Discovering Impact of Market Microstructure on Stock Prices by Computational Intelligence Approach

(Odkrywanie wpływu mikrostruktury rynku
na cenę akcji przy użyciu
inteligencji obliczeniowej)

Karol Stuła Kacper Puchalski

Praca inżynierska

Promotor: dr hab. Piotr Wnuk-Lipiński

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

31 stycznia 2022

Abstract

Presented work describes process of discovering ways of representing current stock market state using microstructure attributes. Said representation is achieved thanks to understanding of the Limit Order Book and access to information describing singular trades. All data comes from companies from FTSE 100 in London Stock Exchange from 02.09.2013 to 13.09.2013. Produced data is applied onto problem of predicting possible price movements in the foreseeable future. We then compare and review performance of deep learning models on said task, alongside Logistic Regression, Decision Trees, XGBoost, MLP and LSTM.

Praca opisuje odkrywanie możliwych sposobów na przedstawienie obecnego stanu Giełdy Papierów Wartościowych przy użyciu parametrów mikrostruktury rynku. Reprezentacja jest uzyskana dzięki dostępowi do wpisów z arkusza zleceń oraz informacji o poszczególnych sprzedażach/kupnach akcji. Dane są pozyskane z wszystkich firm wchodzących w skład FTSE100 na Londyńskiej Giełdzie Papierów Wartościowych z okresu od 02.09.2013 d 13.09.2013. Otrzymane dane są zastosowane przy problemie przewidzenia możliwych ruchów ceny w najbliższej przyszłości. Następnie porównujemy i omawiamy działanie modeli głębokiego uczenia, regresji logistycznej, drzew decyzyjnych, XGBoosta, MLP oraz LSTMa.

Contents

Chapter 1

Introduction

Recently there have been a great change in world of finance, especially in stock market. Even 30 years ago people gained information about stock prices from newspapers, now it is done by few finger moves. What is more important, people started to gather and analyze data from the market which led to the situation where most of traders strategies are based only on dependencies found in data. There are many types of trading, but we can divide it into two main ones: low and high frequency trading. Many individuals practice low frequency trading, as they do not have access to all of the information about trades and orders. Putting focus on executing many trades in a small period of time is a high frequency trading. There are many approaches to this kind of task thanks to access to set of all orders called **Limit Order Book**. We can read about it in [11]. Hedge funds and other big institutions invest a lot of money to access all possible types of data.

As predicting precisely price of stock or other asset in the future is very hard, many people try to do that using data gained over the years. Nevertheless there are lots of other problems in this area of study. Some experts try to predict maximal and minimal swings in price like in [1] using very complex Deep Learning models applied on data, which contain information about all orders made. Others try to predict only direction of price movement using Reinforcement Learning (described in [3]). Every approach and type of problem depends on data the team has access to.

In our work, we will use data from all executed trades and set of all orders every minute called Limit Order Book (**LOB**), described precisely in Chapter 2, from 02.09.2013 to 13.09.2013 for companies, that are part of **FTSE100**, which is an index of 100 biggest companies in London Stock Exchange. Thanks to an access to such a specific and broad range of data, we want to check if knowledge about market microstructure and liquidity may help us improve quality of made predictions.

At the beginning, we calculated many parameters, described in Chapter 2, which we hope to measure the behaviour of the market. Then we apply data engineering to find best market state hyperparameters to get as good predictions as possible. We mostly selected values, that lead to the most balanced data set possible, but also similar to ones used in industry. Later we check dependencies between market parameters to find best set to use in our model. To find that, we checked correlation of parameters and based on this and our domain knowledge, we performed many experiments in which there were different sets of parameters. Thanks to these results, we could choose best set.

We try to predict swings in price by specific threshold, picked by analyzing data imbalance in a short time horizon (in our case one hour) given calculated market parameters from every minute. All details can be found in Chapter 5 and 6. Then we try few Machine Learning and Deep Learning models, such as Logistic Regression, Decision Trees, XGBoost, MLP and LSTM (explained in Chapter 3) to check, which one gives best results after proper data preparation and model fine-tuning. Based on results on this task, we want to find an answer if market microstructure have an impact on stock prices and if knowledge about it could help traders in predicting future behaviour of stock market.

Unfortunately, we found few problems, with which many people in this industry struggle. As every stock market data, our data set is imbalanced, which causes a lot of troubles, as all of our models tend to predict mostly one class. That is a very undesired situation, because it is very hard to compare results between models and choose the best approach in this situation. What is more, stock price of every company behaves differently, as there are multiple factors which have influence on it. To improve quality of predictions by significant margin, one would have to learn all of possible patterns, what is an exceedingly challenging task. Nevertheless, we tried to cope with this problems and perform very precise analysis.

Chapter 2

Preliminaries

2.1 Data

We are given data of every company in **FTSE100** (100 biggest companies in Stock Market in London) from 02.09.2013 - 13.09.2013. Data consists of Limit Order Book (**LOB**) snapshots, one for every minute during trading day, and records of trades which were executed at that time. Typical individual trader doesn't have access to these types of data, because costs of that type of data are very high. Only biggest institutions like hedge funds have access to this, hence they retrieve information of market microstructure and use this in various algorithms.

2.2 Limit Order Book

Limit Order Book (LOB) is a snapshot of all orders on stock market for given stock at specific time. Orders can be divided into two groups: **Bids** and **Asks**. Every order consists of two values: volume of stocks wanted to buy/sell and price. Every client may make an order: accordingly he can either make a bid to buy a specific amount of stocks for given price or sell it. After placing a bid, information about it is stored in order book. Limit Order Book has a FIFO order, so it means that when there were non-executed bids for exactly that amount of money previously, our bid will be executed after fulfilling these orders. As it was mentioned before, there are bid and ask orders. Price for bids are smaller than asks, because everyone would like to buy as cheap as possible and sell for a price as high as possible. There is always a gap between most expensive bid and cheapest ask. If bid price is equal or greater than ask price, then trade is executed and these orders are deleted from Limit Order Book. Great visualization of **LOB** is in [4] thesis made by Joaquin Fernandez-Tapia "*Modeling, optimization and estimation for the on-line control of trading algorithms in limit-order markets*" and much more information and analysis about it can be found in [9] and [10]. In our work, we use 10 best bids and 10 best asks.

2.3 Parameters of Market Microstructure

2.3.1 Mid Price

Mid price is simply a price between best (most expensive) buy order and best (cheapest) sell order. The formula for this parameter MP is simply

$$MP = (BEST\_BUY\_PRICE + BEST\_SELL\_PRICE)/2 \tag{2.1}$$

For example, when we have biggest bid price 1364£ and best ask price 1365£, according to (2.1), mid price MP will equal $(1364 + 1365)/2 = 1364.5$

2.3.2 True Price

In contrast to *mid price*, true price TP take into account volumes of best bid and ask, and take weighted average of this prices given weights as volumes. Assuming, that best buy price is BP, best buy volume is BV, best sell price is SP and best sell volume SV, we may write formula as

$$TP = \frac{BP \cdot BV + SP \cdot SV}{BV + SV} \tag{2.2}$$

If a best bid is for 1500 stocks at 1364£ and best ask for 500 stocks and 1365£ then using Formula (2.2), true price will equal $(1500 * 1364 + 500 * 1365)/1500 = 1364.25$ Figure 2.1 perfectly presents difference between bid, ask and true market prices. We can see, that true price is also between best bid and best ask price, and when these three lines crosses each other, then the trade is executed.

2.3.3 Order Imbalance

When a trader sees that there are much more ask volume than bid volume, it is very likely that price will go down in the next moment, because there is a bigger demand to sell stock in the market. That's what *order imbalance* OI tell us.

$$OI = ln(BIDvolume/ASKvolume) \tag{2.3}$$

We calculate it as in Equation 2.3 and think of it as a direction of market demand at specific time. So we can see that when bid and ask volumes are equal, then order imbalance OI would be 0 because using (2.3) $ln(1) = 0$

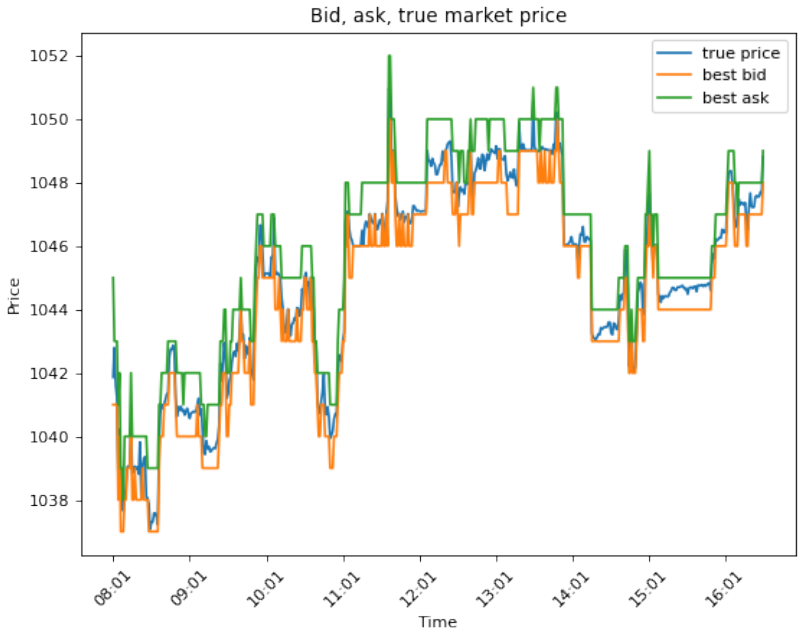

Figure 2.1: Dependencies between besk ask, best bid and true price. We can see that true price is always between best bid price and best ask price. Moreover, when these three lines crosses each other, trade is executed

2.3.4 VWAP

VWAP is a shortage of volume weighted average prices. It helps us estimate hidden liquidity of the market. We can calculate this on data from trades which were executed during specific time period t. At the beginning of this period we calculate true price p, then we save information about trades executed during t. We can divide every trade into two categories: buy and sell. Every sell trade was executed by a price lower than true price p and accordingly buy trade was executed by the higher price than p. Then we can calculate VWAPs for buy and sell trades. So if we have set of trades T performed in some time horizon t which contains pairs $<p_i, v_i>$, where p_i means price of i-th executed trade and v_i volume of that trade, we can compute VWAP for asks as

$$VWAP_{ask} = \frac{1}{\sum_{j=1}^{n} v_{a_j}} * \sum_{i=1}^{n} p_{a_i} * v_{a_i} \tag{2.4}$$

Accordingly we can calculate VWAP for buys as

$$VWAP_{bid} = \frac{1}{\sum_{j=1}^{n} v_{b_j}} * \sum_{i=1}^{n} p_{b_i} * v_{b_i} \tag{2.5}$$

Then the VWAP impact for buys is simply

$$VWAP_{impact\_bid} = VWAP_{bid} - true\_price \tag{2.6}$$

similarly VWAP impact for sells

$$VWAP_{impact\_sell} = |VWAP_{ask} - true\_price| \tag{2.7}$$

For example, when our trades look like [8, 99.95£], [13, 100.05£], [19, 99.85£], [6, 100.15£], [16, 100.10£] and true price was 100£, then VWAPs for sells is

$$(8 * 99.95 + 19 * 99.85)/27 = 99.88 \tag{2.8}$$

and for buys:

$$(13 * 100.05 + 6 * 100.15 + 16 * 100.10)/35 = 100.09 \tag{2.9}$$

We can also calculate impact of VWAP for buys and sells as simply subtraction VWAP and true price. So in our example, VWAP impact of buying 35 contracts will be $100.09 - 100 = 0.09$ and accordingly VWAP impact of selling 27 contracts will be $|99.88 - 100| = 0.12$

2.3.5 Sweep to Fill

Sweep to fill is an operation performed by traders who want to buy a specific amount of contracts instantaneously at time t no matter what stock price is. Of course it could be divided into buy and sell category. If an order contain a very high volume of stocks, it may have a big impact on market, so the gap between best bid and ask in **LOB** grows. In effect the stock price is changing in the direction dependent on category of sweep to fill order. We can see it by looking at Figure 2.2 where difference between sweep to fill cost and true price for different groups of orders is shown. We can observe the pattern, that the bigger volume we want to sweep, the bigger price we would pay to do that and so it lead to bigger change in stock price. Figure 2.3 presents us the impact on market of sweep to fill for given order size. The impact is simply calculated as difference between true prices before and after the sweep to fill operation.

