

5.2 Methodology

To reduce the data complexity and dimensions for the model, we need to eliminate unnecessary variables. Feature selection is a complicated task in data mining. To prevent over-fitting, we evaluate combination of the features by using training and validation set. First, we select a feature and train it with the training set. Next, we use the validation feature set as input for the trained model. Then we compare the predicted result with validation target to calculate the metric scores. The feature with the best score will be selected. In next step the selected feature will combine with the next feature for training and repeat the process again. The feature selection process will continue until all of the features are considered for selection. The summary of the algorithm is shown as below (Algorithm 3).

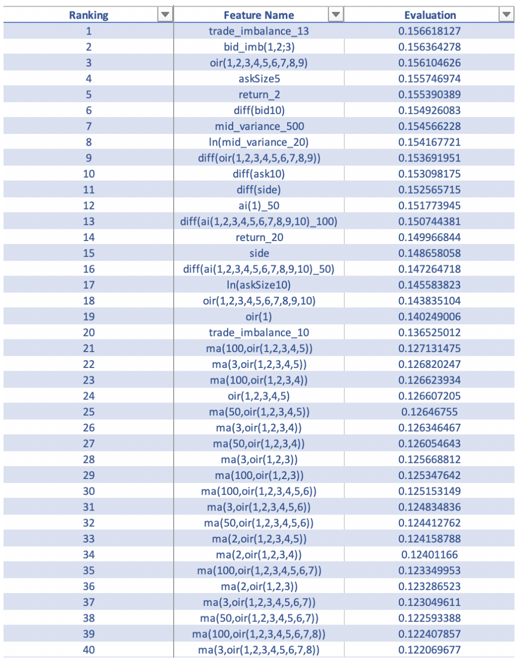
Algorithm 3: Feature Selection

```
included_col = [];
included_col = features ;
selected_col = [];
prev_best_metric = 0;
best_metric = 0;
best_metric_col = [];
while length of selected_col < length of included_col do
    for each c in included_col do
        if c is already selected then
            continue;
        else
            current_col = [];
            Append current_col with selected_col;
            Add c into current_col;
            metrics = [];
            for each ts, vs in (data_training_set, validation_set) do
                X_train = feature inputs from ts;
                Y_train = feature output from ts;
                Train X_train to fit Y_train with model;
                X_validation = feature inputs from vs;
                Y_validation = feature output from vs;
                Predict y_pred using x_validation with the trained model;
                Calculate the metrix of the y_pred by comparing with y_validation;
                Add metric score into metrics;
            end
            mean_metric = average of metric scores;
            update the best_metric value and the feature with the best metric score;
        end
        if best_metric − prev_best_metric > metric_improvement_threshold then
            add best_metric_col into selected_col
        else
            prev_best_metric = best_metric
        end
    end
end
Result: selected_col
```

5.3 Results For Feature Selection

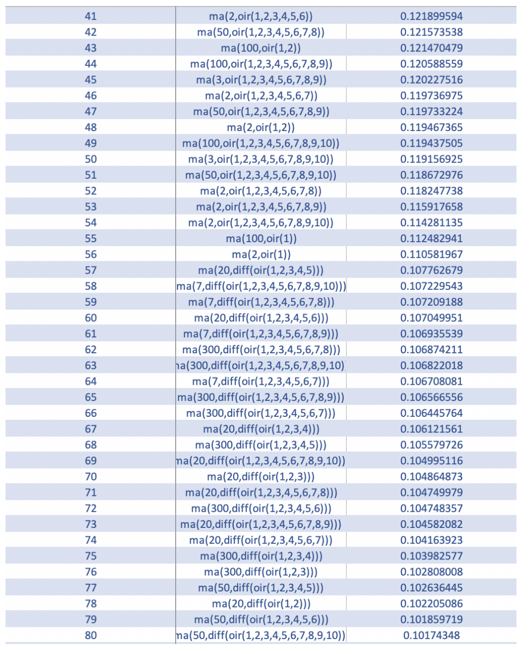
Our feature selection algorithm runs on Imperial College's general purpose HPC, which uses 24 CPU cores and 124 GB of memory.

All 2063 features are ranked and the top 80 features are shown in Figure 5.3. The first column is the ranking, which shows the importance of each feature. The second column is the feature name, and the third column is the metric score of the feature, which is the out of sample R^2 value when using linear regression.



Ranking	Feature Name	Evaluation
1	trade_imbalance_13	0.156618127
2	bid_imb(1,2;3)	0.156364278
3	oir(1,2,3,4,5,6,7,8,9)	0.156104626
4	askSize5	0.155746974
5	return_2	0.155390389
6	diff(bid10)	0.154926083
7	mid_variance_500	0.154566228
8	ln(mid_variance_20)	0.154167721
9	diff(oir(1,2,3,4,5,6,7,8,9))	0.153691951
10	diff(ask10)	0.153098175
11	diff(side)	0.152565715
12	ai(1)_50	0.151773945
13	diff(ai(1,2,3,4,5,6,7,8,9,10)_100)	0.150744381
14	return_20	0.149966844
15	side	0.148658058
16	diff(ai(1,2,3,4,5,6,7,8,9,10)_50)	0.147264718
17	ln(askSize10)	0.145583823
18	oir(1,2,3,4,5,6,7,8,9,10)	0.143835104
19	oir(1)	0.140249006
20	trade_imbalance_10	0.136525012
21	ma(100,oir(1,2,3,4,5))	0.127131475
22	ma(3,oir(1,2,3,4,5))	0.126820247
23	ma(100,oir(1,2,3,4))	0.126623934
24	oir(1,2,3,4,5)	0.126607205
25	ma(50,oir(1,2,3,4,5))	0.12646755
26	ma(3,oir(1,2,3,4))	0.126346467
27	ma(50,oir(1,2,3,4))	0.126054643
28	ma(3,oir(1,2,3))	0.125668812
29	ma(100,oir(1,2,3))	0.125347642
30	ma(100,oir(1,2,3,4,5,6))	0.125153149
31	ma(3,oir(1,2,3,4,5,6))	0.124834836
32	ma(50,oir(1,2,3,4,5,6))	0.124412762
33	ma(2,oir(1,2,3,4,5))	0.124158788
34	ma(2,oir(1,2,3,4))	0.12401166
35	ma(100,oir(1,2,3,4,5,6,7))	0.123349953
36	ma(2,oir(1,2,3))	0.123286523
37	ma(3,oir(1,2,3,4,5,6,7))	0.123049611
38	ma(50,oir(1,2,3,4,5,6,7))	0.122593388
39	ma(100,oir(1,2,3,4,5,6,7,8))	0.122407857
40	ma(3,oir(1,2,3,4,5,6,7,8))	0.122069677

