Feature Selection for Classifying Patterns from Ultra-High Frequency Time Series

(Wybór atrybutów do klasyfikacji wzorców z szeregów czasowych ultra-wysokiej częstotliwości)

Weronika Sikorska

Praca magisterska

Promotor: dr hab. Piotr Wnuk-Lipiński

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

Abstract

Feature crafting is an important part of process of designing solution to financial problem of price prediction. We focus on feature selection from Ultra-High Frequency Time Series by introducing Gaussian Density Filter (GDF) in order to extract features from Limit Order Book (LOB). We verify if extracted features can be useful as an input to various Machine Learning algorithms such as Logistic Regression, Support Vector Machine (SVM), Multilayer Perceptron (MLP), Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) to predict the direction of movement of stock market price. We show that Machine Learning algorithms are suitable for stock market price forecasting problem.

Wybór atrybutów jest istotną częścią rozwiązania problemu finansowego predykcji cen na giełdzie. W tej pracy skupiamy się na wyborze cech z szeregów czasowych ultra-wysokiej częstotliwości poprzez wprowadzenie GDF (ang. Gaussian Density Filter) by wyekstrachować atrybuty z Arkusza Zleceń (LOB). Badamy czy wyodrębnione atrybuty mogą być wykorzytane do algorytmów uczenia maszynowego takich jak: Regresja Logistyczna, Maszyna Wektorów Nośnych (SVM), Perceptron Wielowarstwowy (MLP), LSTM (ang. Long Short-Term Memory) oraz GRU (ang. Gated Recurrent Unit) by przewidywać dynamikę cen na giełdzie. Wykazujemy, że algorytmy uczenia maszynowego mogą być zastosowane do rozwiązania problemu predykcji cen.

Contents

Chapter 1

Introduction

Economic sector is a vital aspect of every developed society, therefore financial authorities have incentive to monitor the stock market data and keep detailed records of the stock changes [Gould et al., 2010]. The stock market is highly unpredictable, because of the amount of factors which may influence it – starting from the state of the global economy, through social media, economic news announcements [Engle et al., 2012] to even the mood of the people who buy stocks.

Thanks to the technological progress we are able to process large amounts of data with greater performance. The capabilities of modern computers make it is possible to gather and store information with ultra-high frequencies and analyze it efficiently. Advancement in this field is propelled not only by inventions increasing computational power and storage capacity, but also by new techniques originating from computer science research [Gould et al., 2010]. Nowadays it is possible to apply powerful Machine Learning algorithms for solving more complex problems.

One of the most analyzed examples of ultra-high frequency time series data is the stock market [Andersen, 2000]. Development of technology allows the electronic trading on the stock market to be performed at more frequent rates. The High Frequency Trading has increased in financial markets over the last ten years and is responsible for roughly half of trading volumes in US and Europe [Gerig, 2012]. The technological progress in this field also makes possible to design automated trading systems like Limit Order Books [Gould et al., 2010]. According to [Engle et al., 2012] Limit Order Books are rich source of information that can be used to predict micro behaviour of liquidity or volatility of the stock market.

Ability to predict stock market price dynamics could potentially yield a great profit, therefore it is understandable that this topic attracted much attention from researchers. One of the interesting questions asked was if it is possible to forecast behaviour of stock market prices and if yes, then how accurate this prediction could become?

There are many angles from which the market price dynamics was studied, for

instance statistical [Ntakaris et al., 2018], economic [Engle et al., 2012] or Machine Learning perspective [Gould and Bonart, 2015]. Recently the Machine Learning approach started to be highly promising as it has been successfully used for modeling of the stock market [Ntakaris et al., 2018]. The breakthrough in forecasting time series was done by inventing Machine Learning algorithm SVM [Adhikari and Agrawal, 2013] by [Cortes and Vapnik, 1995]. The present day Machine Learning offers also other algorithms especially designed to handle sequenced data like time series, for instance: GRU [Chung et al., 2014]. According to [Engle et al., 2012] the models which can make use of high frequency data have ability to extract useful information from Limit Order Book. That makes Machine Learning reasonable approach for stock market price prediction.

Price movement prediction is problematic for several reasons – one of them is because the data is often unstructured [Nousi et al., 2018]. It is also non trivial to tell which features are relevant for the prediction. In [Ntakaris et al., 2018] authors extract over 270 features and still it does not seem that it is enough. According to [Ntakaris et al., 2018], the majority of the research is focused on selecting classification algorithms, while less attention is given to features of the stock market data.

In this thesis we will describe how Machine Learning algorithms can be applied for solving a financial problem of price prediction based on data from stock markets which use Limit Order Books. We will focus on feature selection, moreover we will verify if extracted features can be useful as inputs to various Machine Learning algorithms for predicting the direction of price movement. We will revisit methods used by other researchers and introduce a new way of representing Limit Order Books. We will show that feature crafting is an important part of process of designing solution to such problems.

We will start from implementing approach presented in [Gould and Bonart, 2015] where the authors extract a measure of buying and selling pressure from Limit Order Book and apply Logistic Regression in order to predict the direction of stock market price. We will try more complex Machine Learning algorithms on that feature. We will also present a way to extract useful information from the whole Limit Order Book by introducing Gaussian Density Filter. The filter is necessary for eliminating the noise, which is ingrained in ultra-high time series [Engle et al., 2012]. We will use various Machine Learning algorithms on extracted features starting from Logistic Regression, through often used SVM [Cortes and Vapnik, 1995] to more novel RNN algorithms like LSTM [Hochreiter and Schmidhuber, 1997] or GRU [Chung et al., 2014]. We will show that Machine Learning algorithms are suitable for stock market price forecasting and we will compare different algorithms to choose the most suitable one.

The structure of this thesis is as follows: in Chapter 2 we describe Limit Order Book. We introduce necessary definitions, which will be used in further Chapters

of this thesis and also describe other research about forecasting stock market prices based on Limit Order Book.

In Chapter 3 we describe algorithms, tools and techniques used in this thesis. We start from describing what is time series data. We present a few supervised learning algorithms like Logistic Regression, Support Vector Machines (SVM), Multilayer Perceptron (MLP) and Long-Short Term Memory (LSTM). We discuss how to perform model selection. Next we elaborate on dimension reduction algorithm called Principal Component Analysis.

In Chapter 4 we formalize a problem of forecasting a direction of stock market price based on data from LOBs. We also describe research on that particular problem.

In Chapter 5 we detail on how we can extract information from LOBs to solve the problem. We present features selected by other researches and propose a new representation of Limit Order Book.

In Chapter 6 we show methodology used for research presented in this thesis. We detail how we compare results obtained by different classifiers and we describe which classification algorithms we use for forecasting stock market price movement.