2.3.6 Time to Next Trade

It is a simple indicator which tells us, for every moment m, how much time has passed since m until a trade was executed. It may gives us a lot of information when we combine it with other parameters. For example when we set time to next trade with an order imbalance, then we can see that the bigger absolute value of order imbalance the shorter time to next trade. This behaviour is illustrated on Figure 2.4 .

It gives us a very clear conclusion because the bigger absolute value of order imbalance is, the shorter time to next trade. It seems very reasonable from market sight because bigger order imbalance parameter tells us that stock price is undervalued or overvalued (depends on order imbalance sign) and traders very often use this information to make trades.

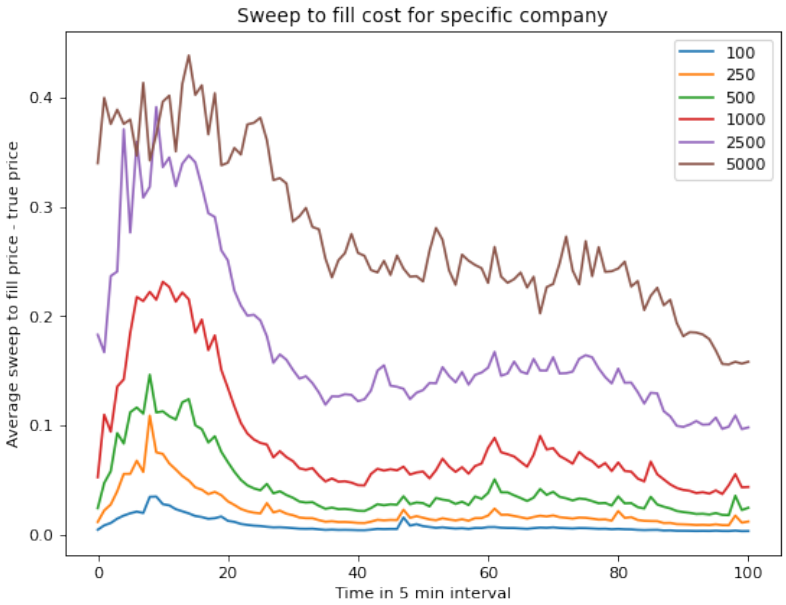

Figure 2.2: Sweep to fill cost for specific company. Here we can clearly see that the bigger order size is, the more money we need to pay to perform this order

2.3.7 Next Trade Price

This parameter is very similar to previous one. Information gained is, given moment m, what was the price of the next executed trade after m. We calculate next trade price at every available timestamp, which combined with order imbalance presents a very interesting plot (Figure 2.5).

Obvious observation would be, that traders prefer to sell stocks (take place at ask price) when order imbalance is positive and buy stocks (take place at bid price) when imbalance is negative. We can also see in Figure 2.6 the relationship between next trade price and order imbalance. There are three solid lines and two thinner. Most of the dots occur at a difference of -0.25£ and +0.25£. It represents half of a tick (measurement of the minimum upward or downward movement in the price of a stock). These cases represent instances of the price trading either at the ask or at the bid when the market was 1 tick wide. There are a few cases where the price change was a full tick (either +0.5 or -0.5) or zero. These outcomes correspond to times when the market was two ticks wide.

2.3.8 Next Trade Volume

Similarly to previous indicator, this one represent the size of first trade being executed after moment m.

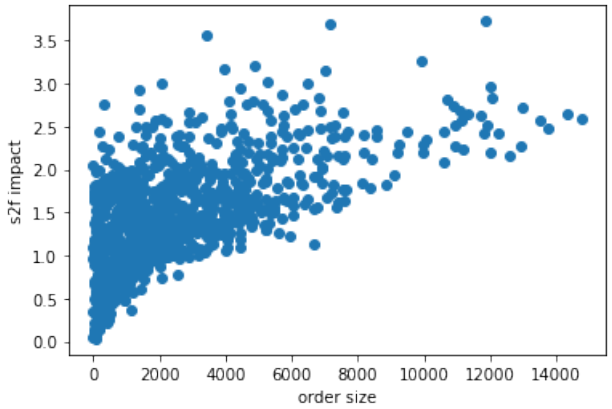

Figure 2.3: Sweep to fill impact for every order size. Here we can see that the bigger order size of sweep to fill operation is, the bigger impact (difference between true prices before and after the sweep to fill) on market it has. It leads to bigger stock price after this order

2.3.9 Trading Volumes

Trading volumes, as name may suggest, are simply summed volumes of every trade during some time interval.

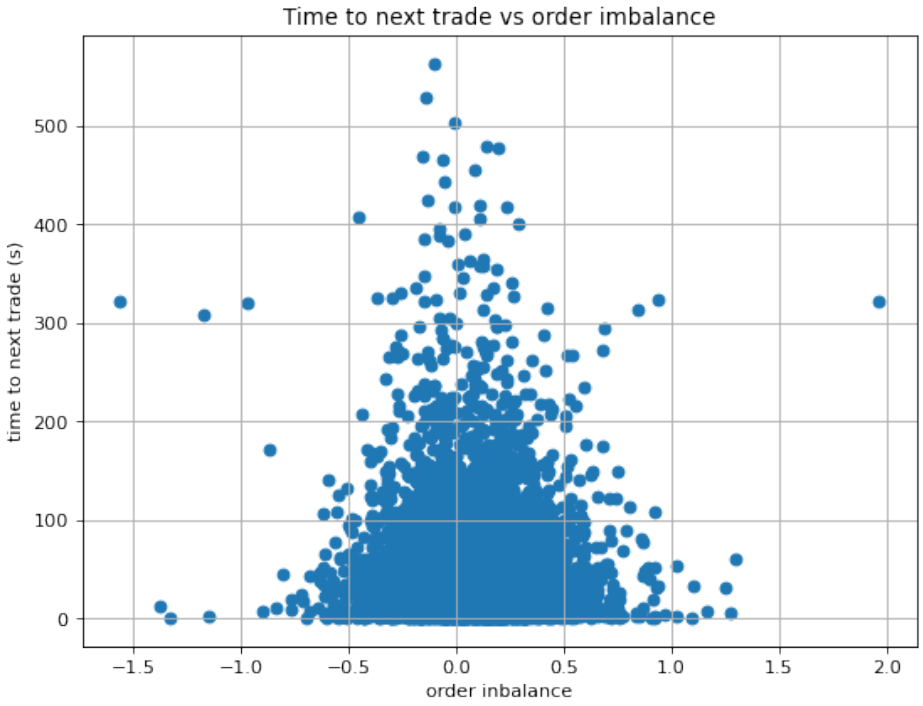

Figure 2.4: Dependence between time to next trade and order imbalance. We can clearly see that the bigger order imbalance, the shorter time to next trade. Big order imbalance means that the stock price is undervalued or overvalued (depends on sign of this parameter's value). Of course when order book is balanced, then on average we need to wait longer for next trade

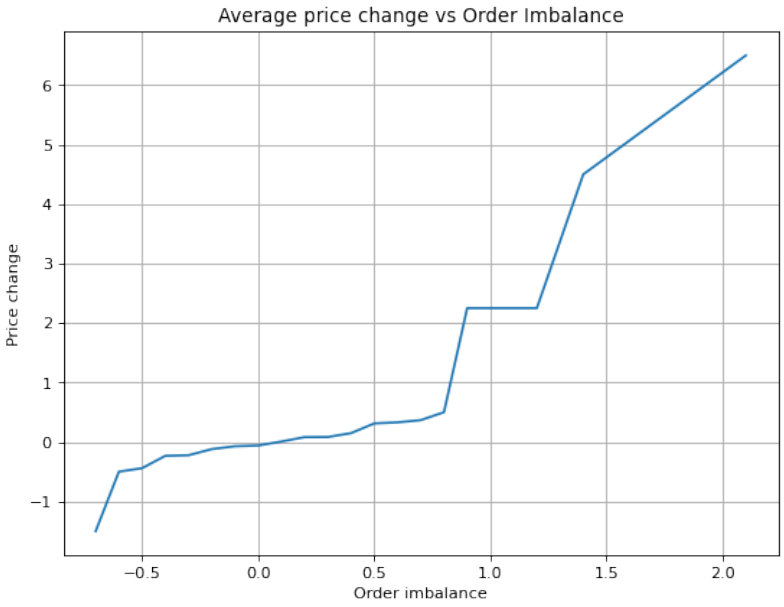

Figure 2.5: Reliance between average price change and order imbalance. Here we can observe that traders prefer to sell stocks when order imbalance is positive (this situation means that price of stock is overvalued) and buy when imbalance is negative (this situation means that the price of stock is undervalued)

Figure 2.6: Dependence between next trade price and order imbalance. There occurs 3 solid lines, where dots lies on value equal half of a tick of the stock, which represent instances of the price trading when market was 1 tick wide. Two thinner lines represents few cases where the price change was equal full tick which correspond to times when the market was two ticks wide

Chapter 3

Models

3.1 Introduction

In this chapter we want to present models which we use in our work and shortly explain how they work.

3.2 Logistic Regression

Logistic regression is a special case of linear regression where the target is categorical. As we will tackle a binary classification problem, it sounds like a good approach. Logistic regression predicts probability of occurrence of a binary event utilizing a logit function. To remind basics, Linear Regression equation is:

$$y = \beta_0 + \beta_1 * X_1 + ... + \beta_n * X_n \tag{3.1}$$

where y - our output is a dependant variable, $X_0...X_n$ is the explanatory variable and $\beta_0...\beta_n$ are coefficients of every dependant variable. Coefficients may also suggest which parameter have biggest impact on prediction. We can obtain formula of logistic regression for binary classification when we apply sigmoid function to linear regression. Sigmoid function is usually referred to as logistic function. Logistic function has a very useful property that it maps every output to value from 0 to 1. Thanks to this it can be interpreted as a probability. Its formula is as follows:

$$S(y) = \frac{1}{1 + \exp^{-y}} \tag{3.2}$$

So when we apply this function on linear regression we get logistic regression formula which return value from 0 to 1 - probability of obtaining specific class.

$$p = \frac{1}{1 + \exp^{-(\beta_0 + \beta_1 * X_1 + ... + \beta_n * X_n)}} \tag{3.3}$$

More about Logistic Regression could be read in [5].

3.3 Decision Tree

Decision tree is a model which has a flowchart-like tree structure. Internal node in the tree represents feature or attribute, branch a decision rule and each leaf node - outcome. Algorithm learns to partition a tree on the basis of the attribute value. It partitions the tree in recursively manner. Such a specific structure of a decision tree helps in decision making. Its visualization as a diagram mimics the human level of thinking. That is why it is very easy to interpret and understand.

Basic idea of decision tree algorithm is:

1. Select the best attribute using Attribute Selection Measures(ASM) to split the records.
2. Make that attribute a decision node and breaks the dataset into smaller subsets.
3. Starts tree building by repeating this process recursively for each child until one of the condition will match:
   - All the tuples belong to the same attribute value.
   - There are no more remaining attributes.
   - There are no more instances.