(a) feature ranking 1-40



41	ma(2,oir(1,2,3,4,5,6))	0.121899594
42	ma(50,oir(1,2,3,4,5,6,7,8))	0.121573538
43	ma(100,oir(1,2))	0.121470479
44	ma(100,oir(1,2,3,4,5,6,7,8,9))	0.120588559
45	ma(3,oir(1,2,3,4,5,6,7,8,9))	0.120227516
46	ma(2,oir(1,2,3,4,5,6,7))	0.119736975
47	ma(50,oir(1,2,3,4,5,6,7,8,9))	0.119733224
48	ma(2,oir(1,2))	0.119467365
49	ma(100,oir(1,2,3,4,5,6,7,8,9,10))	0.119437505
50	ma(3,oir(1,2,3,4,5,6,7,8,9,10))	0.119156925
51	ma(50,oir(1,2,3,4,5,6,7,8,9,10))	0.118672976
52	ma(2,oir(1,2,3,4,5,6,7,8))	0.118247738
53	ma(2,oir(1,2,3,4,5,6,7,8,9))	0.115917658
54	ma(2,oir(1,2,3,4,5,6,7,8,9,10))	0.114281135
55	ma(100,oir(1))	0.112482941
56	ma(2,oir(1))	0.110581967
57	ma(20,diff(oir(1,2,3,4,5)))	0.107762679
58	ma(7,diff(oir(1,2,3,4,5,6,7,8,9,10)))	0.107229543
59	ma(7,diff(oir(1,2,3,4,5,6,7,8)))	0.107209188
60	ma(20,diff(oir(1,2,3,4,5,6)))	0.107049951
61	ma(7,diff(oir(1,2,3,4,5,6,7,8,9)))	0.106935539
62	ma(300,diff(oir(1,2,3,4,5,6,7,8)))	0.106874211
63	a(300,diff(oir(1,2,3,4,5,6,7,8,9,10)	0.106822018
64	ma(7,diff(oir(1,2,3,4,5,6,7)))	0.106708081
65	ma(300,diff(oir(1,2,3,4,5,6,7,8,9)))	0.106566556
66	ma(300,diff(oir(1,2,3,4,5,6,7)))	0.106445764
67	ma(20,diff(oir(1,2,3,4)))	0.106121561
68	ma(300,diff(oir(1,2,3,4,5)))	0.105579726
69	a(20,diff(oir(1,2,3,4,5,6,7,8,9,10))	0.104995116
70	ma(20,diff(oir(1,2,3)))	0.104864873
71	ma(20,diff(oir(1,2,3,4,5,6,7,8)))	0.104749979
72	ma(300,diff(oir(1,2,3,4,5,6)))	0.104748357
73	ma(20,diff(oir(1,2,3,4,5,6,7,8,9)))	0.104582082
74	ma(20,diff(oir(1,2,3,4,5,6,7)))	0.104163923
75	ma(300,diff(oir(1,2,3,4)))	0.103982577
76	ma(300,diff(oir(1,2,3)))	0.102808008
77	ma(50,diff(oir(1,2,3,4,5)))	0.102636445
78	ma(20,diff(oir(1,2)))	0.102205086
79	ma(50,diff(oir(1,2,3,4,5,6)))	0.101859719
80	a(50,diff(oir(1,2,3,4,5,6,7,8,9,10))	0.10174348

(b) feature ranking 41-80

Figure 5.3: selections of feature rankings

Chapter 6

Forecasting

6.1 Methodology

To avoid any bias with the training data set, we use new daily test set of BTC time series for forecasting. We choose the daily BTC tick data on 01-06-2020. From the feature ranking we have defined before, we choose to use top 30 ranked features as the input set and mid price as the target. Before we perform the prediction of mid price on next 100 days, parameter search optimisation techniques are needed to find the best parameters for the ensemble methods like Random Forest and Gradient Boosting Tree.

Since the parameter space for the Random Forest (RF) are all integers and not too large, grid search is used to search for its best parameters. Grid search is a traditional and popular method of hyper-parameters optimization which makes a full sweep over a given subset of the hyper-parameters space. Grid search is a method that can be easily paralleled since the hyper-parameter values which the algorithm works with are usually independent with each other [PL19].

For Gradient Boosting Tree (GBT), some of its parameters are floating numbers. To perform more accurate global optimisation for the model, Differential Evolution (DE) is used. Since the machine learning algorithms for both RF and GBT will include spaces with unlimited values for some parameters, it is possible that we need to specify a boundary to apply a grid search.

During our initial run for optimisation using DE in HPC, the optimisation job has insufficient memory issue eventhough it is assigned with 124GB of memory. Therefore we aggregate the tick data into N number of bins to reduce the storage size of the data. With the compressed data, we run the job again with 24 CPUs and 124GB. We managed to get the optimised parameters after the job has run for 9 hours.

The forecasting process is shown as figure 6.1.

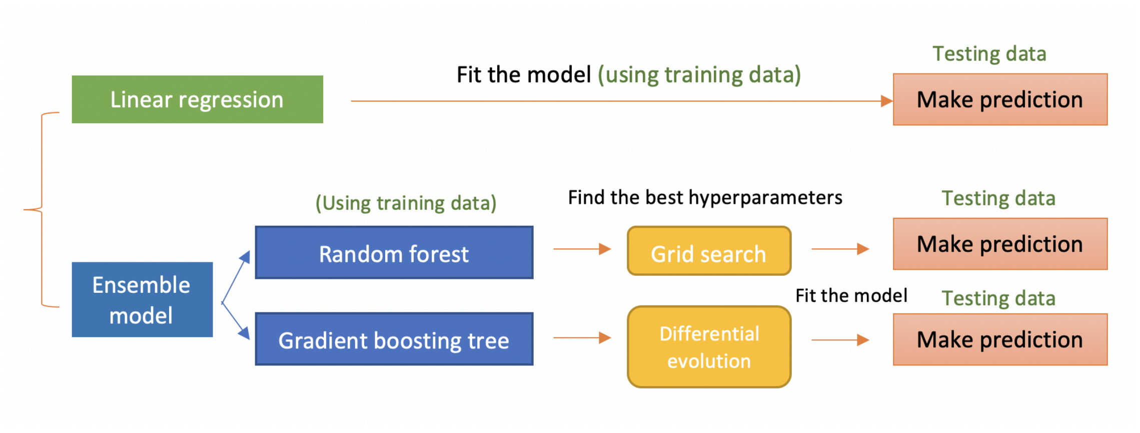


Figure 6.1: Forecasting process

6.2 Forecasting With Linear Regression

The figure 6.2 shows the forecasting results of the top 30 selected features. The results are indicated by out-of-sample R^2. The R^2 value is approximately greater than 23%, which shows a significant result. This indicates that these top 30 features are useful to be used to forecast the future prices from the historical data.

Among these features, the most significant dependent variable is OIR(1,2,3,4,5) which has R^2 value in between 15% and 17%. This result coincides with the OIR feature as shown in (Figure 4.1), which suggests that the OIR is the best feature for prediction.

6.3 Forecasting With Random Forest

We use the RF model provided by Sci-Kit Learn Python package to predict the BTC prices. With top 30 ranked features as the input, we divide the data into training set, validation set and test set with the ratio of 0.5, 0.3 and 0.2 respectively.

6.3.1 Grid Search Parameters

In our experiment, we use the grid search algorithm provided by Sci-Kit-Learn package to find the best hyper-parameters for RF. The parameter space we use is shown in table 6.1.

Hyperparameter	**Grid for Random Forest**
n_estimators	[50,60,70, ..., 500]
max_depth	[10,20,30, ..., 110]
min_samples_split	[2,3,4, ..., 50]
min_samples_leaf	[1,2,3, ..., 20]

Table 6.1: grid search hyperparameters space for random forest

The optimisation process runs on Imperial College's HPC and it took around 621.9 minutes. The most optimised parameters for RF are shown as table 6.2.