In Chapter 7 we describe the experimental setup. We report and interpret the results. Also we detail about the data used for research.

Finally in Chapter 8 we present the conclusion of this thesis.

Chapter 2

Limit Order Book

At modern stock markets trade occurs using Limit Order Books (LOBs). LOBs are used for matching buyers and sellers orders, hence it is possible to automatically perform trading on a stock market [Gould et al., 2010]. On a stock market there are two actions which can be performed:

- *bidding* which is placing buy orders (amount of shares to buy for certain price)
- *asking* which is placing sell orders (amount of shares to sell for certain price)

2.1 Formalization of Limit Order Book

For each stock on a stock market Limit Order Book is a list of **limit orders**: *ask* and *bid* orders. *Ask* orders, called also *sell* orders, are the commitments from sellers of how many shares they wish to sell on what price. *Bid* orders, called also *buy* orders, are analogically the commitments from the buyers of how many shares they wish to buy on certain price. When using Limit Order Book buying orders are executed immediately when there are corresponding selling orders for the same price [Gould et al., 2010].

Figure 2.1: Example of Limit Order Book. We can see that all sell orders (orange bars) are above **mid price** and all buy orders (blue bars) are below **mid price**

Table 2.1: Prices and volumes from first 4 levels from exemplary Limit Order Book

	Buy Orders				Sell Orders			
	Level 4	Level 3	Level 2	Level 1	Level 1	Level 2	Level 3	Level 4
Price	688.2	688.3	688.4	688.5	688.7	688.8	688.9	689.0
Volume	4224.0	6306.0	34224.0	3332.0	3431.0	257.0	6915.0	7603.0

On Figure 2.1 we can see an example of Limit Order Book. Each bar denotes number of volumes ordered for corresponding price. In Table 2.1 there are exact values for prices and volumes for first 4 levels in the same Limit Order Book. Level denotes the position of the orders sorted by price and counted starting from **mid price**: level 1 corresponds to orders for the price closest to **mid price**, level 2 to second closest and so forth. We can see that prices between levels differ by only \$0.1, but number of volumes differs a lot – even by more than 30000 between level 1 and level 2 of buy orders.

2.1.1 List of Orders

Let us denote Limit Order Book $lob(t)$ as a list of orders $o = (p_o, v_o)$ at time t, where $p_o \in \mathbb{R}^+$ is the price of the order o and $v_o \in \mathbb{Z}$ is a volume. For buy orders volume $v > 0$ and for sell orders $v < 0$. There are no items in the order list with volume $v = 0$ [Gould et al., 2010].

2.1.2 Ask and Bid Lists

Let's denote ask list $askList(t)$ at time t as a list of orders $a = (p_a, |v_a|)$ for each order $o(t) = (p_a, v_a) \in lob(t)$ where $v_a < 0$. Ask list $askList(t)$ is a list of sell orders, but with positive number of volumes [Gould et al., 2010].

Let bid list $bidList(t)$ at time t be a list of orders $b(t) = (p_b, v_b)$ for each order $o = (p_b, v_b) \in lob(t)$ where $v_b > 0$. Bid list $bidList(t)$ is a list of buy orders [Gould et al., 2010].

2.1.3 Bid, Ask and Mid Price

Let bid price $bidPrice(t)$ be the **maximum** price in bid list $bidList(t)$ at time t [Gould et al., 2010]:

$$bidPrice(t) = \max_{\{b \in bidList(t)\}} b.p$$

Let ask price $askPrice(t)$ be the **minimum** price in ask list $askList(t)$ at time t [Gould et al., 2010]:

$$askPrice(t) = \min_{\{a \in askList(t)\}} a.p$$

Mid price is a mean of the maximum *bid price* and minimum *ask price*. It is an important measure of what is current state of stock, because orders which prices are the closest to **mid price** have the best chance of being executed in next time steps. We define mid price $midPrice(t)$ at time t as a mean of the $bidPrice(t)$ and $askPrice(t)$ [Gould et al., 2010]:

$$midPrice(t) = \frac{askPrice(t) + bidPrice(t)}{2}$$

Mid price is shown on Figure 2.2 using dashed line.

Figure 2.2: **Mid price** and Spread in Limit Order Book

2.1.4 Spread

We define spread at time t as difference between *ask* and *bid* prices from time t [Gould et al., 2010]:

$$spread(t) = askPrice(t) - bidPrice(t)$$

On Figure 2.2 we can see spread marked with red arrow, which points out two orders: order with ask and bid price. The ask price is the price of the lowest priced sell order, the bid price is the price of the highest priced buy order.

Chapter 3

Background

In this chapter we describe necessary tools used for the research presented in this thesis. We start from describing what is time series and what are the main challenges of time series analysis. Next we focus on supervised learning, where we describe classification algorithms like Logistic Regression, SVM, MLP and LSTM used for research presented in this thesis. We discuss model selection. Next we present an example of unsupervised learning algorithm – Principal Component Analysis, which is often used for reducing the dimensionality of the data [Bishop, 2006].

3.1 Time Series

The data from the stock market is recorded in time with ultra-high frequencies. In order to analyze such data first we need to understand what is time series. For time steps from 1 to t time series can be described as set of observations [Brockwell and Davis, 2002]:

$$x^{<1>}, x^{<2>}, \ldots, x^{<t>} \text{ where } x^{<i>} \in \mathbb{R} \text{ for } i = 1, \ldots, t$$

Intuitively time series is an occurring in an order set of data points measured in chronological time. Main components of time series are [Adhikari and Agrawal, 2013]:

1. *Trend*, which is a tendency to rise or decrease over longer periods of time, for instance: population growth.
2. *Cyclical variation*, which represents short cycles in the data.
3. *Seasonality*, which is long-time periodic variation, for example changes of temperature over the year are seasonal.
4. *Irregular variation* is a pattern, which is not repeated within the data and cannot be predicted.

3.1.1 Time Series Analysis

Time series analysis is used in many fields like finance, economics or chemistry. In this thesis we will analyze stock market time series data. The main goal of analysing time series is drawing inferences from past data [Brockwell and Davis, 2002]. To achieve that we develop a model which characterizes time series using the prior observations. One of applications of time series analysis is to forecast the future based on the past data [Brockwell and Davis, 2002]. The future values of variable we wish to foresee can be forecast using past values of this variable or using past values of other variables [Adhikari and Agrawal, 2013]. Common Machine Learning models used for forecasting time series are:

S.1 Autoaggressive Integrated Moving Average (ARIMA) [Brockwell and Davis, 2002]

S.2 Artificial Neural Networks (ANN) [Adhikari and Agrawal, 2013]

S.3 Support Vector Machines (SVM) [Adhikari and Agrawal, 2013]

ARIMA model **S.1** is suitable for time series which are linear and follow normal distribution [Brockwell and Davis, 2002]. Mostly it is used on univariate and *stationary* time series (the joint statistical distribution of such time series does not change over time). Considering these limitations it will not be used in this thesis.