There are three main Attribute Selection Measures: Entropy, Information gain and Gini index. We chose Gini index, because it performed best out of other measures. We can formulate Gini value as

$$Gini(X) = 1 - \sum_{i=1}^{n} P_i^2 \tag{3.4}$$

where P_i denotes the probability of an element being classified for a distinct class. For a binary split we can compute a weighted sum of the impurity of each partition. When we try to split on an attribute A data D into D_1 and D_2, the Gini index is as follows:

$$Gini_A(D) = \frac{|D_1|}{|D|} * Gini(D_1) + \frac{|D_2|}{D} * Gini(D_2) \tag{3.5}$$

At the end we check, which subset give the minimum gini index and that attribute is selected as a splitting attribute. We calculate it as:

$$\Delta Gini(A) = Gini(D) - Gini_A(D) \tag{3.6}$$

More information about Decision Tree could be gained from [6].

3.4 XGBoost

XGBoost (Extreme Gradient Boosting) is one of the most popular Machine Learning algorithms. It is well known from the fact that either this algorithm or complex neural networks win most competitions. It is a very popular algorithm because it mix all of simpler machine learning models ideas into one. what is more it has some enhancements such as parallelized tree building or in-built cross-validation.

XGBoost uses gradient boosting framework as a core of whole algorithm. Boosting is a sequential technique which works on the principle of an ensemble. It combines a set of weak learners and delivers improved prediction accuracy. It is important to remember that weak learner is a very simple model which is slightly better than random guessing, for example decision tree with depth= 1. At any moment *t*, model outcomes are weighed based on results of previous moment *t-1*. Its objective(loss function and regularization) function at iteration t which we need to minimize is

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{3.7}$$

where y_i is real value known from the training dataset, $\hat{y_i}$ is prediction of weak learner, l is a differentiable convex loss function that measures difference between $\hat{y_i}$ and y_i. Ω is an additional regularization term which helps to smooth the final weight to avoid overfitting. More details about XGBoost can we find in [7].

3.5 MLP

To quickly explain MLP, it is important to understand the simplest neural network - *perceptron*. It is very similar to biological neuron, which has axons and dendrites. *Perceptron* has a simple tree structure with multiple input nodes and one output node, which is connected to each input node. Every input node is a simply a number - could be integer or decimal. Every connection between input and output has an associated weight. Then all of the input values and their weights are brought together as a weighted sum:

$$y = \sum_{i=1}^{n} w_i * x_i + b \tag{3.8}$$

On the resulted weighted sum we apply non-linear activation function. There are many activation functions, such as $RELU : f(x) = max(0, x)$ or $Softmax : f(x)_i = \frac{\exp^{x_i}}{\sum_{j=1}^{K} \exp^{x_j}}$. At the end there could be added *bias*, which can be considered as the the weight associated with an additional input node that is permanently set to 1. The bias value is critical because it allows us to shift the activation function either way, which could determine the success of your learning. This diagram compare graphically biological neuron to perceptron.

When we understand what a perceptron is, we can define MLP - *Multi-Layer Perceptrons* as a networks of perceptrons. The more inputs and layers has MLP, the more complex is the model so we need to precisely and with a great caution set a size of neural network, because too small network would not be able to learn to solve complex problem, but too big model's training time lasts too long and very often we can experience overfitting to the data. More about MLP and training of the model we can read in [8].

3.6 LSTM

LSTM - Long Short-Term Memory is a special kind of Recurrent Neural Networks, which are capable of learning long-term dependencies. We may think about recurrent neural networks simply as copies of the same network, each passing a message to a successor. But simple RNN are known from the fact, that they look mostly at recent information. Unfortunately, when we need a bigger context to predict some class it does not perform well. Luckilly, *LSTM* cope with this task quite well. It is designed to avoid long-term dependency problem. It has also, as simple RNN, chain like structure, but the repeating module has a different structure - instead of having single neural network layer, it is built from 5 essential components, which interacts with each other in a special way:

- Cell state (c_t) - it represents the internal memory of the cell which stores both short term memory and long-term dependencies
- Hidden state (h_t) - it is an output state o_t information which is calculated with respect to current input, previous hidden state h_{t-1} and current cell input which we want to use in our prediction. Additionally, the hidden state h_t can decide to only retrive the short or long-term or both types of memory stored in the cell state to make the next prediction.
- Input gate (i_t) - it decides how much information from current input is transferred to the cell state c_t
- Forget gate (f_t) - it decides how much information from the current input i_t and the previous cell state c_{t-1} flows into the current cell state c_t
- Output gate (o_t) - it decides how much information from the current cell state c_t flows into the hidden state h_t, so that, if needed, LSTM can only pick the long-term memories or short-term memories and long-term memories

We can specify formulas to update every component at time t as:

$$f_t = \sigma(X_t U_f + H_{t-1} * W_f) \tag{3.9}$$

$$\hat{C}_t = \tanh\left(X_t * U_c + H_{t-1} * W_c\right) \tag{3.10}$$

$$I_t = \sigma(X_t * U_i + H_{t-1} * W_i) \tag{3.11}$$

$$O_t = \sigma(X_t * U_o + H_{t-1} * W_o) \tag{3.12}$$

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{3.13}$$

$$H_t = O_t * \tanh(C_t) \tag{3.14}$$

where W and U are weight vectors for each gate. More about structure of LSTM and its training process we can read in [8]

Chapter 4

Our Approach

4.1 Problem Definition

We are given data from **LOB** and information of every trade from period of ten trading days. We considered two problems to solve in our work. First one was a simple prediction if price of stock would go up or down after a short interval of time. Of course we wanted to eliminate random noise from our prediction, so we decided to ignore any movement lesser than 10^{-6} of stock price, which means that some stock price p needed to rise or fall by $10^{-6} \cdot p$ in order to state that p went up or down. As it is an interesting problem, we want to solve a more practical task, which would help potential trader during his work. We want to predict a swing in price in some future time horizon. For a chosen stock **s** in moment **m** we state, if absolute value of difference between price at m and stock price **p** during some time period (e.g. an hour) since m until m' in near future, exceeds a predefined value calculated with regard to price at exactly m.

4.1.1 Binary Classification

Throughout duration of checked time horizon price of stock can either rise enough to surpass specified value (*threshold*), fall by that much (exceeding the *threshold* in negative manner), stay at relatively same level or (if stock is unstable) both rise and fall by said amount. We struggle with defining of our classification problem, but there are two ways to solve it: either predict four classes at once (price will stay on same level, price will only rise by specific threshold, price will only fall by threshold and price will both rise and fall by threshold in some time interval) or perform binary classification (first one will be if price will rise by threshold and second one, similar to first, but for price falling).

We decided to continue with binary classification, as it is much more easier to classify the data set. In order to predict price swings in rise or fall we simply create separate models for each of those tasks.

4.2 Approach

Given the data, we calculated market microstructure parameters described in Chapter 2. Evaluating of model performances was quite difficult, our dataset is very imbalanced. That is why we check three scores and compare every one of them. The most obvious one is model accuracy which tell us, how many samples were classified correctly by our models. It is the most popular indicator, but for imbalanced dataset it won't necessarily be appropriate choice for comparison of models. That is why we would check two more statistics: precision and recall. Precision tells us what proportion of positive identifications was actually correct and recall shows us which proportion of actual positives was identified correctly.

Our main goal is to create models which results are, most importantly, trustworthy. We think of trustworthy model, as one that does not have falsely good performance. Say an untrustworthy model would (almost) always return either "1" action or "0" action, no matter the input. Such a model could yield very high precision and accuracy measures, if tested on imbalanced data. An untrustworthy model would have outmost values of recall, because there would be next to none false negatives $recall \approx 1$, or very little true positives $recall \approx 0$. That is why we need recall so much and check its value firstly during performance testing. It could save us from wrongly thinking we created a very precise model.

Secondly we focus on precision as we want it to be as high as possible. Reason is, the cost of making a wrong decision (false positive) is very high, since it means investing into a stock that might drop rapidly and not recover in near future. Not investing when it is advised is not so harmful, in view of the fact we would not lose money, just potential profit.

To sum it up, we use market microstructure parameters described in Chapter 2 to create representation of current market state. Afterwards we utilize said representation feeding it to various models, and if it is good enough the model should be able to answer mostly correct whether or no the price will go up or down by specified amount. If we succeed, our work could be used in high-frequency applications.

4.3 Data Usage

4.3.1 Scaling

There exists a large difference in order of magnitude between microstructure parameters. Some parameters belong mostly $\in [-1; 1]$ (e.g. order imbalance), when others such as order sizes might reach values of $\approx 10^4$. Such a difference might cause extra trouble when it comes to training models and extracting feature importance. Therefore we decided to scale each feature and compare performance.

Table 4.1: Average accuracy, precision, recall and time needed to train logistic regression model based on feature scaling method used. Model was trained and tested independently 3 times per each company in **FTSE100**. Task was to classify samples generated with *interval*= 1 minute, *time skipped* equal to half an hour, *times back*= 5 and *threshold* value of $th = 0.00167$. Sample from moment m is a positive if price of stock reached value of $(1 + th) * p(m)$ between m and some m', where $p(m)$ is price at moment m.

method	Accuracy (%)	Precision(%)	Recall(%)	Train time (s)
raw data	0.548	0.416	0.565	0.158
standard scaler	0.557	0.483	0.494	0.151
max abs scaler	0.550	0.457	0.553	0.069
l1 normalization	0.548	0.277	0.545	0.086
l2 normalization	0.548	0.417	0.574	0.083

From Table 4.1, we can tell that difference in model accuracy is relatively small no matter the method. On the other hand distinction of precision is quite significant. L1 normalization is the worst when it comes to correctness of models positive predictions. Such small value means if model returns a positive class, it is most likely a false positive. Furthermore l2 normalization, despite having much higher precision that l1 norm, has next to none impact on logistic regression performance. Average values, despite train time, are on the same level as when raw data is fed to the model.

Surprisingly standard scaler has almost the same average accuracy as max abs scaler, but is a little better regarding precision. However recall is in favour of max abs scaler, same with average train time which is more than twice shorter. Therefore we have trade-off between small decrease in precision and remarkable improvement in both train time and recall. So come we will use *Max Abs* scaler from this point onward.

4.3.2 Permutation of Data

Shuffling data samples is a regular practise when it comes to data set preparation. Especially in case of classification tasks it is done with purpose of achieving more contrasting train/validation/test sets. Although our problem focuses on binary classification, we should not use any kind of permutation on whole data set. Reason for that, is we are working on time series. Suppose we have a moment t and a permutation π, such that π puts data from moments $t, t+2$ in training set and data from $t+1$ in the test set. Any model would then make predictions of samples from $t+1$, given both past (t) and future $(t+2)$ data. In case a stock is in a trend that spans over $t, t+1, t+2$ more complex models could extract such an information resulting in much higher than normal accuracy of predictions. Models trained on permutated data yield from 5% up to 20% falsely better accuracy.

Above clearly does not apply to train set permutation during batch training of deep learning models.

Chapter 5

Calculation Setup

Chapter 2 introduced various parameters with which we represent current market state. They are only achievable thanks to the data we have. Hence comes the question, are they any better than features widely available? If so, how helpful are they when it comes to predicting price movement across the market? But before, we need to introduce and then find best model hyperparameters on which we would compare influence of market microstructure parameters. We strongly suppose our representation is indeed a good one and will prove it in this chapter. In order for the computations to be more understandable, let us explain how we exploit the **LOB** and trades information to produce needed data.

5.1 Data Representation

We think a clarification of how a Limit Order Book looks might come in handy later on. In Table 5.1 we give an example of single snapshot. Single bid (or ask) is given in a pair, where first number represents price at which an investor wishes to buy (sell) a certain amount of stock, represented by second number. Both bids and asks are sorted in ascending order. Respectively the best bid/ask is given as the last/first order in bids/asks list. We then collect information about 10 best bids and asks present at every minute. In combination with trades data (Table 5.2) we use them to calculate parameters described in Chapter 2.