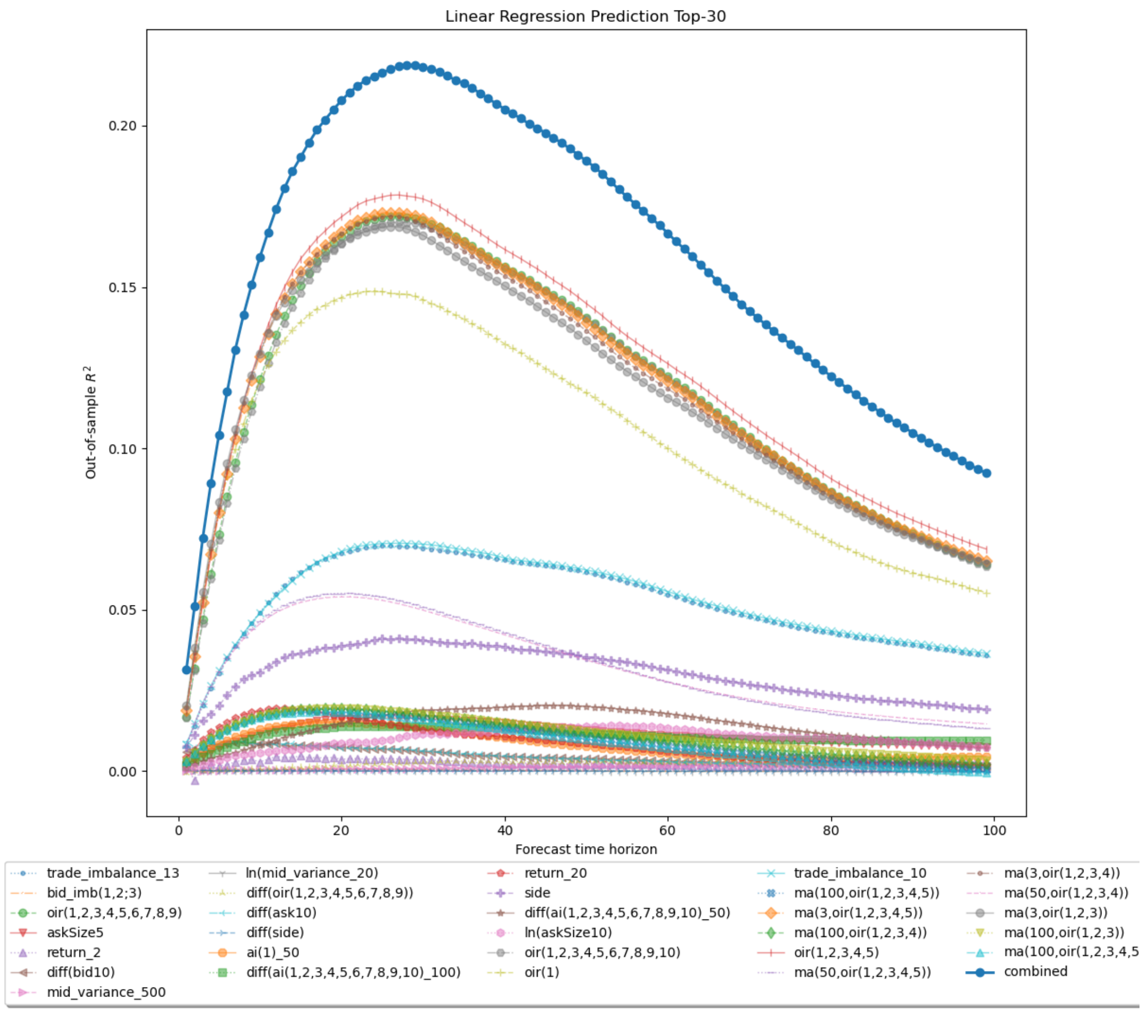


Figure 6.2: Linear regression prediction top30

Hyperparameter	Grid for Random Forest
n_estimators	400
max_depth	70
min_samples_split	2
min_samples_leaf	1

Table 6.2: grid search result for random forest

6.3.2 Result

Using the features we selected above, we print the all features when fitting the random forest model with the best parameters.

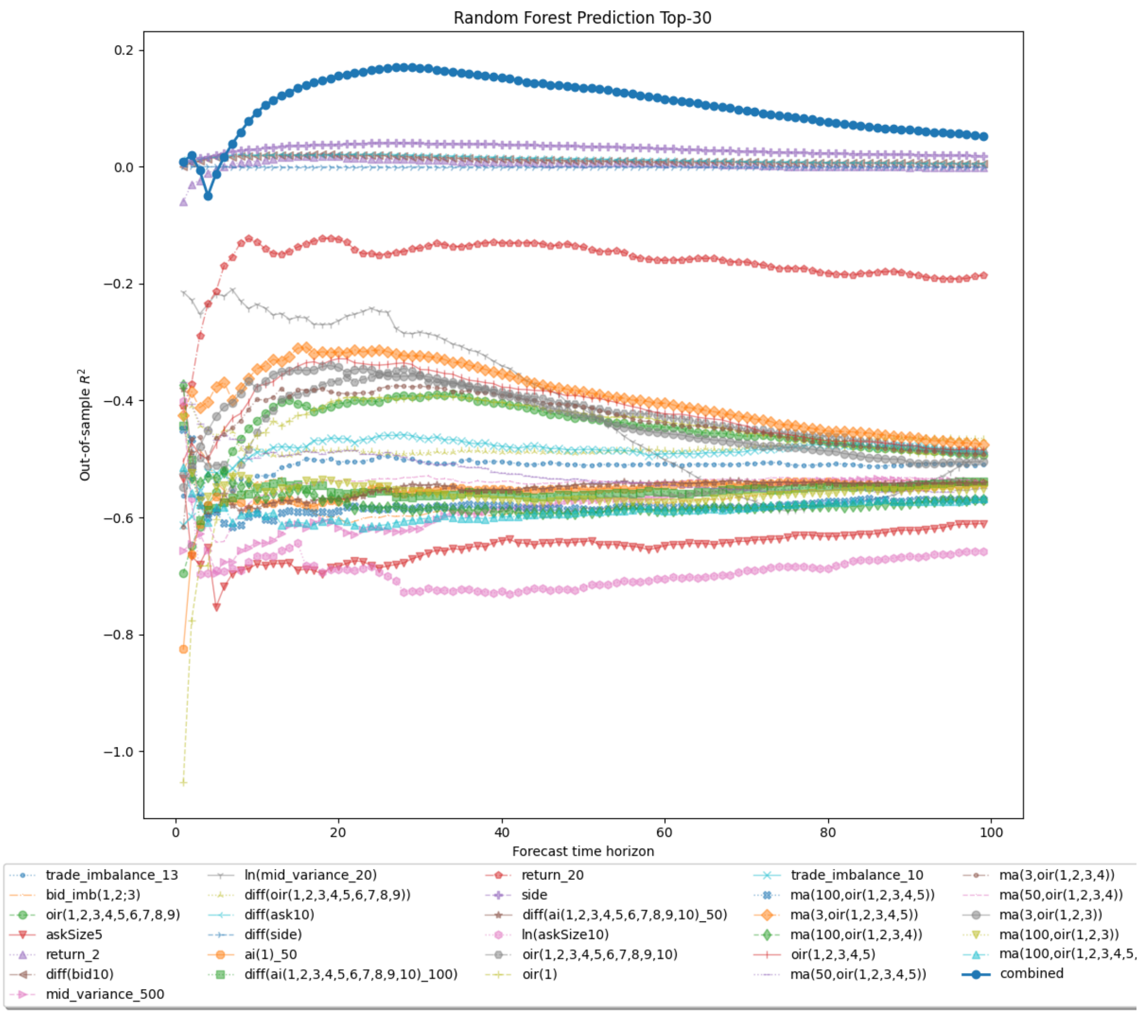


Figure 6.3: random forest top 30 forecasting

From the Figure 6.3, it shows that the combined result of all 30 top features has the maximum out-of-sample R^2 in between steps 20 and 40. In overall, the performance of prediction by using RF is not as good as LR. Majority of the features which are able to make a great contribution for LR now have the R^2 values in negative region. The poorer performance for RF prediction could be explained below.