For non-linear time series modeling, without any assumption over the distribution of the time series ANN or SVM models are more suitable. There are a lot of different ANN models used for non-linear time series modeling, like Multilayer Perceptron (MLP) [Adhikari and Agrawal, 2013]. Nowadays also Recurrent Neural Networks are used for this application [Fischer and Krauss, 2018].

3.2 Classification Algorithms

Classification algorithms are commonly used for pattern recognition in economy, biology, chemistry and many other fields. The goal of classification algorithms is to predict qualitative categories (also called *classes* or *labels*) for observations (also called input or feature vectors). For d-dimensional feature vector $\boldsymbol{x} \in \mathbb{R}$ we want to assign one of k discrete classes $C_0, C_1, \ldots, C_{k-1}$ [Bishop, 2006]. In this thesis we will focus on *binary* classification, therefore we will assign each sample to one of two classes C_0 or C_1. To be able to assign labels on unseen data, we train an algorithm on training set which consists of feature vectors with already assigned labels. After algorithm is trained, it can be applied to new observations and predict categories for them [Hastie et al., 2001]. In this section we briefly describe algorithms used in this thesis: Logistic Regression, SVM, MLP, and RNN (LSTM and GRU).

3.2.1 Logistic Regression

Logistic Regression is a Probabilistic Discriminative model used for solving classification problems. It assigns probability that observation belongs to a class for each class from the problem. That is why it is called regression despite being more suitable for classification [Bishop, 2006].

For feature vector $\boldsymbol{x} = (\boldsymbol{x_1 x_2 \dots x_n})$ where $\boldsymbol{x}_i$ for $i = 1, 2, \dots, n$ is m-dimensional sample vector and n-dimensional label vector $\boldsymbol{y} = (y_1 y_2 \dots y_n)$ for all $i = 1, 2, \dots, n$, $y_i \in C_0, C_1$ where n is the number of samples in the training set, we say that probabilities of classes C_0 and C_1 are:

$$p(C_0|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$
$$p(C_1|\boldsymbol{x}) = 1 - p(C_0|\boldsymbol{x})$$

for m-dimensional parameter vector $\boldsymbol{w}$ and for *sigmoid* function (Appendix A.1.2).

The likelihood function is:

$$p(y|\boldsymbol{w}) = \prod_{i=1}^{n} \tilde{y}_i^{y_i} (1 - \tilde{y}_i)^{1-y_i}$$

where $\tilde{y}_i$ is a prediction of a class: $\tilde{y}_i = p(C_0|\boldsymbol{x}_i)$ [Bishop, 2006].

The error function is defined as negative natural logarithm of likelihood, which is often called *crossentropy function* is given by [Bishop, 2006]:

$$E(\boldsymbol{w}) = -\ln(p(\tilde{y}|\boldsymbol{w})) = -\sum_{i=1}^{n} \tilde{y}_i \ln(y_n) + (1 - \tilde{y}_n) \ln(1 - y_n)$$

During training the goal is to calculate the parameters of m-dimensional vector $\boldsymbol{w}$, so it minimizes the error function E. Then we can use function $f(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{X})$ for predicting labels for unseen data. This minimization problem can be solved using the Gradient Descent Algorithm.

3.2.1.1 Imbalanced Classes

When classes are imbalanced we can use modified version of error function. In case of binary classification problem we use two weights c^+ and c^- [King and Zeng, 2001]:

$$E(w) = -\sum_{i=1}^{n} c^+ \tilde{y}_i \ln(y_n) + c^- (1 - \tilde{y}_n) \ln(1 - y_n)$$

We calculate weights c^+ and c^- as follows:

$$c^+ = \frac{n}{2P}$$
$$c^- = \frac{n}{2F}$$

where n is number of samples in training set, P is number of samples with assigned class C_1 and F is number of samples belonging to class C_0.

3.2.2 Support Vector Machines

Support Vector Machines (SVM) are widely used for classification, regression or anomaly detection. The classification by SVM is done by finding a hyperplane which has the largest margin from the observations as shown on Figure 3.1. This hyperplane divides input space into decision regions [Bishop, 2006]. The samples which are on one side of the hyperplane will be classified as belonging to different class than the samples on the other side of the hyperplane. The boundaries of decision regions are called decision boundaries [Bishop, 2006]. The aim of SVM is to find the most optimal decision boundary. Because of that goal SVM is famous from its good generalization properties.

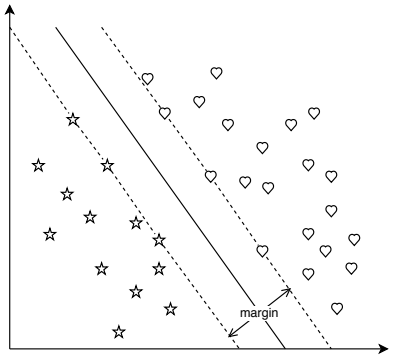

Figure 3.1: Example of margin for data which is linearly separable

Support vectors are these observations, which are the closest to the decision boundary. The maximal margin hyperplane depends on them directly.

For the feature mapping $\phi(\boldsymbol{x})$ which maps input vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ to their corresponding labels $y_1, y_2, \ldots, y_n$, where n is number of samples, and where labels are either -1 or 1, the distance from a point $\boldsymbol{x}_i$ for $i = 1, 2, \ldots, n$ to the decision hyperplane is given by:

$$\frac{y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b)}{\|\boldsymbol{w}\|}$$

where $\boldsymbol{w}$ is a parameter vector, which SVM learns during training.