Table 5.1: One snapshot of Limit Order Book at 08:53 02.09.2013. Single bid (or ask) is given in a pair, where first number represents price at which an investor wishes to buy (sell) a certain amount of stock, represented by second number. Both bids and asks are sorted in ascending order. Respectively the best bid/ask is given as the last/first order in bids/asks list

date	day time	bids	asks
2013.09.02	08:53	950 400, $\cdots$, 1040 3600	1042 5285, $\cdots$, 1110 5970

Table 5.2: Fragment of trades data set we use, from ≈14:20. We have information about price at which trade was executed, number of stocks traded and an exact time at which trade had occurred. In oppose to **LOB**, this data is not sorted

trade price	trade size	trade timestamp
2241.0	212	2013-09-09 14:28:03
2245.0	100	2013-09-09 14:21:21
2245.0	144	2013-09-09 14:21:10
2244.0	143	2013-09-09 14:20:48
2245.0	417	2013-09-09 14:21:37
2244.0	858	2013-09-09 14:20:52

5.2 Market State Hyperparameters

Before we delve deeper, let us define hyperparameters which dictate the method how the microstructure features are calculated. It will introduce methods we use in order to create our data set out of **LOB** and trades data, so that we could feed it to machine learning models later on.

5.2.1 Time Skipped

As we know on *London Stock Exchange* we can trade from 8:00 to 16:50. Orders may be added to Limit Order Book at any moment during the day, so moments before 8:00 there are loads of unmatched orders. When stock market opens up, one could observe a huge chaos within trading, a lot of trades are being executed and it is nearly impossible to get any meaningful information out. That is why we avoid collecting data samples from beginning of the trading day. Therefore in all of the following calculations we will be using *time skipped*= 30 minutes, e.g. first sample gained per each day is from 8:30.

5.2.2 Interval

Stands for time period from which we gather data in order to produce one sample. As stated in Chapter 2, our data consists of **LOB** snapshots, one every minute. Because of snapshot frequency default *interval*= 1 minute, whilst the data distribution prevents us from becoming more accurate.

We can classify our features into two classes. One would be parameters calculatable given only **LOB**, like *true price, order imbalance, bid ask spread.* Second class requires checking trades data from some time period, like *VWAP, s2f impact, trading volume.*

Example:

Suppose we want one sample of microstructure parameters at moment t =9:00 AM given *interval*= 5 minutes.

To calculate first class of features we would check **LOB** state at exactly t (9:00 AM) and apply on it our parameter calculation functions.

In the second case, we need to gather all trades data from period $(t-interval, t]$, which corresponds to $(8:55, 9:00]$, apply the math and return our features.

What follows is hopefully clear, as bigger *interval* translates to much smaller amount of available samples. 5 minute *interval* leaves us with about 100 samples per each trading day, that gives roughly 1000 samples big data set, which is a very shallow one. However 1 minute *interval* generates $\approx$ 5000 samples. That is why we decided to run our experiments on *interval*= 1 minute.

5.2.3 Times Back

That parameter is responsible for calculating moving average of last $k \geq 1$ samples. Sample at t is substituted with average of last k samples, including itself. Idea beneath it, is so we could represent data from previous e.g. half an hour, given much smaller *interval* (for *interval* = 5 minutes, k would be 6). We will use *time back*= 5, as representing last 5 minutes (combined with *interval*=1 minute) yielded the best results.

5.2.4 Sample Generation

While *Threshold* is a mean of sample classification, we firstly introduce the method of creating the data set. With above explanation of hyperparameters like *Interval* it should be comprehensible one.

We define a timestamp t (we may refer to it as a moment, depending on the context) such that total number of timestamps per trading day would be

$$number\_of\_timestamps = \frac{duration\_of\_trading\_day - time\_skipped}{interval} \quad (5.1)$$

where *duration_ of_ trading_ day*, *time_ skipped* and *interval* are all given in minutes. We then produce one sample of parameters per each timestamp.
It is easily to calculate, that given trading day that respectively starts and ends at $8:00AM - 4:30PM$, 1 minute *interval* and *time_ skipped*= 30 minutes, we get 480 samples.

5.2.5 Threshold

As said in Chapter 4, we are focusing on predicting whether or not maximum price swing in some near future crosses selected value. We name hyperparameter by which we determine that value a *threshold, th*. Given moment m and price $p(m)$ at m, it specifies the percentage by which stock price has to go up (down for minimum swing) in next time horizon h. If at any point during said period stock price is equal or greater than $p(m) * (1 + th)$ (for minimal swing less or equal than $p(m)$ times $1 - th$), then we classify the sample from m as a positive (Equation 5.2).

$$y = \begin{cases} 1 & \text{if } \exists_{m_i} p(m_i) >= p(m_0) * (1 + th) | i \in \{1, ..., h\} \\ & \text{where } m_h \text{ is the last moment in following time horizon} \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

We still need to determine the value, of how big such threshold needs to be. Of course if we select a small value $th \leq 0.001\%$ of current price, the probability of price going up (or down) by said amount is very high, as it is almost a random noise. Same applies when taking a big $th \geq 1\%$. In this scenario price will almost never swing by that much in foreseeable future.

Figure 5.1 shows that the optimum *threshold* value is somewhere in $[0.0012, , 0.0022]$, when checking next hour. As th decreases, models become less sensitive to input and tends to always predict a positive class, which we know because recall is very high. That is probably due to high number of positive samples in the data set. What happens as th increases is very similar. Lack of precision plot once again points to none of the test samples being classified as positive, therefore precision cannot be calculated. This is an indication that models for high values of *threshold* also ignore the input and mostly return predictions of negative class. Similar behaviour can be observed when checking price swings during next three hours, although at different values.

Apart from model's performance, we also check what is the positive/negative ratio of samples in our data set given specific threshold value. Needless to say, we want to get as balanced data set as possible. Cause for that is accurate measures of models functionality. Suppose a model always predicts positive class. If our data set is imbalanced in favour of positive samples, say $3 : 1$, then such a model would have 75% precision. That might seem great, but in reality means the model does not learn anything about the data at all.

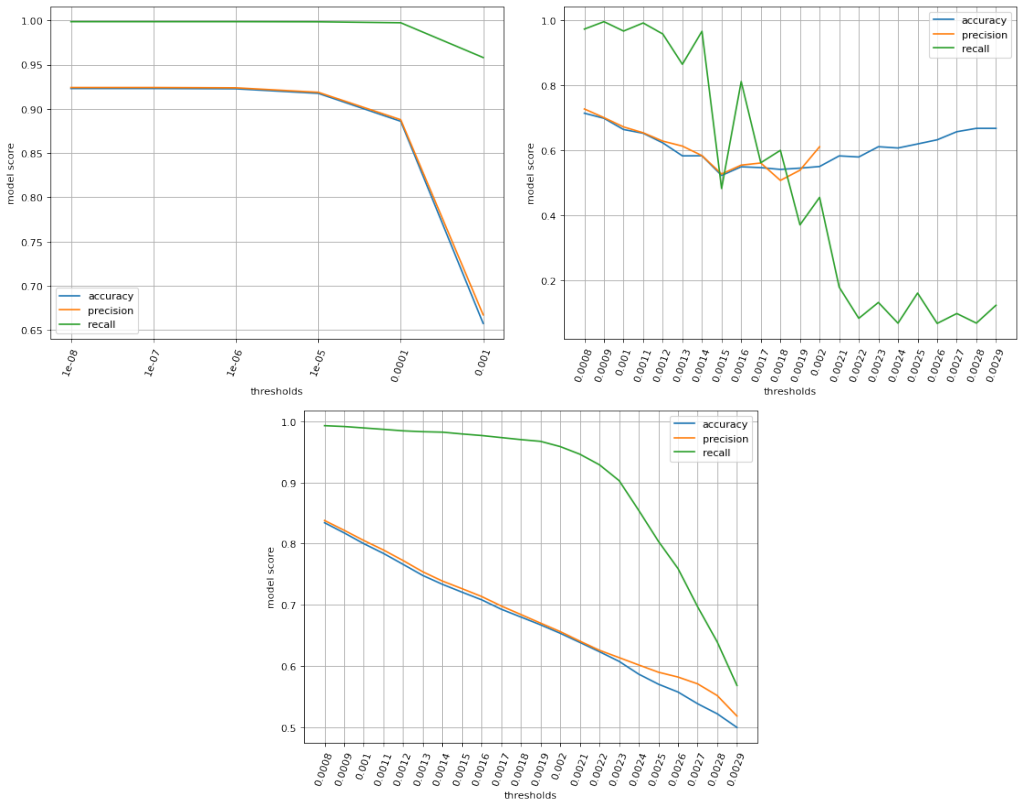

Figure 5.1: Each plot presents average of accuracy, precision and recall of logistic regression model run separately 5 times per each company in **FTSE100**. Model was put to task of classifying each sample, generated with *interval*= 1 minute, *time_skipped* equal to half an hour and *times_back*= 5. Sample from moment m is a positive if price of stock reached value of $(1+th)*p(m)$ between m and some m', where $p(m)$ is price at moment m. First two plots represent $th \in \{10^{-8}, 10^{-7}, \cdots, 10^{-3}\}$ and $th \in \{8 \cdot 10^{-4}, 9 \cdot 10^{-4}, \cdots, 3 \cdot 10^{-3}\}$ %, looking one hour into the future. Third plot takes the second set and 3 hours into the future

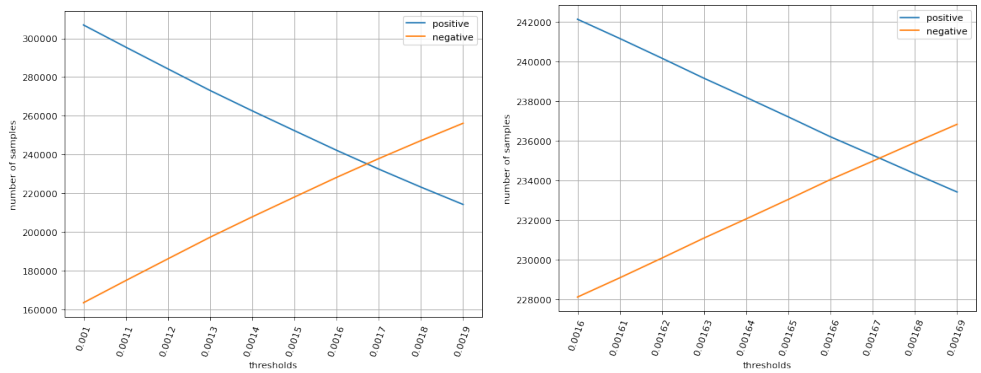

Figure 5.2: Number of positive and negative samples (for maximum swing) summed up for all companies in **FTSE100**. Sample is classified as positive or negative in the same way as in Figure 5.1. Notice difference in $X - axis$ scale

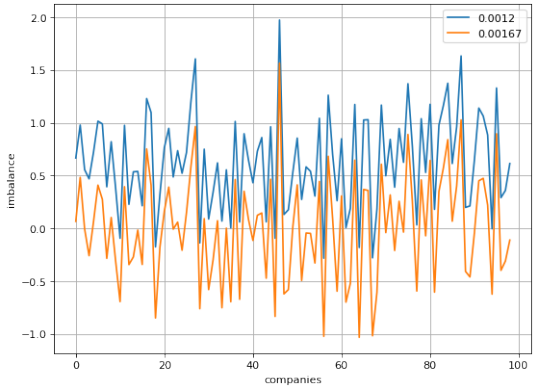

Figure 5.3: Given two *threshold* values of $th_1 = 0.0012$ and $th_2 = 0.00167$ and market hyperparameters as in Figure 5.1, we compare class imbalance of samples for each company in **FTSE100**. th_1 is first somewhat optimal value looking at Figure 5.1, th_2 is the best when it comes to average (over all companies) imbalance of positive to negative samples in data set (Figure 5.2). $log_2(\frac{positive}{negative})$ samples is shown with respect to each company

Figure 5.2 clearly shows, that the best balancing *threshold* value is ≈ 0.00167. As it represents averaged values of all companies, there is a risk that some still are largely positive or negative covered. The above is confirmed on Figure 5.3. Absolute value of the biggest imbalance is smaller than for 0.0012. However some stocks now present imbalance ≥ 1 in either direction rather than being only positive heavy. This confirms, that adjusting hyperparameters so they are applicable on every company in equal form is a wearying task. Thankfully, when *threshold*$= 0.00167$ is applied, imbalance in train and test sets is roughly the same (Figure 5.4), as we split the data set in $8:2$ manner.