1. The features are selected by using linear regression. Therefore, it captures the linear relationship between the input features and the BTC mid-prices. The nonlinear relationship could not be well represented when selecting these features.

2. The selected features by LR are more bias to LR model. This could explain why these top features ranked by LR do not perform well with RF prediction.

6.4 Forecasting With Gradient Boosting Tree

6.4.1 Differential Evolution Parameters

We use the DE provided by the Scipy optimisation package. For its parameters, we set maxiter=100, popsize=16, updating='deferred', workers=12.

For the parameters of the Gradient Boosting class, we set the parameters objective='reg: squared error' and learning_rate=0.1.

The boundary for the gradient boosting tree parameters are set as below table 6.3

Hyperparameter	**Range for Gradient boosting**
col_sample_by_tree	(0.0, 1.0)
alpha	(0.0, 1.0)
max_depth	(2, 30)
n_estimators	(50, 200)

Table 6.3: differential evolution hyperparameter range for gradient boosting tree

The differential evolution gives us the parameter results as below table 6.4

Hyperparameter	**Differential evolution result**
col sample by tree	0.05
alpha	0.003
max depth	29
n estimators	50

Table 6.4: differential evolution parameter for gradient boosting tree

6.4.2 Result

After using the hyper-parameters generated by DE optimisation algorithm for GB, we obtain the forecast result as shown in figure 6.4.

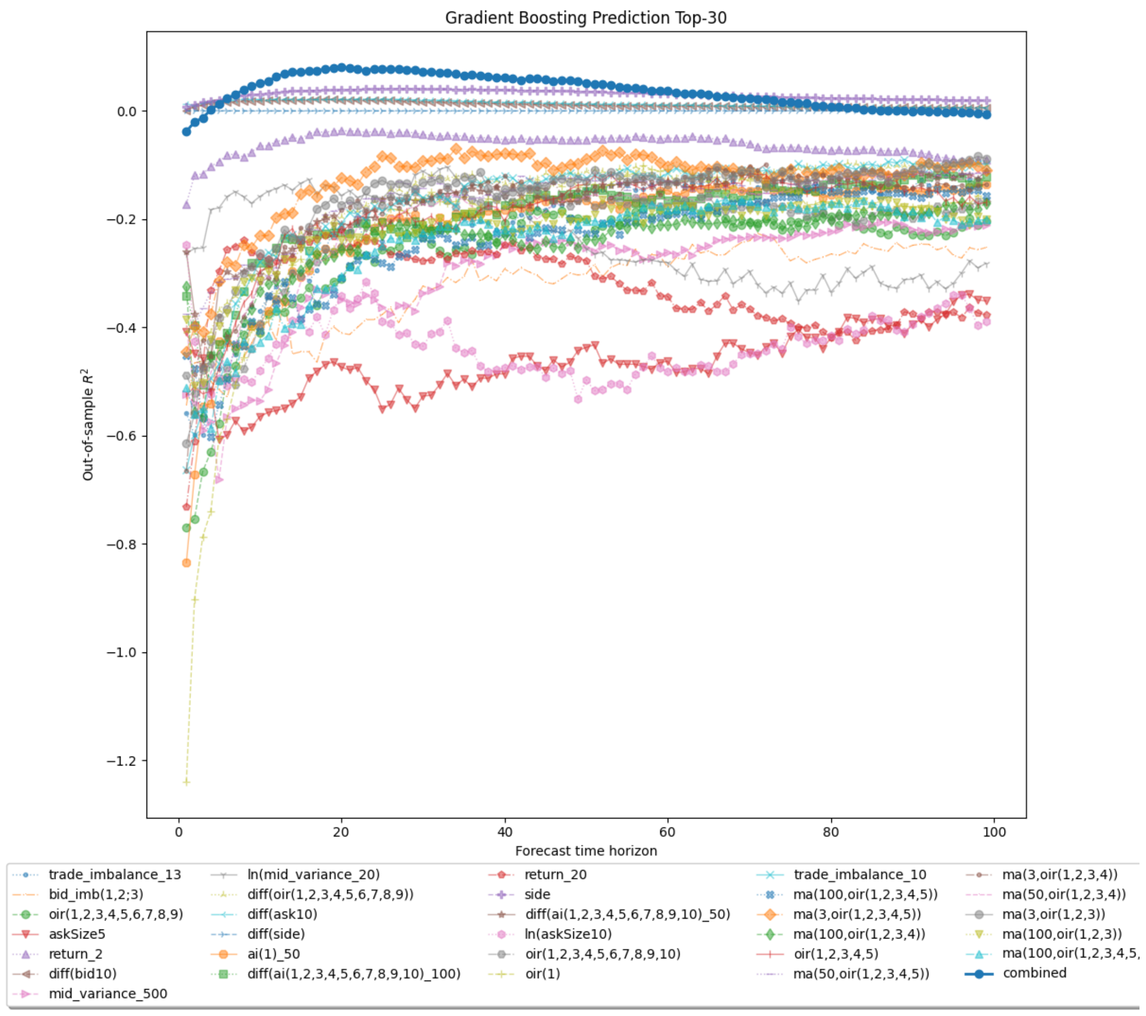


Figure 6.4: gradient boosting out of sample forecasting

Gradient boosting (GB) model should produce the best results compared to other models. Figure 6.4 shows that the combined result of R^2 are approximately 0.1 at the steps in between 15 and 25. The best feature for this model is the 'Side' of the trades. However most of the other features have negative values with the feature 'AskSize5' has the worst performance.

The reasons for the poor performance could be explained as follow:

1. Similar to RF, gradient boosting uses the features selected by LR, therefore this may not be appropriate to GB. Some features that have non-linear relationship with the target that could produce better results with GB model may not be captured by LR during the feature selection stage.

2. As GB algorithm forms a strong learner by training several weak learners iteratively, where subsequent models learn from prior errors, and aggregating the predictions with a deterministic procedure. If the features selected by the LR are not fitted with the GB since the

beginning, with the 'boosting' effect provided by the GB for the next few iterations, it could make the results become worse. This could explain why most of the features selected by the LR are not performing well.