The closest points to the margin (support vectors) set the position of the margin. The goal is to maximize the margin with penalization of the points which lie on a wrong side of the margin [Bishop, 2006]. We introduce set of *slack variables* $\boldsymbol{\xi} = \{\xi_1, \xi_2, \ldots, \xi_n$ for penalization and parameter $C > 0$ which controls the trade-off between margin and penalty [Cortes and Vapnik, 1995]. Hence, we minimize:

$$\arg\min_{b,\boldsymbol{w}} C \sum_{i=1}^{n} \xi_i + \frac{1}{2} \|\boldsymbol{w}\|^2$$

subject to constraints:

$$y_i(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i \text{ for } i = 1, 2, \ldots, n$$
$$\xi_i \geq 0 \text{ for } i = 1, 2, \ldots, n$$

It is a minimization problem with inequality constraints, hence we can use Lagrange Multipliers with Karush-Kuhn-Tucker conditions to solve it [Bishop, 2006]. We introduce of Lagrange Multipliers $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, \ldots \lambda_n\}$ and $\boldsymbol{\mu} = \{\mu_1, \mu_2, \ldots \mu_n\}$ (Appendix A.3):

$$L(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \lambda_i (y_i(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b) - 1 + \xi_i) - \sum_{i=1}^{n} \mu_i \xi_i$$

For $i = 1, 2, \ldots, n$ for the Lagrangian above the Karush-Kuhn-Tucker conditions are:

$$\lambda_i \geq 0$$
$$y_i(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b) - 1 + \xi_i \geq 0$$
$$\lambda_i (y_i(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b) - 1 + \xi_i) = 0$$
$$\mu_i \geq 0$$
$$\xi_i \geq 0$$
$$\mu_i \xi_i = 0$$

After optimizing out $\boldsymbol{w}, b$ and $\boldsymbol{\xi}$ we obtain dual Lagrangian form in which we optimize [Bishop, 2006]:

$$\tilde{L}(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

subject to:

$$0 \leq \lambda_i \leq C \text{ for } i = 1, 2, \ldots n$$
$$\sum_{i=1}^{n} \lambda_i y_i = 0$$

To solve that we use quadratic programming methods [Bishop, 2006]. Function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ represents *kernel function.*

3.2.2.1 Kernels

Support Vector Machines use *kernel functions.* For feature space mapping $\phi(\boldsymbol{x})$ the *kernel function* is:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x'})$$

Commonly used kernels are [Bishop, 2006]:

- *Linear* kernel: $k(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{x}^T \boldsymbol{x'} + C$
- *Radial Basis Function* (RBF) kernel: $k(\boldsymbol{x}, \boldsymbol{x'}) = \exp\left(-\gamma \left\|\boldsymbol{x} - \boldsymbol{x'}\right\|^2\right) + C$
- *Sigmoid* kernel: $k(\boldsymbol{x}, \boldsymbol{x'}) = \tanh(\gamma \boldsymbol{x}^T \boldsymbol{x'} + \text{coef0}) + C$

We can notice that these kernels have different numbers of parameters. For the most simple *linear* kernel we need to choose only one parameter C. For *RBF* kernel we need to also select additional parameter γ. *Sigmoid* kernel requires setting up three parameters: C, γ and $coef0$.

3.2.2.2 Class Imbalance

SVM is prone to class imbalance, because each sample has same misclassification cost, what will result in shifting separating hyperplane towards the minority class [Batuwita and Palade, 2013]. To reduce effect of class imbalance one of solutions is to introduce weights C^+ and C^- to the minimization problem definition [Batuwita and Palade, 2013]. We minimize:

$$\frac{1}{2}\boldsymbol{w}^T \boldsymbol{w} + c^+ \sum_{i|y_i=C_1}^{n} \xi_i + c^- \sum_{i|y_i=C_0}^{n} \xi_i$$

subject to:

$$y_i(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0 \text{ for } i = 1, 2, \ldots, n$$

3.2.3 Multilayer Perceptron

Multilayer Perceptron (MLP) is an example of Feed-Forward Artificial Neural Network. Its basic form compromises multiple layers using *sigmoid* activation function (Appendix A.1.2), which is also used in Logistic Regression, hence it is often considered as a natural extension of Logistic Regression without the *curse of dimensionality* [Bishop, 2006]. MLP also can be faster and more compact than SVM, but the price for that is that likelihood function is no longer a convex function of model parameters [Bishop, 2006].

The main idea of MLP comes from how neurons in the human brain work. An example of a single neuron (also called node) is shown on Figure 3.2.

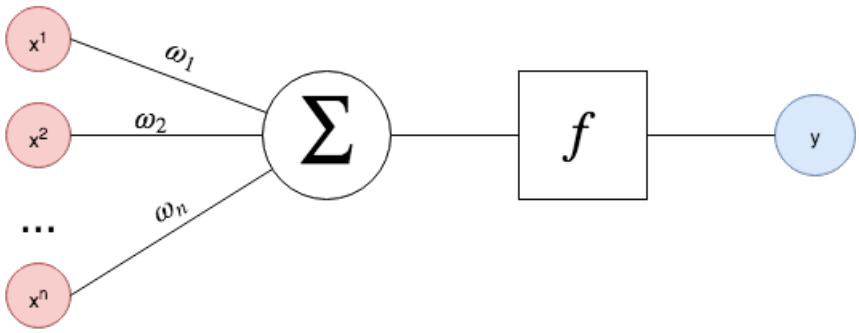

Figure 3.2: Single neuron in neural network

For n-dimensional sample vector $\boldsymbol{x}_i = [x_i^1, x_i^2, \ldots, x_i^n]^T$ and weight vector $\boldsymbol{w} = [w_1, w_2, \ldots, w_n]^T$ of weights, the output y from neuron is calculated as:

$$y = f(\boldsymbol{w}^T \boldsymbol{x}_i)$$

where f is an activation function (Appendix A.1).

The simplest MLP contains three layers of neurons: *input*, *hidden* and *output* layer. An example is shown on Figure 3.3. The input features are marked with red color, hidden layer is marked with purple color and the output with blue.

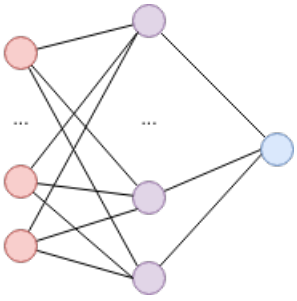

Figure 3.3: MLP network

The output of i-th neuron at d-th layer is given by:

$$y_i^{(d)} = f(h_i^{(d)}) \text{ for } h_i^{(d)} = \sum_{k=1}^{m^{(d-1)}} \boldsymbol{w}_{ik}^{(d)} y_k^{(d-1)} + \boldsymbol{w}_{i0}^{(d)}$$

where $m^{(d-1)}$ is a number of neurons in the $(d-1)$-th hidden layer. To train MLP we use **backpropagation algorithm**, which feeds the network with samples and corrects the weights $\boldsymbol{w}_{ik}$ for each layer $i = 1, \ldots, d$ and neuron $k = 1, 2, \ldots, m^{(i)}$ in order to minimize the loss function using for example Gradient Descent Algorithm.

3.2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are an extension of Feed-Forward Artificial Neural Networks designed for solving supervised learning problems on sequential data [Chung et al., 2014]. RNNs are used to solve problems on sequenced data like: speech or handwriting recognition, natural language processing, music generation or time series prediction. Basic RNN suffers from *vanishing gradient* problems, therefore it does not capture long-term dependencies well. To solve that issue in 1997 in [Hochreiter and Schmidhuber, 1997] the authors proposed Long-Short Term Memory (LSTM). In 2014 researchers suggested less complex extension of LSTMs called Gated Recurrent Unit (GRU) [Chung et al., 2014]. In this subsection we briefly describe LSTM and GRU.