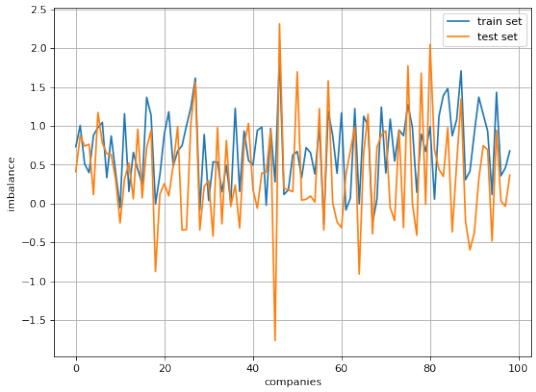

Figure 5.4: With hyperparameters as in Figure 5.1 and *threshold*$= 0.00167$, we plot $log_2(\frac{positive}{negative})$ of samples in each company (of **FTSE100**) train and test set. Split is performed in 8:2 fashion, which corresponds to data from 02.09-11.09 going into train set and samples out of 12.09,13.09 to test set

5.2.6 Final Configuration

To sum it up, we decided to set market hyperparameters values to those listed in Table 5.3. We will use that setup in all calculations later on, unless mentioned otherwise.

Table 5.3: Final values (units are provided if necessary) of market hyperparameters discussed in Section 5.2. We will use them in all of following calculations

hyperparameter	value
time skipped	30 minutes
interval	1 minute
times back	5 times
threshold	0.00167

5.3 Naive Technique

To show market microstructure features actually do a pleasant job when it comes to representing the market, suppose we are a private trader and do not have access to none of the complex market data. We compare models created only on one widely available parameter regarding stock market, which is a stock price alone. We might see what is models' average performance for each company of **FTSE100**.

Table 5.4: Averaged scores of accuracy, precision and recall for every model on every company from **FTSE100** given only true price. In this example the key is the recall line. We can clearly see that logistic regression completely cannot cope with this problem. For XGB and Decision trees we can see that for most of companies it can distinguish two classes, but there are some other cases, where recall is nearly 1 or nearly 0, which means that model predict nearly always one class, so it cannot learn correctly. There are also few examples, where precision line does not exist, so recall line was equal 0. Nevertheless, for general compare, we set *nan* values as 0 to find difference

	Accuracy (%)	Precision(%)	Recall(%)
logistic regression	55.1	26.8	55.5
decision tree	54.2	48.2	47.1
XGBoost	53.3	48.3	46.5
MLP	52.58	23.40	38.8
LSTM	56.5	28.5	48.4

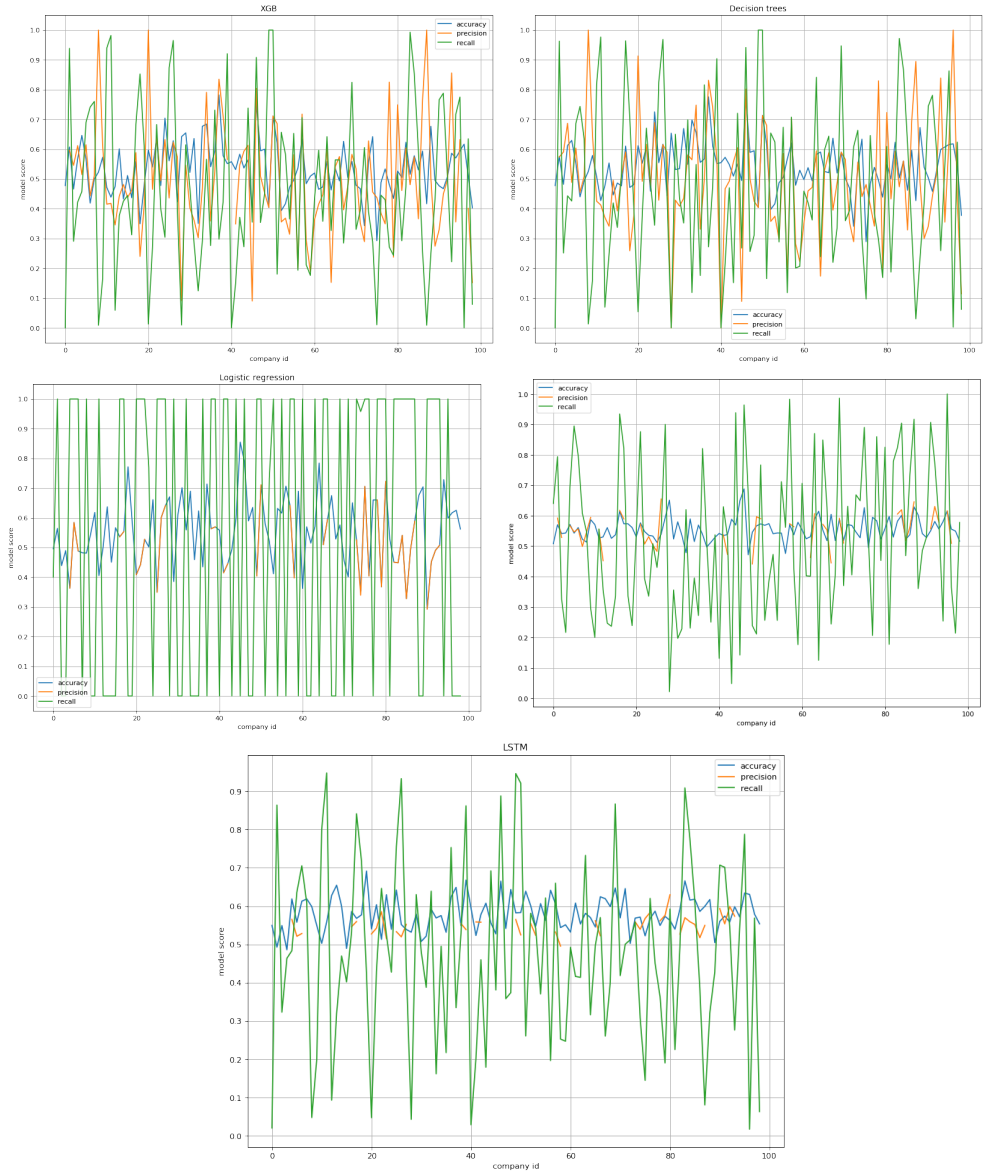

Figure 5.5: Average accuracy, precision and recall per company from **FTSE100**, trained and tested on single company data five times in order to reduce randomness, for all models given only true price. The most important is recall function. It shows us that models mostly cannot appropriately distinguish both classes. Logistic regression nearly always predicts only one class. XGB and Decision Trees for most than half of companies predict mostly one class with eventual few samples classified as other. MLP and LSTM has the best recall values. This statistic show us that beyond the fact that result are not so bad, we cannot take into serious consideration this approach as all results depend on data imbalance and in most cases every model cannot cope with this problem

Average accuracy of logistic regression with true price as one and only parameter is 55.1%, for Decision tree is 54.2%, for MLP 52.6%, for XGB 53.3% and for LSTM 56.5% . Accuracy over fifty percent comes mostly from imbalanced data for some companies. When we look at companies as a whole, the dataset is balanced, but there are companies that $\approx 75\%$ of samples are those of positive class but also there are companies that $\approx 67\%$ of samples are those of negative class (Figure 5.4). That is a great diversity in imbalance of dataset and it prove that it is very hard to learn models to distinguish classes. That is why we need to take into consideration also recall and precision. When it comes to precision, we can see that logistic regression and MLP have a great problem to solve this task. Model mostly predicts only one class for the whole test set, which corresponds to denominator = 0 in $precision = \frac{TP}{TP+FP}$. It clearly shows, that stock price alone is far from being enough when it comes to making any predicitons for this model. However MLP can distinguish samples into two classes in a little more cases, so we can generalize that beyond the accuracy it is better than logistic regression. The plot of recall also proves the point that logistic regression does not work at all in naive approach. Decision tree, XGB and LSTM also tend to predict only one class but not so often. Unfortunately we cannot calculate precision for decision trees and XGB for every company (as denominator is 0), so we state that data from naive approach is not enough in solving our problem.

5.4 Performance on All Features

The easiest approach to compare influence of market microstructure parameters is to simply take all parameters we calculated and feed them to one model. Splitting parameters like *VWAP* into both buy and sell values leaves us with about 14 features. We use the same model structure as in previous section, where only difference is in data provided. Here all of parameters representing microstructure are given.

Table 5.5: Average scores of accuracy, precision and recall for every model on every company in **FTSE100** given all market microstructure parameters. Accuracy is very similar to values in Table 5.4, precision is a little bit higher, beyond models that previously did not have precision, here we can calculate it. Nevertheless the recall statistics change a lot (for models in which we could calculate precision, recall have risen by $\approx 2\%$, and for models where we could not calculate precision, it fell by $\approx 1\%$). But real essence of improvement show us plots in Figure 5.6

model	accuracy (%)	precision (%)	recall (%)	train time (s)
logistic regression	54.81	45.48	54.84	0.0593
decision tree	53.62	48.46	49.73	0.0994
XGBoost	53.83	48.93	48.92	1.276
MLP	52.91	37.29	54.20	15.0037
LSTM	55.45	49.45	48.84	22.438

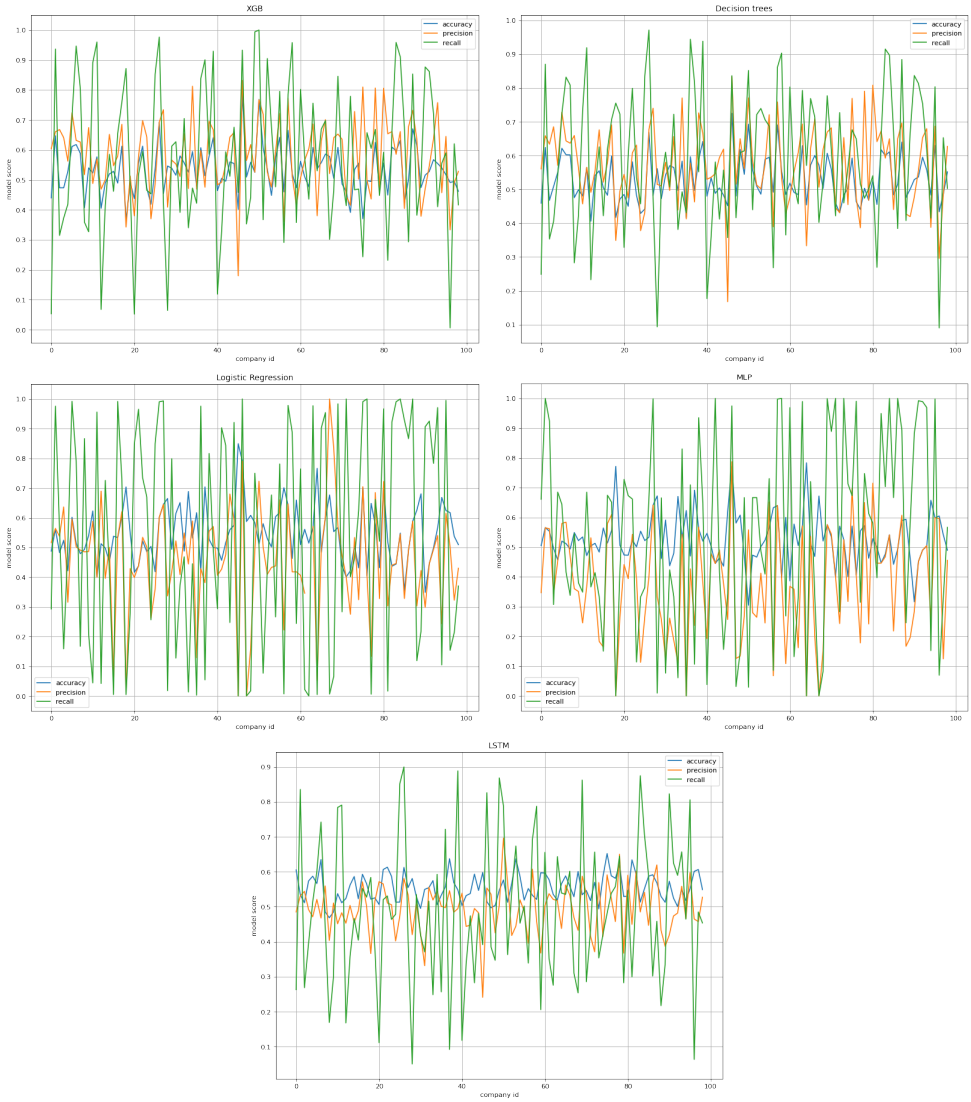

Figure 5.6: Average (5 train/test run) model accuracy, precision and recall per company from **FTSE100** for all features (trained and tested on single company data). Thanks to recall function, we see that only logistic regression still has problem with predicting single class only, yet it improved as well. For rest of our models, they predict both classes, which is very important. As it is very hard to predict price swings, results are around 50%, but still we can see a major improvement in predictions thanks to market microstructure knowledge