6.5 Comparison

When putting all the combined results together, we can have the figure shown as below. (figure 6.5)

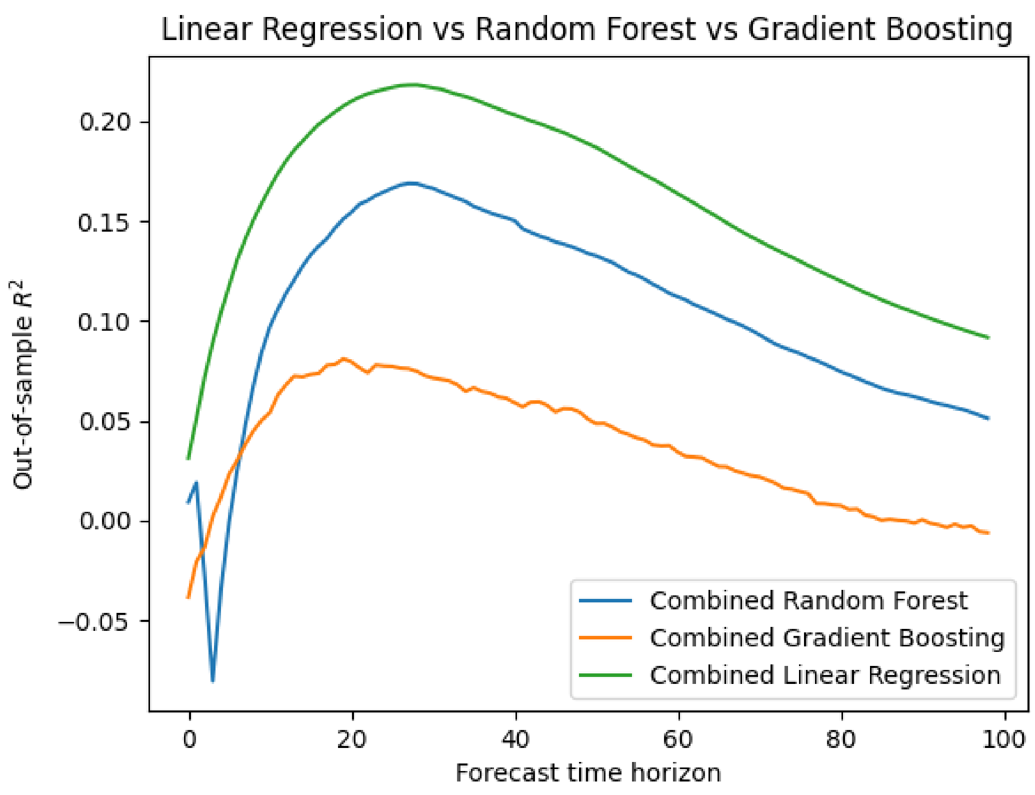


Figure 6.5: model comparison

Figure 6.5 shows the combined results of three different models. Their out-of-sample R^2 values are stated in the 'Results' section of each model as above. LR model is the winner for this BTC's next 100-step forecasting experiment. This is followed by RF and then GB ranked the bottom. The result curves of these models have similar patterns from 10th step onward, the curves rises to the peak at around step 20 and then go downhill slowly from the peak.

Chapter 7

Experiment With Prediction Using Features Selected By Random Forest

7.1 Methodology

After observing the results obtained from the feature selection using linear regression (LR), we decide to make some modifications on the feature selection stage to find out if this will make better prediction. This time, the random forest (RF) model is used to select the best features for the prediction.

Firstly we change the model from LR to RF for feature selection. In order to save time to select the best features, we modified the Algorithm 3 by changing the length of selected columns to 3 for Forward Feature Construction technique. Also we resample the training and test data set to 15000 and 5000 respectively to further reduce the computational time. In addition, this will also save the memory within HPC. The modified algorithm is showed in algorithm 4.

With the modified algorithm, we select the top 5 features from the main parquet file which contains 957 features and top 3 features for the rest of moving-average related parquet files, each with 23 derived features (23 MA window sizes for each feature from main parquet file). The feature selection from main parquet file takes 24 hours and the selection from all other MA files takes 6 hours in total.

As mentioned in section 5.1, we have 957 MA files and it will be very time-consuming if we train the RF model with those MA-derived features in sequence. Therefore we process those MA files by using map-reduce technique.

1. First we divide 957 MA files into 48 file groups with each of them has 20 files except the last group, which has 17 files. All file groups are labelled from 0 to 47.

2. We produce 48 command scripts, one script for each file group. Each script will perform 4 for feature selection on all of the features from each file group.

3. All of these command scripts are submitted as jobs in HPC to be run in parallel.

4. The selected features from each file group are ranked and only top 3 features will be selected for next step.

5. We rank all 144 features again and then top 30 features will be selected for prediction.

The steps for the map-reduce method mentioned above are illustrated in 7.1.

Algorithm 4: Modified Feature Selection Algorithm

```
included_col = [];
included_col = features ;
selected_col = [];
prev_best_metric = 0;
best_metric = 0;
best_metric_col = [];
while length of selected_col $< 3$ do
    for each c in included_col do
        if c is already selected then
            continue;
        else
            current_col = [];
            Append current_col with selected_col;
            Add c into current_col;
            metrics = [];
            for each ts, vs in (data_training_set, validation_set) do
                X_train = compressed feature inputs from ts;
                Y_train = compressed feature output from ts;
                Train X_train to fit Y_train with model;
                X_validation = compressed feature inputs from vs;
                Y_validation = compressed feature output from vs;
                Predict y_pred using x_validation with the trained model;
                Calculate the metrix of the y_pred by comparing with y_validation;
                Add metric score into metrics;
            end
            mean_metric = average of metric scores;
            update the best_metric value and the feature with the best metric score;
        end
        if best_metric $-$ prev_best_metric $>$ metric_improvement_threshold then
            add best_metric_col into selected_col
        else
            prev_best_metric = best_metric
        end
    end
end
Result: selected_col
```