3.2.4.1 LSTM

LSTM is Recurrent Neural Network, which can keep track of long term dependencies. LSTM keeps track of long term dependencies by memorizing previous hidden states. To memorize the hidden states it uses neural network units called gates with various activation functions (Appendix A.1).

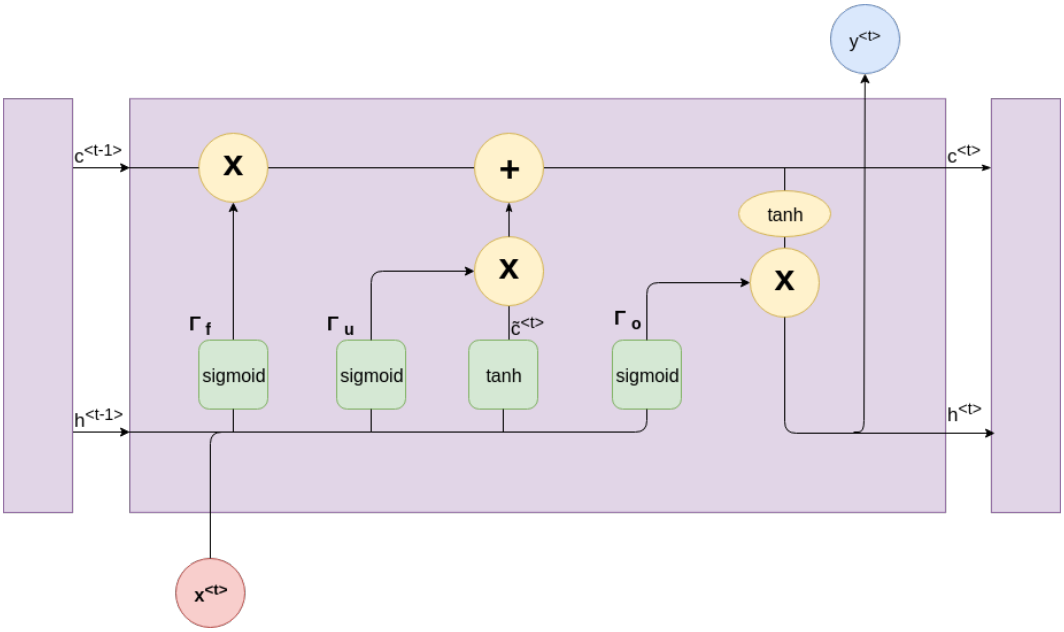

Figure 3.4: LSTM unit

Given a time series $x^{<0>}, x^{<1>}, \ldots, x^{<m>}$ where each $x^{<t>}$ is m-dimensional vector of features for t time step and $h^{<t-1>}$ is an output from the previous LSTM block at time $t-1$ (also called hidden state) the update of gates is as follows:

- update gate: $\Gamma_u = \sigma(w_u[h^{<t-1>}; x^{<t>}] + b_u)$
- forget gate: $\Gamma_f = \sigma(w_f[h^{<t-1>}; x^{<t>}] + b_f)$
- output gate: $\Gamma_o = \sigma(w_o[h^{<t-1>}; x^{<t>}] + b_o)$

Matrices w_u, w_f, w_o are weights for *update*, *forget* and *output* gates respectively. Vectors b_u, b_f, b_o are biases for *update*, *forget* and *output* gates respectively. All gates use *sigmoid* function σ as *activation* function (Appendix A.1.2).

LSTM uses hidden states h and cell states c as its memory. The new cell state memory is given by:

$$\tilde{c}^{<t>} = tanh(\omega_c[h^{<t-1>}; x^{<t>}] + b_c)$$

For each LSTM cell *forget* and *update* gates control the final cell state update, which is based on the past memory state $c^{<t-1>}$ and current memory state $\tilde{c}^{<t>}$ [1]:

$$c^{<t>} = \Gamma_u \circ \tilde{c}^{<t>} + \Gamma_f \circ c^{<t-1>}$$

[1] $\circ$ is an element-wise multiplication.

The hidden state update is based on cell state:

$$h^{<t>} = tanh(c^{<t>} \circ \Gamma_o)$$

3.2.4.2 GRU

Gated Recurrent Unit (GRU) introduced by [Chung et al., 2014] is based on similar idea as LSTM, but realizes it in simpler way, because it uses only two gates. Also it utilizes hidden state instead of cell state to transfer the memory between units.

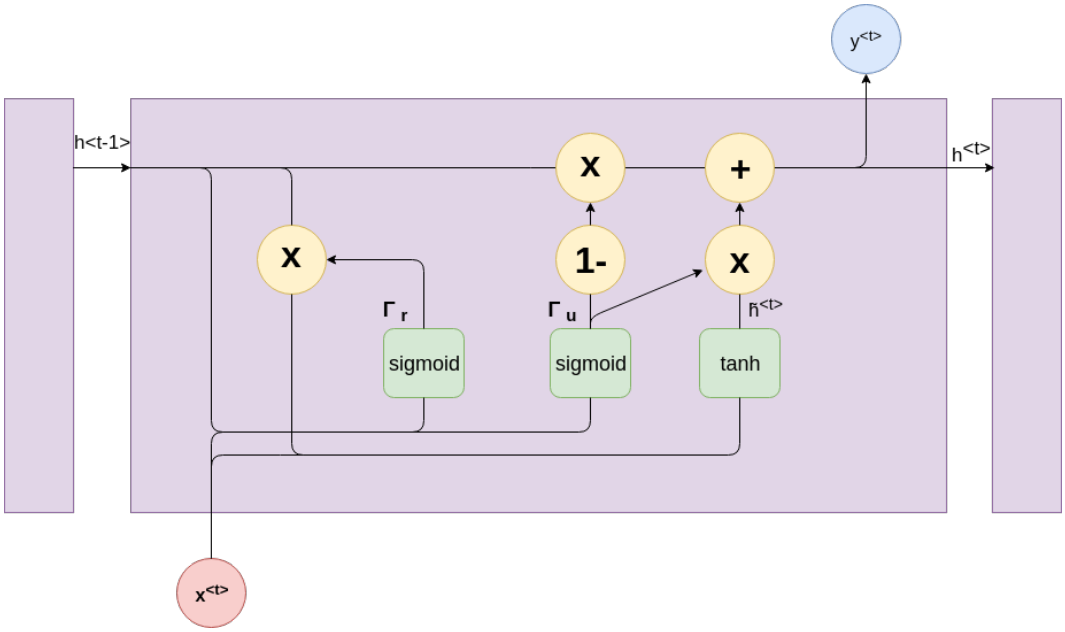

Figure 3.5: GRU unit

GRU consist of gates: *update* and *reset*. For σ is *sigmoid* function (Appendix A.1.2) the gates update works as follows:

- update gate: $\Gamma_u = \sigma(w_u[h^{<t-1>}; x^{<t>}] + b_u)$
- reset gate: $\Gamma_r = \sigma(w_r[h^{<t-1>}; x^{<t>}] + b_r)$

Matrices w_u, w_r are weights for *update* and *reset* gates respectively. Vectors b_u, b_r, b_o are biases for *update* and *reset* gates respectively. All gates use *sigmoid* function σ as *activation* function (Appendix A.1.2).