Accuracy for Logistic Regression looks very alike to Figure 5.5. This model still cannot cope with predicting different classes, as it predicts mostly only one class (we can see it thanks to recall line of plot). Nevertheless, we note a big improvement in performance for every model, as it can be observed in the recall statistic line, where there are less amount of values near 1 or 0 in all cases. It means that more often our models learn to distinguish both classes which is the core of our problem. What is more, models improved their results very strongly.

As accuracy for Decision Trees and XGB did not change in value, the precision metrics of both models emerged, which means that for every company, XGB, Decision Trees and LSTM learned to predict both classes which is the most important case. Unfortunately precision score is not very high in both cases (for XGB it was 48.9%, for Decision Trees 48.4% and for LSTM 49.5%) but the data is very complex and we can think of the results as satisfactory. Nevertheless we can see a big difference in results which prove us the importance and big influence of knowledge on market microstructure.

In following chapter we would like to improve performance of our models. We could naturally do so by focusing on models hyperparameters but that is not the goal of our work. Therefore we will conduct a series of experiments, trying to select a subset of market microstructure parameters that does the best when supplied to models with task of predicting if potential price swings are of satisfying scale. We hope that no matter how we select the subset, its value as market state representation is still much better than of raw price data. Even if we fail with outclassing models with full range of parameters, we should be able to improve on models computation time, as the data set used during training will be smaller.

Chapter 6

Computational experiments

6.1 Feature Importance Extraction

It is commonly known, that feeding machine learning models with as many features as possible is rarely a good idea. Such approach makes models more complex than they need to be, causes training to be more time consuming and hinders process of choosing right hyperparameters. Therefore performing a feature importance extraction is a crucial step in reducing model complexity and capturing the most valuable informations out of provided data set.

6.1.1 Heatmap

A good point to start is getting some insights on data created by calculation of market parameters. presented in a definition section in Chapter 2. Main goal is to find dependencies. That is why we calculated correlation between every pair of parameters (Figure 5.5). We do not want to check Pearson coefficient, because it measures only linear relationship between data, and that is clearly not the case when it comes to stock markets. Therefore we could use either Spearman's or Kendall's coefficient, as they represent rank correlations. That is why we will present heatmap of Kendall coefficient, because it calculates the strength of dependence between two random variables. When we consider two samples a and b, where each sample size is n, we know that total number of pairings with a and b is $(n * (n-1)/2)$. Then, given n_c as number of concordant (ordered in the same way) and n_d as number of discordant (ordered differently), value of Kendall rank correlation is

$$\tau = \frac{n_c - n_d}{\frac{1}{2} * n * (n-1)}$$

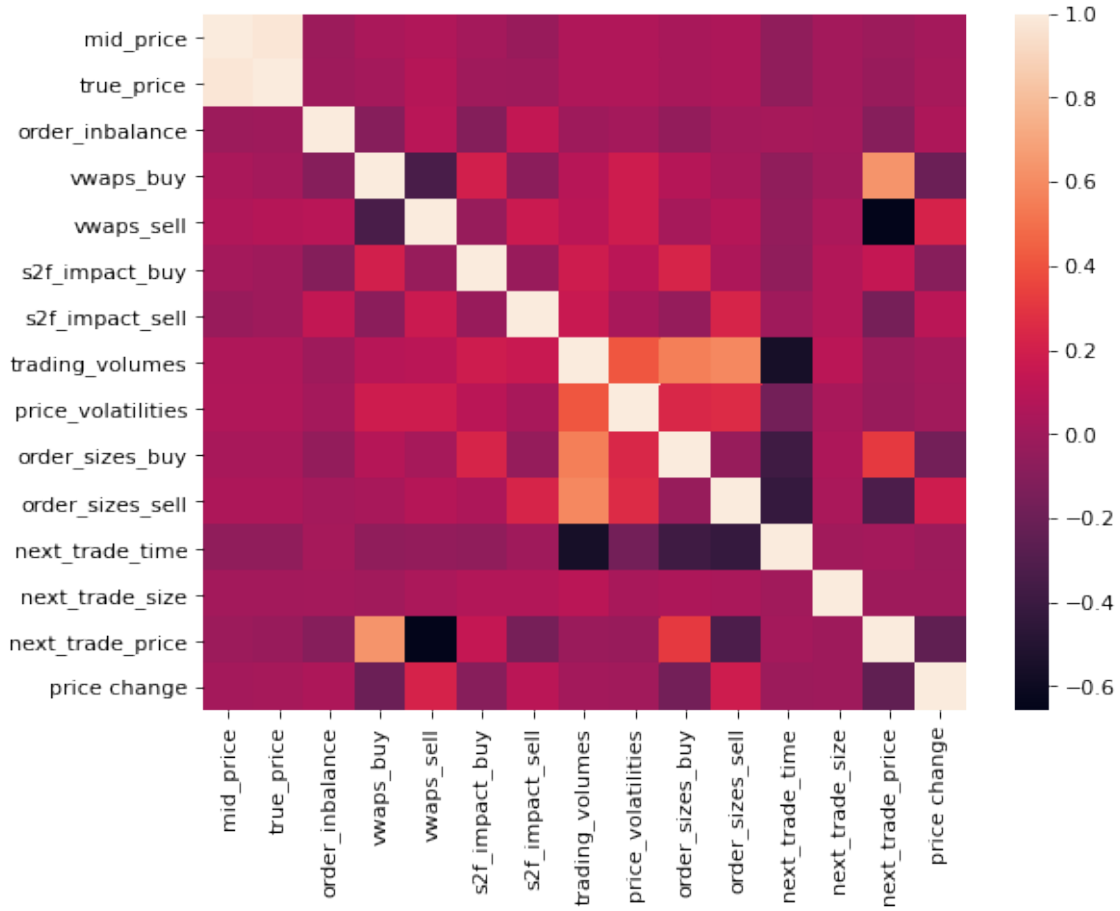

Figure 6.1: Values of Kendall's correlation coefficient between every parameter described in Chapter 2. Calculations were done given with regard to setup from Section 5.2.6. Parameters like *VWAP*, which value represents both buyers and sellers actions are splitted into 2 separate parameters for each respectively

On Figure 6.1 we can clearly see very obvious dependencies like big correlation between *mid price* and *true price*. Another major observation are *trading volumes*, which relatively strongly correlate with other parameters regarding volume being traded on the market, those being *order sizes*. Moreover *next trade price* has vivid correlation with both buy and sell version of *VWAP*. That gives an idea of which features could be discarded in order to decrease models complexity. Other parameters somewhat correlate with each other, but not enough to allow us straightforward elimination.

6.1.2 Skipping Parameters

Heatmap did not give us enough information about features importance. It seems like they are quite independent (with few outliers) of each other. Therefore we need to find another way (if there is any) to decide which parameters we can discard from further calculations.

Idea is pretty straightforward. Compose data set without one of microstructure parameters, train model on said data set, and compare its performance to model with all available parameters.

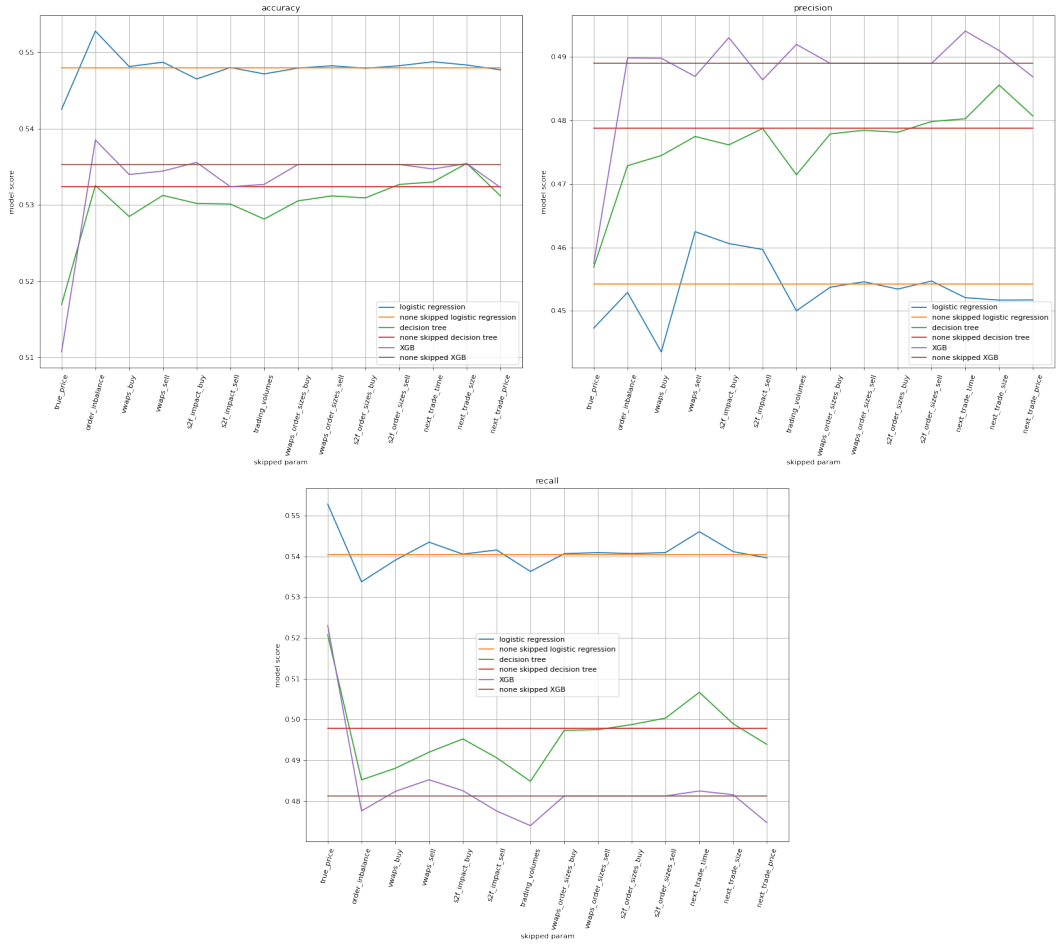

Figure 6.2: Different models performance (accuracy, precision and recall are displayed on separate plots), when singular feature (x-axis) is discarded from training set. Value plotted is an average of model accomplishments on every one of **FTSE100** companies with setup described in 5.2.6. Respective horizontal lines present average value when no feature is dropped

Based on Figure 6.2, we can conclude which parameters influence our models the most. Needles to say, if model performs better (on average) without some parameter, we should consider getting rid of it. Same applies to opposite case, when predictions worsen whilst discarding a feature, most rational is to keep it in the data set. We will focus on results from plot representing precision, as potential mistakes during trading made due to false positives would be the most costly.