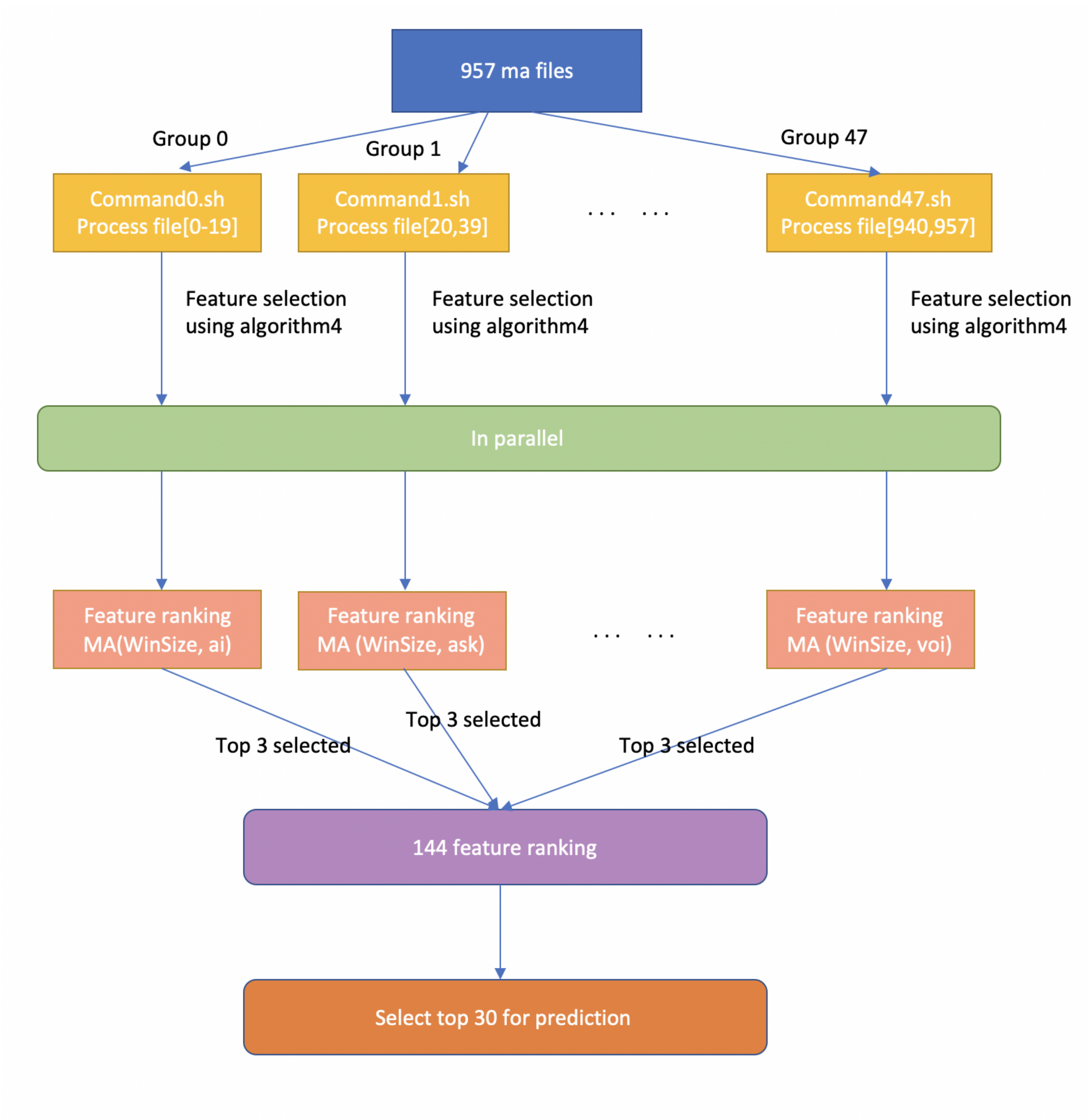

957 ma files
Group 0
Group 1
Group 47
Command0.sh Process file[0-19]
Command1.sh Process file[20,39]
Command47.sh Process file[940,957]
Feature selection using algorithm4
Feature selection using algorithm4
Feature selection using algorithm4
In parallel
Feature ranking MA(WinSize, ai)
Feature ranking MA (WinSize, ask)
Feature ranking MA (WinSize, voi)
Top 3 selected
Top 3 selected
Top 3 selected
144 feature ranking
Select top 30 for prediction

Figure 7.1: feature selection improvement: parallel running

7.2 Prediction Result

It can be seen on figure 7.2, figure 7.3 and figure 7.4, the chart patterns of the predictions are quite different with patterns we have seen on figure 6.3 and 6.4. At first sight, it may seem the the prediction results are not as good as the results shown in Chapter 6 in overall, however the results perform better when predicting first few steps.

In figure 7.2, the combined prediction result by using RF has quite high out-of-sample R^2 for the first few steps. The values at step 1 is around 0.35, at step 2 is 0.30, at step 3 is 0.22 at step 3 and etc. The values continue to diminish and reach zero after 20 steps.