To update new hidden state h we use *tanh* activation function:

$$\tilde{h}^{<t>} = tanh(\omega_h[\Gamma_r \circ h^{<t-1>}; x^{<t>}] + b_h)$$

Final hidden state h is updated in similar way as cell state in LSTM:

$$h^{<t>} = (1 - \Gamma_u) \circ h^{<t-1>} + \Gamma_u \circ \tilde{h}^{<t>}$$

3.3 Model Selection

When solving classification problems researchers usually use multiple classification algorithms. In this section we discuss how to select the best model. First we describe how to measure performance of classifier, then we elaborate how to choose the best parameters for the classification algorithm – we specify validation method which is suitable for time series data. Next we discuss how to compare different classification algorithms using scoring methods and McNemar test.

3.3.1 Measuring Performance

There are many scoring methods for measuring performance of binary classifiers. To understand them first we need to define *confusion matrix* [Ntakaris et al., 2017]:

Table 3.1: Confusion matrix

	Actual YES	Actual NO
Predicted YES	TP	FP
Predicted NO	FN	TN

In a confusion matrix we compare predictions to actual values by calculating numbers of:

- True Positives (TP) – observations correctly predicted as "YES"
- False Positives (FP) – observations incorrectly predicted as "YES"
- False Negatives (FN) – observations incorrectly predicted as "NO"
- True Negatives (TN) – observations correctly predicted as "NO"

The most commonly used is *accuracy* [Stapor, 2017], which is defined as:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

It measures how many of the predictions were predicted correctly.

We define *precision* and *recall* as:

$$\text{precision} = \frac{TP}{TP + FP}$$
$$\text{recall} = \frac{TP}{TP + FN}$$

F1 score is a harmonic average of recall and precision:

$$\text{F1 score} = \frac{2TP}{2TP + FP + FN}$$

Also, we define Matthews Correlation Coefficient [Boughorbel et al., 2017]:

$$MCC = \frac{TP\ TN - FP\ FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Choice of which scoring method to use depends on a problem. For example when we want to classify if person has a cancer, based on RTG pictures, we prefer to have more False Positives instead of any False Negatives. On the other hand if we classify whether there is a cat on a picture, we might care only how accurate are our predictions.

3.3.1.1 ROC Area Score

Lets define True Positive Rate TPR and False Positive Rate FPR:

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

The ROC area score is the area under curve obtained by plotting TPR and FPR at different classification thresholds. The area under the curve is the probability that a random negative observation will have a smaller score than a random positive observation. It is a good indicator of how well positives are separated from negatives and it is not influenced by *class imbalance* [Bradley, 1997].

3.3.2 Validation

When a classifier needs hyper-parameter tuning we need to measure if the classifier is improving or not. Calculating scores on a training set is not a good idea, since it does not provide any information how well the classifier is generalizing on unseen

data. Therefore, it is beneficial to introduce another data set, on which we will test the performance of the classifier. We call it *validation* data set.

There are many approaches on how to chose the validation data set. The most simple one is to choose one fixed part of the training set. It is not practical, because it is not easy to make sure that this kind of choice of validation set represents the whole data set well enough (i.e. data set has the same distribution of classes and features).

The other approach is to use k-fold Cross-Validation [Bishop, 2006], in which we split data in k folds and treat one of the groups as the validation set, while training classifier on the rest of the groups. In this approach we calculate score on k validation data sets.

3.3.2.1 5-Fold Forward Cross-Validation

Regular k-fold cross-validation is not a good approach for validation for time series data, because it may loose some information about ordering of the data points. We need to make sure that we choose continuous parts of data set for validation. That is why we use Forward Cross-Validation, in which validation and training set are not shuffled. The validation set begins directly after the last sample of training set. The training set have the same length for each fold. In [Ntakaris et al., 2017] the authors proposed 10-Fold Forward Anchored Cross-Validation in which training set always starts from the beginning of the set, which means that for the bigger number of fold the bigger training set is used to train classifier. Therefore, it is harder to compare scores obtained on different folds, since they use different lengths of training set. The mean value of scores obtained that way might be biased by the scores from the last folds.

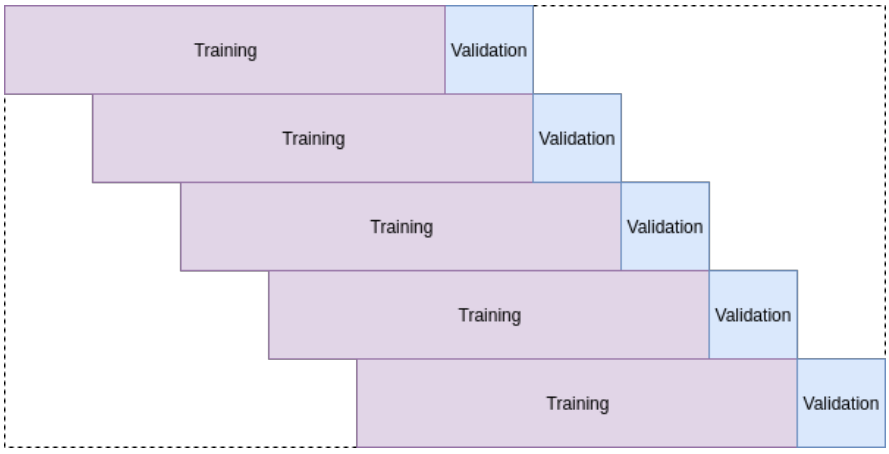

Figure 3.6: 5-Fold Forward Cross-Validation scheme

3.3.3 Comparing Classifiers

To determine which classifier (Cl_0 or Cl_1) will perform better on unseen data, we need to measure a score using one of the scoring methods mentioned in Subsection 3.3.1 on a test data set and apply McNemar test [Dietterich, 1998].