6.2 Models Performance on Trimmed Data Set

To verify assumptions from previous section, we decided to run some experiments. Most focus was given on finding a set of features that when fed to model, yielded similar or better performance than models on all possible parameters.
All calculations in this section follow setup described in Table 5.3.

6.2.1 Volumes

Parameters describing volume being traded have high correlation with each other, as observed on the heatmap (Figure 6.1). On plots from Figure 6.2 a confirmation takes place. When discarding *VWAP order sizes buy/sell* and corresponding sweep to fill parameters, models performance barely differs from that on all features. Therefore our first experiment will be to remove *VWAP order sizes buy/sell* and *s2f order sizes buy/sell* from data set.

Table 6.1: Performance of models on task of predicting whether or not maximum price swing in near future exceeds predefined value. Here all of features representing order sizes, apart from *trading volumes*, were discarded from data set. Value shown in table is an average of 3 train-test circles across every one of **FTSE100** companies, with setup described in 5.2.6

model	accuracy (%)	precision (%)	recall (%)	train time (s)
logistic regression	54.66	44.83	53.96	0.0548
decision tree	53.33	48.10	50.17	0.0826
XGBoost	53.50	49.18	48.80	1.211
MLP	53.23	39.82	49.84	16.134
LSTM	54.34	48.9	49.57	21.267

Comparing Table 6.1 with Table 5.5, we may tell models scores are exceptionally close. Distinctions are almost non existing, as the largest one does not surpass 1% with few exceptions. We can not name it an improvement in case of performance even if e.g. XGBoost achieved slightly better precision $\approx 0.25\%$, as we spotted such small differences to occur due to randomness involved in training process.

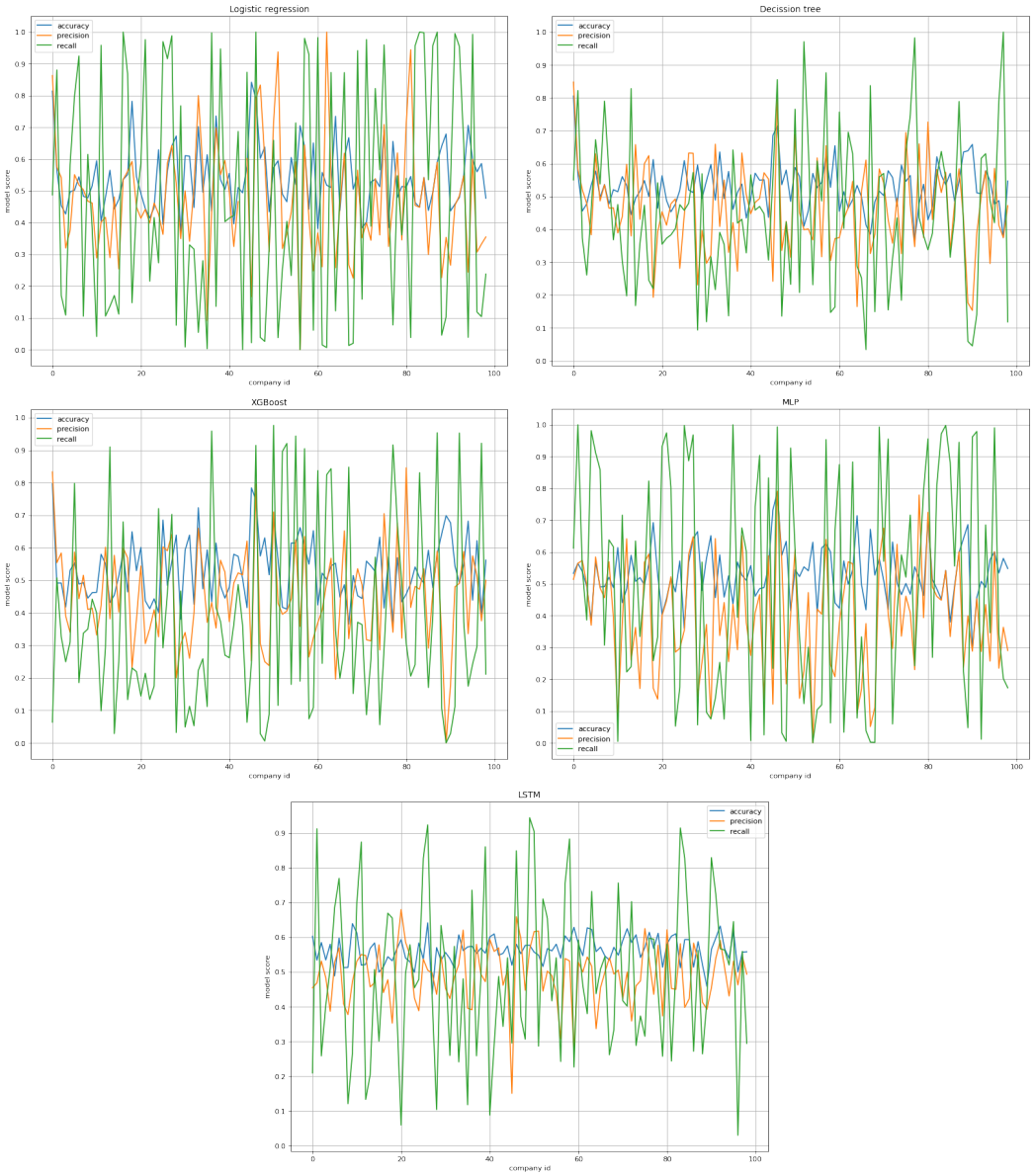

Figure 6.3: Performance of models on task of predicting whether or not maximum price swing in near future exceeds predefined value. Here all of features representing order sizes, apart from *trading volumes*, were discarded. Value plotted is an average of 3 train-test circles for every one of **FTSE100** companies, with setup described in 5.2.6

What stands out here, is noticeable speed up of decision tree training. Most likely, when we apply the model on truncated data set, the could be of lesser depth, therefore training faster. Opposite applies to MLP, as its precision rose by more than 2%, but training takes more time as well.

There are not many dissimilarities on scores per each company. Performance on data set without order sizes is shown on Figure 6.3. Logistic regression looks very alike to Figure 5.6. Decision tree and XGBoost also have similar plot, although now predictions of this models seem to be more negative heavy, as recall is more often $\leq 50\%$.

6.2.2 Next Trade Information

Recalling Figure 6.1, both *next trade price* and *next trade time* are in visible relation to some other parameters, although *next trade size* does not seem to share that behaviour. However from Figure 6.2 we assume that parameter alone is not a significant one. We will then check models performance whilst those three features are scrapped.

Table 6.2: Average performance of models on maximum price swing problem. Features representing *next trade price*, *next trade time* and *next trade size* were dropped from data set. Value shown in table is an average of 3 train-test circles across every one of **FTSE100** companies, with setup described in 5.2.6

model	accuracy (%)	precision (%)	recall (%)	train time (s)
logistic regression	54.83	45.93	54.51	0.0504
decision tree	53.38	48.43	50.40	0.0788
XGBoost	53.26	49.35	47.96	1.079
MLP	54.13	36.48		47.36
18.387				
LSTM	54.50	49.04	49.24	20.87

Looking at Table 6.2, we are unable to point out major contrasts to performance on all features. Nonetheless in decision tree training time has improved, similarly to previous test. Furthermore XGB also picked up on training speed. Models functioning on particular companies is not much different from represented in previous section, so plot is redundant.

6.2.3 Order Imbalance and Trading Volumes

Going other way around than in Section 6.2.1, we now reject *trading volumes*. It correlates strongly with various order sizes, so it should not heavily impact our models. We also get rid of *order imbalance* as models (mostly XGBoost) perform better without it.

Same as previously, Table 6.3 does not yield any more information regarding models accuracy, precision or recall. In opposition, train time worsens, although not much considering time considered with all of the features (Table 5.5).

Table 6.3: Average performance on models on maximum price swing problem. Features representing *trading volumes* and *order imbalance* were dropped from data set. Values shown in table are an average of 3 train-test circles across every one of **FTSE100** companies, with setup described in 5.2.6

model	accuracy (%)	precision (%)	recall (%)	train time (s)
logistic regression	55.19	45.60	52.85	0.0611
decision tree	53.24	47.70	48.53	0.0981
XGBoost	53.86	48.82	47.48	1.391
MLP	54.21	36.75	49.20	19.770
LSTM	53.97	49.05	48.24	21.524

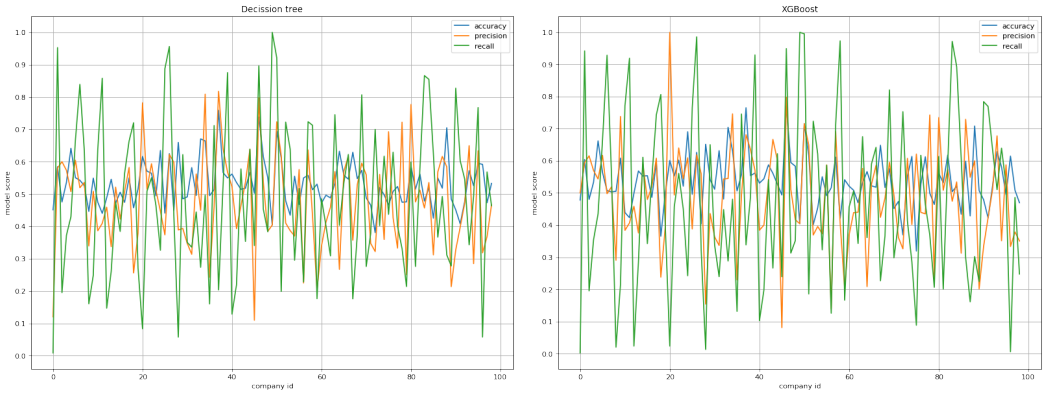

Figure 6.4: Performance of Decision Tree and XGBoost (other models did not have any interesting changes in behaviour) on task of predicting whether or not maximum price swing in near future exceeds predefined value. On those test we got rid of *order imbalance* and *trading volumes*. Value plotted is an average of 3 train-test circles on each of **FTSE100** companies separately, with setup described in 5.2.6. Most surprisingly, decision tree seems to be working much better than XGBoost

6.3 Outcome

6.3.1 Company Specific Models

In previous sections we gained some insight into model performances. We saw that models' score for some companies was great whilst tragic for other. Predicting price swings using XGB could yield up to 67% accuracy, which is a fantastic result because given complexity of stock market. Unfortunately there are companies, which stock price movement is much harder to predict. In some cases, model (XGB) accuracy was below 40%. This points to conclusion, that representation of stock market state is indeed very complex, so it is hardly possible to predict price movements with a very high rate of success.

It also tells us that for every company stock price behaves differently. It is because companies are from a wide variety of industries and each industry has its factors which influence it. Of course there might be some events, for example COVID outbreak, that change stock market in the same way, but it is a very rare situation. Of course such a high results in accuracy is thanks to data imbalance as for some companies there are more "positive" classes (about 75%) than "negative" and for other there more "negative" (about 65%) than "positive".

Figure 6.5: Compare of average accuracy for all companies for models which perform best in our previous experiments. Here we can see that all of them have similar results

6.3.2 Difference between Max/Min Swing Predictions

When we try to trade, we need to know whether we should buy new stocks, sell from current portfolio or just wait for a better moment. Earlier we explained that binary classification gives much better scores. That is why we predict swings for both maximum and minimum value of a price. It is very important to know if there is a difference in accuracy of predicting these two. That is why we performed few tests for the same parameters and time horizon either for rise or fall. In first test we checked the movement of price in next hour given data from last 5 minutes.