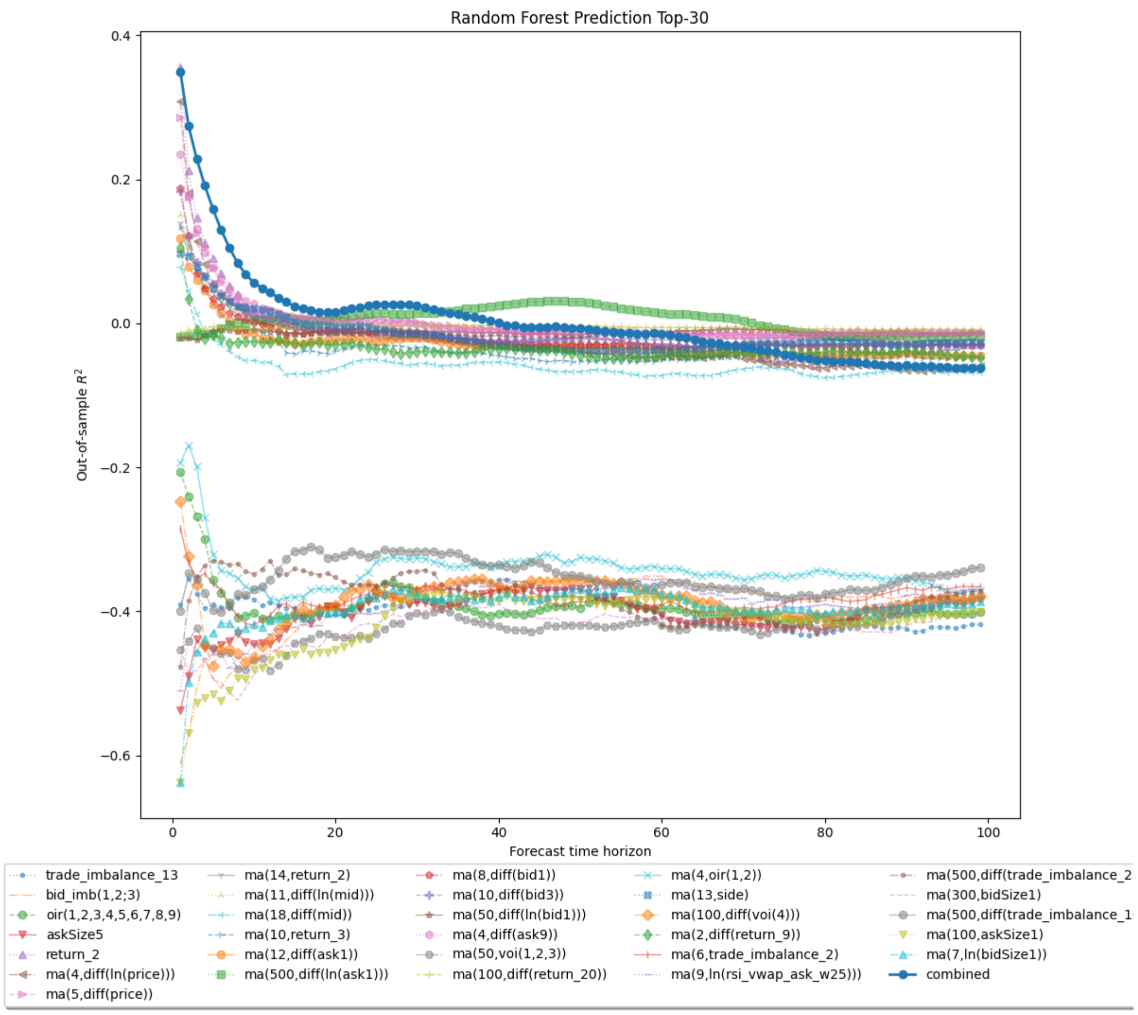


Figure 7.2: random forest forecasting by random forest feature selection

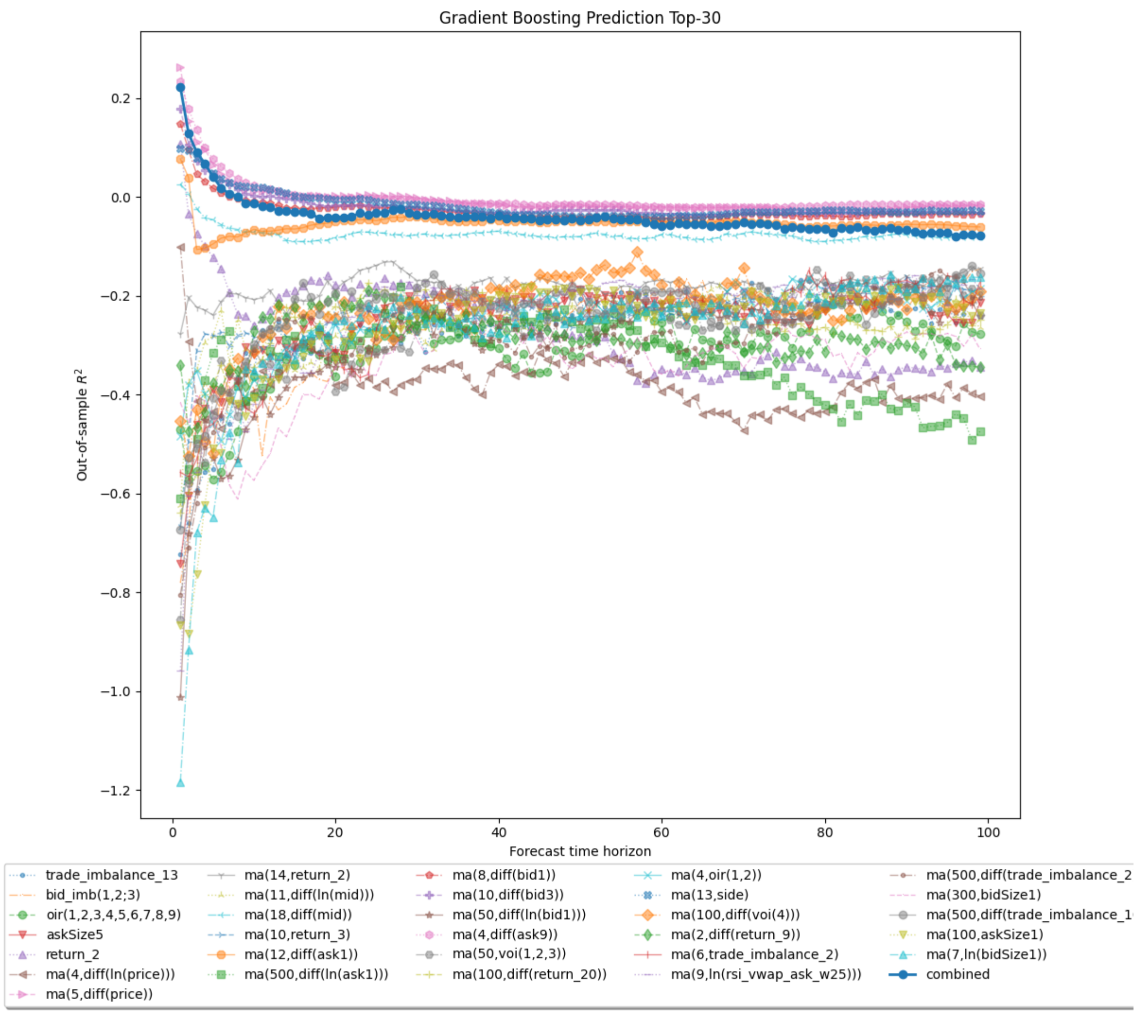


Figure 7.3: gradient boosting forecasting by random forest feature selection

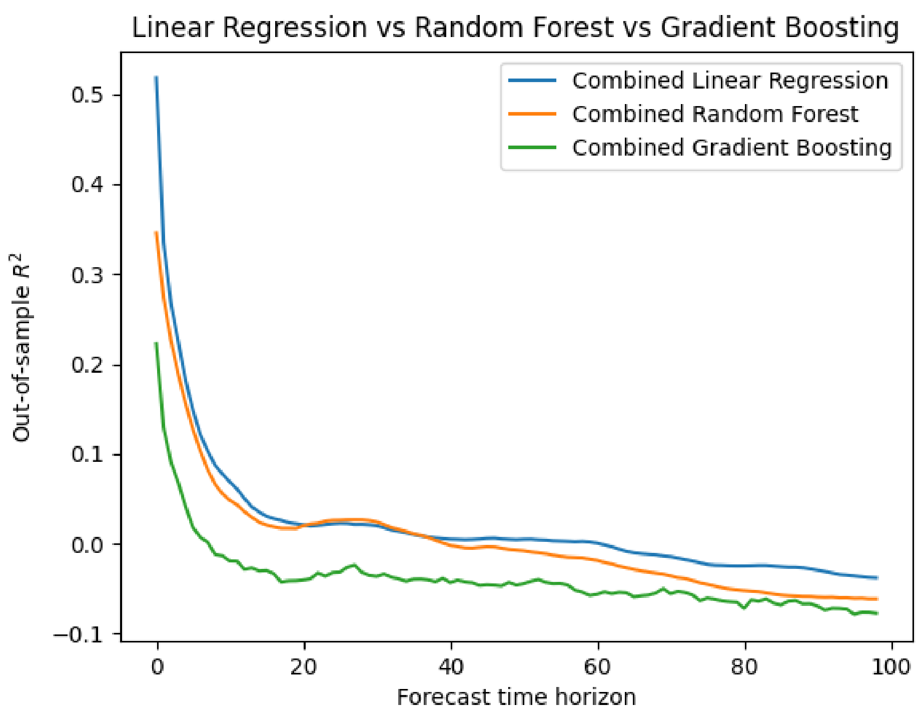


Figure 7.4: Comparison by random forest feature selection

In figure 7.3, the combined prediction R^2 shows that the predictions by using GB performs worse than RF in overall. At step 1, the R^2 value is around 0.22. The value is 0.15 at step 2 and is 0.10 at step 3. Similar to pattern of RF results, the R^2 values diminish at faster rate and reach zero at around step 10.

The performance of prediction results comparison for LR, RF and GB are shown in figure 7.4. LR has the best result, its out-of-sample R^2 value is more than 0.5 for the first step. Then RF comes next and GB is the worst.

It seems that by using data compression technique, we can improve the prediction for the next few steps. We believe with the data compression, there will be more volatility with the data due to the nature of data aggregation. For instance, we may not see any value change in the next 50 ticks in tick data, but for tick bar data, the change of value in current step and next step could be significant. The prediction models will have chance to learn these behaviours during the training stage and therefore make better prediction for the next few steps as opposed to the prediction made by tick data.

Chapter 8

Future Improvement

Due to time constraint, there are still many plans we have yet to complete and improve. The future improvement can be listed as below:

1. The tick data are inhomogeneous time series which contains too many useless and repeatable information. By changing the financial data structure such as converting the tick data into volume bar, tick bar time bar [Par18] to make them more homogeneous may provide more stable observations to train the models and hence more accurate in prediction.

2. In the future we plan to apply more advanced feature selection technique known as autoencoder (AE) [KH15]. The ability of AE to capture the most important features by performing non-linearity transformation without losing too much information is an powerful tool to select useful features with reduced dimensions.

3. We would like to use economic events as part of the features for forecasting. At the moment we derive most of the features from order book and trade data, we believe that by adding the economic events will provide us some new perspectives and different angles for forecasting.

4. Due to the fact that the financial data are mostly non-stationary and non-linear, we would like to use deep learning networks such as Long-Short Term Memory (LSTM), Generative Adversarial Network (GAN) for more efficient and accurate forecasting.