3.3.3.1 McNemar Test

The *null hypothesis* during McNemar test assumes that both classifiers Cl_0 and Cl_1 have the same error rate. To apply McNemar test for classifiers Cl_0 and Cl_1 [Dietterich, 1998]:

1. divide data into training and test data set
2. train both classifiers Cl_0 and Cl_1 on training data
3. predict labels for both classifiers Cl_0 and Cl_1 on test data set
4. create a contingency table:

$n_{0,0}$	$n_{0,1}$
$n_{1,0}$	$n_{1,1}$

   where

   - $n_{0,0}$ is a number of misclassifications by both Cl_0 and Cl_1
   - $n_{0,1}$ is a number of misclassifications by Cl_0 but not Cl_1
   - $n_{1,0}$ is a number of misclassifications by Cl_1 but not Cl_1
   - $n_{1,1}$ is a number of correct classifications by both Cl_0 and Cl_1

5. calculate $\chi^2 \approx \frac{(|n_{0,1}-n_{1,0}|-1)^2}{n_{0,1}+n1,0}$
6. calculate corresponding p-value

For McNemar test the resulting p-value is a probability that classifiers Cl_0 and Cl_1 have the same error rate. According to [Dietterich, 1998] McNemar test is the only statistical test with acceptable level of **type I** error (rejection of true null hypothesis), which does not require to run classification algorithm more than once. It is not advised by authors to view this method as rigorously correct statistical test, since it does not take into account variation in training data set.

3.4 Principal Component Analysis

Principal Component Analysis (PCA) is a technique which is used for dimensionality reduction, data visualization or data compression with loss [Bishop, 2006].

To reduce dimensionality of input set of d-dimensional vectors $\boldsymbol{x}_i$ for $i = 1, 2, \ldots, n$, we project these vectors onto m-dimensional space, where $m < d$ and we wish to keep the maximal possible variance of projected data [Bishop, 2006].

Let $\boldsymbol{u} = \{\boldsymbol{u}_1 \boldsymbol{u}_2 \ldots \boldsymbol{u}_m\}$ be a complete set of m orthonormal basis vectors in $\mathbb{R}^d$. Then any vector $\boldsymbol{x}_i \in \mathbb{R}^d$ can be represented as:

$$\boldsymbol{x}_i = \sum_{j=1}^{d} (\boldsymbol{x}_i^T \boldsymbol{u}_j) \boldsymbol{u}_j$$

We wish to approximate vector $\boldsymbol{x}_i$ using m variables, which corresponds to making a projection onto space with lower dimension [Bishop, 2006]. We approximate $\boldsymbol{x}_i$ by:

$$\tilde{\boldsymbol{x}}_i = \sum_{j=1}^{m} z_{ij} \boldsymbol{u}_j + \sum_{j=m+1}^{d} b_j \boldsymbol{u}_j$$

.

For all $j = 1, 2, \ldots, m$ we need to choose $\boldsymbol{u}_j$, z_{ij} and b_j so the error introduced by projection to lower dimension space is as small as possible. Therefore we need to minimize error subject to z_{ij} and b_j [Bishop, 2006]:

$$J = \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|^2$$

After minimization of J with respect to z_{ij} we obtain we obtain $z_{ij} = \boldsymbol{x}_i^T \boldsymbol{u}_j$ for $j = 1, \ldots, m$. After minimizing J with respect to b_j we obtain $b_j = \bar{\boldsymbol{x}}^T \boldsymbol{u}_j$ for $j = m + 1, \ldots, d$ where $\bar{\boldsymbol{x}}$ is a mean of input set $\bar{\boldsymbol{x}} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i$. After substitution of z_{ij} and b_j we obtain [Bishop, 2006]:

$$\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i = \sum_{j=m+1}^{d} ((\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T \boldsymbol{u}_j) \boldsymbol{u}_j$$

For covariance matrix of input variable given by $\boldsymbol{S} = \sum_{i=1}^{n} (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T$ we can write error function as:

$$J = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=m+1}^{d} (\boldsymbol{x}_i^T \boldsymbol{u}_j - \bar{\boldsymbol{x}}^T \boldsymbol{u}_j)^2 = \sum_{j=m+1}^{d} \boldsymbol{u}_j^T \boldsymbol{S} \boldsymbol{u}_j$$

We wish to minimize J with respect to $\boldsymbol{u}_j$ for $j = 1, 2, \ldots, d$ subject to constraints $\boldsymbol{u}_j^T \boldsymbol{u}_j = 1$, which is solved using Lagrange Multipliers (Appendix A.3). The solution leads to setting $\boldsymbol{u}_j$ to be *eigenvectors* of covariance matrix $\boldsymbol{S}$: $\boldsymbol{S}\boldsymbol{u}_j = \lambda_j \boldsymbol{u}_j$ where λ_j for $j = 1, 2, \ldots, d$ are *eigenvalues* [Bishop, 2006].

Chapter 4

Problem Definition

We want to select features from Limit Order Book for forecasting the direction of movement of stock market price. There is more than one way, in which we may indicate the movement of stock market price [Gould et al., 2010] – some researchers use opening or closing prices (for example [Fischer and Krauss, 2018]). We will construct an indicator based on mid price instead, similar as in [Gould and Bonart, 2015].

The problem we want to solve is to be able to classify if this mid price will raise in the next tick based on a current Limit Order Book. For solving that problem we need to select features from Limit Order Book, which will describe it in the most relevant way. Then we need to apply proper classification algorithm on these extracted features, so we can obtain predictions.

In this chapter we will introduce necessary definitions for solving a problem of classifying patterns in Limit Order Books, so we can predict if the mid price will raise in next time step. We define **mid price indicator**, which indicates if the **mid price** will raise in next tick. In this thesis **mid price indicator** is a target for classification algorithms for forecasting stock market price movement direction. Next we elaborate on why it is hard to extract features from LOBs. In the last section of this chapter we describe related research.

4.1 Mid Price Indicator

To represent the direction of movement of stock market price we define **mid price indicator** $midInd(t)$ at tick t:

$$midInd(t) = \begin{cases} 1 & \text{for } midPrice(t) > midPrice(t-1) \\ 0 & \text{for } midPrice(t) \leq midPrice(t-1) \end{cases}$$

Intuitively the **mid price indicator** value indicates if the *mid price* will raise in the next tick and it can be used as an indicator if the stock price will go upwards.

We will use it as a prediction target for classification algorithms.

4.2 Challenges

Before we will be able to use classification algorithms for features extracted from Limit Order Book we need to face a few challenges. The main challenge in classification of Limit Order Books is their structure. It is not practical to use them explicitly in the classification algorithms, because Limit Order Book can change size in consecutive ticks. Also, orders may change their positions in subsequent ticks, since the new orders may appear. This could affect negatively prediction ability of classifier.