Table 6.4: Difference between average results of swing min and max for all models

Model	Type	Accuracy (%)	Precision(%)	Recall(%)
Logistic Regression	swing max	55.1	45.5	54.1
	swing min	54.8	44.9	54,7
XGB	swing max	53.3	48.7	48.9
	swing min	53.1	48.5	48.9
Decsion Trees	swing max	54.2	47.9	49.8
	swing min	53.9	48.0	49.9
MLP	swing max	52.9	37.3	54.2
	swing min	52.6	37.7	53.8
LSTM	swing max	55.5	49.5	48.8
	swing min	55.7	49.3	49.2

From Table 6.4 we can see that there is nearly no difference in predicting rise or fall in price. Of course, results are not the same, because of the character of the data, but we can generalize, that models' performances are very similar.

6.3.3 Our Best Model

We choose the best model based on three statistics: accuracy, precision and recall. The most important feature of model for us is to distinguish both classes. We can deduct it from recall lines for every company. We look at this by looking at number of boundary values. The less of them the better, because value of recall near one means, that model predict nearly always only "positive" class, and value near zero means the opposite. Then we need to look at precision and accuracy of model. As a traders, we should look mostly at precision line, because it tells us how many times we chose correctly to buy/sell assets which is crucial, because these operations define our profit or loss.

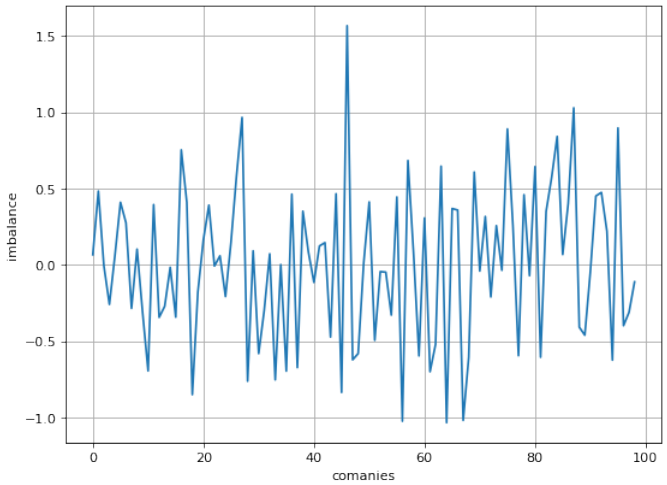

Figure 6.6: Imbalance between positive and negative samples for every **FTSE100** company. Imbalance is calculated as $\log_2 \frac{postive}{negative}$ samples, with *threshold* set to 0.00167 and setup from 5.2.6. Despite average over all comps is close to 0, one third of all companies is heavily one sided, as $|imbalance| \geq 0.5$. Therefore selecting model and hyperparameters structure that performs in good manner no matter the company, seems impossible

Based on our tests, it turns out that three models: XGBoost, Decision Trees and LSTM have the best statistics for our task. It is because they generalize our data very well and based on the fact that stock market is very chaotic and nearly unpredictable, they find some features which help to make both classes distinct. As they have very similar results, we propose to create an ensemble model, where three models have distinct prediction and then thanks to voting rule, model choose the output.

6.3.4 Model for All of FTSE100

An experimental approach would be to create one model with purpose of applying it on any company. We do so by simply calculating microstructure parameters for every company, then combine them into one larger data set. Guaranteeing we scaled each parameter per company, every one of them lies $\in [-1, 1]$. Such action will provide with much more data we could use to properly train and test more complex structures, like neural networks.

Splitting Data Set for MLP and LSTM

Using all the data together gave us total of $\approx 480k$ samples. We used $6 : 2 : 2$ split for MLP and LSTM training, applied on every company separately. That corresponds to $282k$ of train samples, $94k$ validation and test samples each.

Model with 7 hidden layers up to 4096 neurons wide resulted with 54% precision on test set. Training looked as in Figure 6.5:

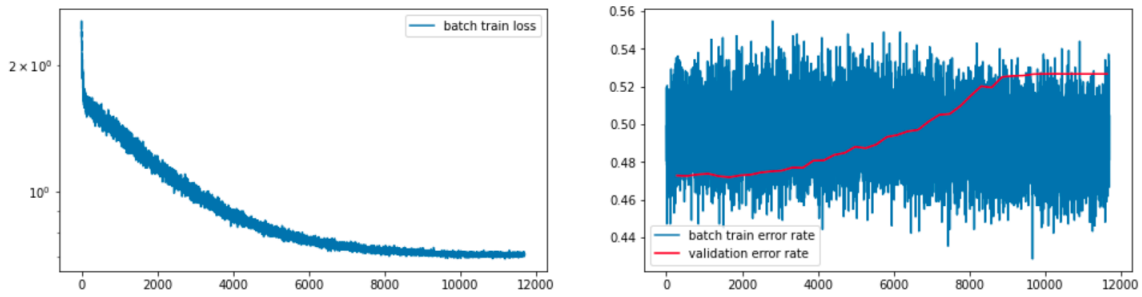

Figure 6.7: Example of one MLP trained, validated and tested (6:2:2 split) on combined data from all **FTSE100** companies. Batches consisted of 1024 samples, therefore one full epoch equals ≈ 300 iterations as we have almost $300k$ samples in train set. Both plot's x-axis represents iteration count. Left plot shows value of batches loss function steadily decreasing to ≈ 0.7. Right represents batches and validation set error rate during training. As validation error is rising as iteration go onward, it is most likely the model is underfitting. Such behaviour is reasonable, as stock market data is already very complex, when additionally various stocks are put together, model just cannot cope with it

Table 6.5: Accuracy, precision and recall on test set of MLP working on data from all of **FTSE100**. Data was divided into three sets, train, validation and test with 6:2:2 manner per each company. Problem the model was supposed to solve alongside specifications of calculation is described in Section 5.2

Accuracy (%)	Precision(%)	Recall(%)
53.93	46.74	08.56

Model quite quickly (about 20 epochs, even faster for larger learning rate) converges into what looks like local optimum. It could be observed on Figure 6.7. After that it's performance does not improve, more training is a waste of time and computing power. Most samples are classified as one class, either positive or negative depending on randomness of weights initialization. In case of MLP which training is shown on Figure 6.7, more than 90% were classified as negative. Thanks to recall being less than 10% (Table 6.5) we know such predictions are hugely incorrect.

There are some explanations of that incident. Most probably, combination of parameters from various companies had such an effect. Every company's specific microstructure usually does not correspond to others. Therefore same feature values could indicate stock price rising for one company in following hours, while foreseeing downfall of another. This is highly likely to confuse the model. Further cause is that mixing already very complex data was simply to much for a simple MLP. Above example persuades us not to train one model for all companies, although the fact that we have much more data accessible.

Chapter 7

Conclusions

We showed that it is possible to at least partially represent current market state using mentioned microstructure parameters and that they have a great impact on stock prices. Each of many experiments prove that fact. Of course not every model can solve this problem, as it was shown that Logistic Regression and MLP have much worse results than LSTM, XGBoost or Decision Trees. The cause might be, that our data set is heavily imbalanced for some companies, alongside **LOB** and trades information being available from only 10 trading days. Moreover, every company's stock price is influenced by different factors, which impact it in many ways. That is why it is impossible to learn model correctly for every company. What is more, the complexity of stock market problems is very high, which ensure us that our results are indeed satisfactory.

We have presented that our representation of market state is much better than using stock price as only guidance, which was one of key points. Models created with naive technique had similar problems, of being unable to distinguish positive and negative samples. Using wide range of market mircostructure parameters largely improved their performance, as expected. It can be seen mostly on recall line plots, where adding parameters made the values much further from limit values, which means that models started to distinguish much better both classes. What is more, precision statistic also went up. As for XGBoost and Decision Trees results were slightly better (about 0.2%), but much complex models, like MLP or LSTM made much more progress. As MLP still did not work as well as we would like to, it still made progress, because precision result was about 14% better. Nevertheless LSTM started to learn dependencies in our data and its average precision score became 21% better.

Thanks to the use of machine learning and deep learning models, such as Logistic Regression, XGBoost, Decision Trees, MLP and LSTM, we were able to predict if stock price will rise or fall during next hour by $\approx 0.15\%$. We managed to get up to 65% accuracy while being able to detect both negative and positive samples. Therefore we consider the main goal accomplished and think it is enough to consider further research. It turned out, that XGBoost, Decision Trees and LSTM have best results and we suggest to use ensemble method based on voting by these three models. Of course, every reader may look differently on results and pick differently best model, but in our opinion that is the best decision.

Beyond this, we did manage to separate less impactful microstructure elements. Those being parameters regarding order sizes and information about first trade executed in every checked time interval. Discarding them from data set supplied to models did not influence their performance negatively whilst improving time needed to train them.

Further work could focus on tuning our calculations so that predictions would be made on a specific company only. Such an approach could put more attention on particular behaviour of stock, therefore eliminating problem of searching for golden mean. As every company works differently, selecting market hyperparameters like *interval, threshold* more carefully would likely result in much more precise predictions.

More knowledge could be gained, if our calculations were applied to much bigger data set. Cited papers describe work performed on data spanned over no less than 3 months. More data would surely help with our problem of imbalanced data set. Most importantly, remembering that we had 2 work weeks of **LOB**, it would be much easier to develop and train very data hungry models, like neural networks.

Another way to go, would be return to our first approach, described in Chapter 4. Prediction of price movement after some short time interval, if correct, could be very useful when it comes to high frequency trading. Looking at longer time horizons should not be a much different, after cautious picking of used market hyperparameters or length of kept moving average.

Appendix A

Repository of codes

To this paper, we include zipped folder called "Thesis_codes", where we can find 3 files, which contain all of our calculations and plots of our engineering thesis.

First one called "Discovering_market_microstructure_parameters.ipynb" presents implementation of market microstructure parameters and show dependencies between them.

Second one called "Experiments with ML models and LSTM.ipynb" presents implementation of Machine Learning models and LSTM as well as experiences described in Chapter 5 and 6 in order to find optimal set of parameters in predicting maximal swing on price.

Last one called "skipping_params_MLP_and_th_selection.ipynb" presents process of choosing optimal market state hyperparameters, finding optimal set of market mirostructure parameters and experiences described in Chapter 5 and 6 of MLP model.

We used few python libraries such as: NumPy (Version 1.20.2), Pandas (Version 1.2.3), Matplotlib (Version 3.2.2), Pytorch (Version 1.10.1) and Tensorflow (Version 2.7.0).

Bibliography

[1] A. Briola, J. Turiel, T. Aste : *Deep Learning Modelling of the Limit Order Book: a comparative perspective*, 2020.

[2] G. Burghardt, J. Hanweck, L. Lei : *Measuring market impact and liquidity*, 2016.

[3] M. Kearns, Y. Nevmyvaka : *Machine Learning for Market Microstructure and High Frequency Trading*, 2018.

[4] J. Fernandez-Tapia : *Modeling, optimization and estimation for the on-line control of trading algorithms in limit-order markets*

[5] D. Hosmer, S. Lemeshow, R. Sturdivant : *Applied Logistic Regression*

[6] C. Bishop : *Pattern Recognition and Machine Learning*

[7] T. Chen, C. Guestrin : *XGBoost: A Scalable Tree Boosting System*

[8] I. Goodfellow, Y. Bengio, A. Courville : *Deep Learning*

[9] Z. Shi, J. Cartlidge : *Deep LearningThe Limit Order Book Recreation Model (LOBRM): An Extended Analysis*, 2021

[10] M. Dramee : *Limit Order Book (LOB) shape modeling in presence of heterogeneously informed market participants*, 2020

[11] M. Avellaneda, S. Stoikov : *High-frequency trading in a limit order book*, 2008