5. Some other model comparison algorithms [KJ20] could be used to compare the performance between two models. By using the algorithm 5, we could also evaluate each model and make the comparison to find the best predicting model.

Algorithm 5: Model comparisons using resampling -based performance metrics

```
for each resample do
    Use the resample's 70% to fit models $M_1$ and $M_2$ ;
    Predict the remaining 30% for both models;
    Compute the area under the ROC curve for $M_1$ and $M_2$;
    Determine the difference in the two AUC values.
end
Use a one sided t-test on the differences to test that $M_2$ is better than $M_1$
```

Chapter 9

Conclusion

In this paper, we have performed statistical analysis on the BTC time-series data. Then we augment the original features to form another 2062 enriched features. To avoid the insufficient memory issues due to large size of feature set, we use bespoke data storage methods and feature selection algorithm to rank these features. Grid search and differential evolution are used to search the optimised parameters for random forest (RF) and gradient boosting (GB), respectively. Three models, i.e. linear regression (LR), random forest (RF) and gradient boosting (GB) tree are used to predict the future 100-day prices.

From the experiment, LR has the best combined result among all three models we have tested. It has the out-of-sample R^2 value of around 0.2 in between step 15 and 30. The R^2 for the RF is approximately in between 0.15 and 0.18 in between steps 18 to 30. The GB has the worst result and it has the R^2 in between 0.05 and 0.08 around the peak in between steps 15 and 30. The chart in 6.5 shows similar patterns among 3 models in which their R^2 values reach the peak at around step 20 and then diminish afterwards.

From the results shown in figure 6.5, we observe that the features selected by LR are not suitable to be used by non-linear prediction models such as RF and GB. The reason for this could be the features selected by LR fail to capture the non-linear interactions with the non-linear models.

We conduct another experiment for feature selection by using RF and with data compression. The prediction results behave differently with the results shown in Chapter 6. It seems to have better forecasting power in the immediate vicinity of forecasting. Regarding to the performance among LR, RF and GB, LR is still the best model, RF is the second and GB is the worst.

It seems that by using the tick data without compression, it will have better prediction power for longer term for the BTC prices (up to 100 steps). However if the tick data is compression into tick bar, the models seem to do well in immediate forecasting in first few steps.

Bibliography

[Bre01] Leo Breiman. Random forests. *Machine Learning, 45, 5–32, 2001*, 2001. https://link.springer.com/content/pdf/10.1023/A:1010933404324.pdf.

[CKS14] Rama Cont, Arseniy Kukanov, and Sasha Stoikov. The price impact of order book events. *Journal of financial econometrics*, 12(1):47–88, 2014.

[CZ12] Yunqian Ma Cha Zhang. Ensemble machine learning methods and applications. *Springer Science+Business Media, LLC 2012*, 2012.

[Fam70] Eugene F. Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):383, may 1970.

[Fri99] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Doundations of TreeNet(tm)*, 1999. https://pdfs.semanticscholar.org/1679/beddda3a183714d380e944fe6bf586c083cd.pdf?_ga=2.238032968.917874185.1598224443-1812885405.1598224443.

[GMT] Gauss–markov theorem. in: The concise encyclopedia of statistics.

[Ham94] James Douglas Hamilton. *Time Series Analysis*. Princeton University Press, 1994.

[Han12] Hall J. Hanson, T. A. tatistical arbitrage trading strategies and high frequency trading. 2012. https://c.mql5.com/forextsd/forum/173/statistical_arbitrage_trading_strategies_and_high_frequency_trading.pdf.

[Jos08] Mark S. Joshi. *The Concepts and Practice of Mathematical Finance*. Cambridge University Press, 2 edition, 2008.

[JPB18] JonaThan Donier Jean-Philippe Bouchaud, Julius Bonart. *TRADES, QUOTES AND PRICES*. Financial Markets Under the Microscope. Cambridge University Press, 2018.

[JS08] W. James and Charles Stein. Estimation with quadratic loss. *University of California Press*, 2008. https://projecteuclid.org/download/pdf_1/euclid.bsmsp/1200512173.

[KH15] Chao Zhang Chao Li Chao Xu Kai Han, Yunhe Wang. Autoencoder inspired unsupervised feature selection. 2015. https://arxiv.org/pdf/1710.08310.pdf.

[KJ20] Max Kuhn and Kjell Johnson. Feature engineering and selection - a practical approach for predictive models. *CRC Press*, 2020. Visit the Taylor Francis Web site at http://www.taylorandfrancis.comandtheCRCPressWebsiteathttp://www.crcpress.com.

[LPS14] Alex Lipton, Umberto Pesavento, and Michael Sotiropoulos. Trading strategies via book imbalance. *Risk*, March 2014.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography mailing list at https://metzdowd.com*, March 2009.

[NBGD19] Jan Novotny, Paul Alexander Bilokon, Aris Galiotos, and Frédéric Délèze. *Machine Learning and Bid Data with kdb+/q*. Wiley, 2019.

[PA-] the Web site is http://https://arrow.apache.org/docs/python/parquet.html.

[Par18] Marcos Lopez De Pardo. Advances in financial machine learning. 2018. chapter two.

[PGP10] Sébastien Laurent Pierre Giot and Mikael Petitjean. Trading activity, realized volatility and jumps. *Journal of Empirical Finance, 17(1), 168-175.*, 2010. https://econpapers.repec.org/article/eeeempfin/v_3a17_3ay_3a2010_3ai_3a1_3ap_3a168-175.htm.

[PL19] Pavlo Liashchynskyi Petro Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for nas. *arXiv:1912.06059v1[cs.LG]*, December 2019. https://arxiv.org/pdf/1912.06059.pdf.

[PR15] Rui Rebelo Patrícia Ramos, Nicolau Santos. Performance of state space and arima models for consumer retail sales forecasting. *ELSEVIER*, 2015. `https://core.ac.uk/download/pdf/47142231.pdf`.

[Rip16] Alexandre Ripamonti. Corwin-schultz bid-ask spread estimator in the brazilian stock market. *Brazilian Administration Review*, 2016. `https://www.scielo.br/pdf/bar/v13n1/1807-7692-bar-13-01-00076.pdf`.

[RKL08] Wei-Hsiu Huang Pei-Chann Chang Robert K. Lai, Chin-Yuan Fan. Evolving and clustering fuzzy decision tree for financial time series data forecasting. *ELSEVIER*, 2008. `https://www.journals.elsevier.com/expert-systems-with-applications`.

[Roy] Andrew Royal. Short-term alpha signals. `https://www.fixglobal.com/home/short-term-alpha-signals/`.

[She15] Darryl Shen. Order imbalance based strategy in high frequency trading. Master's thesis, Linacre College, University of Oxford, 2015.

[Sil19] Eduard Silantyev. Order flow analysis of cryptocurrency markets. *Digital Finance*, 1(1-4):191–218, March 2019.

[Sto97] Price K. Storn, R. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11, 341–359*, 1997. `https://link.springer.com/article/10.1023/A:1008202821328#citeas`.

[WX19] Xiaoyong Zeng Feng Zhou Xiaoying Tian-Xiaoyan Peng Wenquan Xu, Hui Peng. A hybrid modelling method for time series forecasting based on a linear regression model and deep learning. *Springer Science+Business Media, LLC, part of Springer Nature 2019*, 2019. `https://link.springer.com/article/10.1007/s10489-019-01426-3`.

[Zha01] G. Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *ELSEVIER*, November 2001.