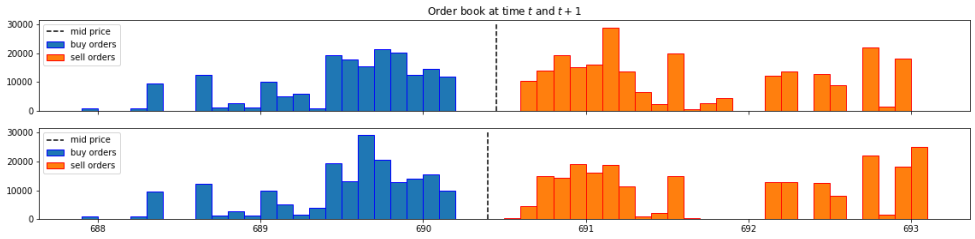

Figure 4.1: Example of Order Book on tick t (top) and $t+1$ (bottom). Values on y-axis denote *volume*, x-axis presents *price*

On Figure 4.1 we can see that in the time $t+1$ a new sell order appeared (the most left very small orange bar on a bottom plot), because of which **mid price** is lower at time $t+1$ than at time t. Also the smallest sell order at time t was partially executed, since at time $t+1$ the corresponding order has smaller volume. When using explicit representation of Order Book it would be impossible for classification algorithm to keep track of corresponding orders. Therefore it is crucial to find a way of representing Limit Order Book in such way, which will be free of these obstacles.

4.3 Related Works

In this section we describe related works which are focusing on forecasting stock market price using different ways of selecting features from Limit Order Book and various classification algorithms. The problem of stock market prediction can be attempted in many different ways.

In [Gould and Bonart, 2015] the authors introduce **queue imbalance** to predict raise of **mid price**. They use Logistic Regression for classification. They compare their results on 10 stocks from Nasdaq. They notice an improvement by about 2% when comparing against Null-Hypothesis model using ROC area score. We will use their approach as a baseline.

In [Ntakaris et al., 2018] the authors propose over 270 features, which can be extracted from Limit Order Book. They divide features into three groups: directly extracted, extracted by using Technical Analysis and by Quantitative Analysis. They use F1 score to compare results obtained using features as an input for three different classifiers.

In [Nousi et al., 2018] the authors present usage of different Machine Learning algorithms on a set of different features to predict mid price movement direction. They use handcrafted features like mid price, spread, or orders from Limit Order Book directly. To use information from the time series sequence they use window of features. They also apply autoencoders and *bag of features* algorithms to extract more features from Limit Order Book. They apply SVM and MLP algorithms to obtain predictions. The authors conclude that Machine Learning algorithms are suited for forecasting mid price movements.

The authors of [Tsantekidis et al., 2017] describe how to use Convolution Neural Networks (CNN) for forecasting direction of mid price change. They apply temporally aware (based on one previous day) standardization with z-score to normalize order lists, they input normalized order lists directly to CNN. As their target they use average of previous and next k **mid prices** instead of introducing mid price indicator based on two subsequent mid prices only. They compare their results with Linear SVM and Multilayer Perceptron (MLP). Authors notice that CNN outperforms other methods they tried.

In [Kercheval and Zhang, 2015] authors use multi-class SVM with polynomial kernel to predict the mid price movement. They use nine different sets of features for each level of the Limit Order Book, starting from the basic ones like prices and volumes to time sensitive like accelerations. They use k-fold cross validation for hyper-parameter tuning and F1, precision and recall scores for measuring the performance.

In [Ntakaris et al., 2017] authors describe benchmark data sets for forecasting **mid price** from a stock market data based on Limit Order Books. They describe the status of the current research on using Machine Learning methods. Also they propose protocol for experiments: how to measure the results and methodology of validation for hyper-parameter tuning. The authors use three different normalization methods: min-max, z-score and decimal precision normalization. There is also short overview of possible features, which can be used for further research.

Article [Li et al., 2018] introduces using feature importance in multi-input LSTM units for predicting stocks opening price for the next day.

In [Fischer and Krauss, 2018] the authors compare LSTM with logistic regression classifier, random forests and deep neural networks. They use data from the S&P 500, which contains around 25 years of daily returns for around 500 stocks. For each stock they denote input features as sequences of daily returns normalized

using standard score (they use mean and standard deviation from the training set to avoid look-ahead biases). The authors apply LSTMs using 240 steps for each sample, which consists stacked vectors of features for all stocks. They treat the problem as classification problem, where class indicates if stock outperforms cross-sectional median of return prices for all stocks in the next time step.

In [Chung and Shin, 2018] the authors use features like: low and high price, opening and closing price and trading volume to predict the closing price for the next day. They are using Genetic Algorithms to find the hyper-parameters for LSTM.

Chapter 5

Feature Selection

Feature selection is crucial in Machine Learning and the right choice of features can greatly improve the learning process. On the other hand, the wrong selection of features can even render the learning impossible [Domingos, 2012]. Therefore choice of the right set of features deserves extra time and care from researcher.

We wish to choose the representation of Limit Order Book which could be used for stock market price prediction. In order to perform **mid price** movement forecasting first we need to extract features from Limit Order Book. In this chapter we present how we select different *features* from LOB to be useful as inputs for classification algorithms.

Most of classification algorithms have a limitation of taking feature vectors of the same size as an input. Because between subsequent time steps Limit Order Books can have different sizes, it is challenging to describe it with the fixed amount of values without loosing useful information.

Finding good representation for Limit Order Book is non trivial task, mainly because there are so many different ways of selecting features from Limit Order Books. Many researchers tried to tackle this problem. For example in [Ntakaris et al., 2018] authors extracted 273 features from LOB. They split extracted features into three groups: *basic* (bid-ask spreads, mid price, price differences, mean prices and volumes, etc), features obtained using stock market *technical analysis* (for instance Accumulation Distribution Line, Highest High and Lowest Low, Variable Moving Average, etc) or *quantitative analysis* (for example autocorrelation, partial correlation or cointegration, etc). In [Kercheval and Zhang, 2015] authors used features from the basic set described in [Ntakaris et al., 2018]. In [Gould and Bonart, 2015] the authors used **queue imbalance**, which is a measure of how balanced are the closest to mid price bid and ask orders. We also use this feature as a baseline since the authors obtained very good results using Logistic Regression with this single feature as an input. In [Chung and Shin, 2018] the authors chose as features the high, low, opening and closing prices and the trading volumes. They also tried tech-

nical indicators like RSI or 10-day moving averages. In [Tsantekidis et al., 2017] the authors used Limit Order Book directly, limiting the number of orders taken into account to 10. In [Nousi et al., 2018] authors picked similar set of features as in [Kercheval and Zhang, 2015], but they also introduced *feature learning* methods in which they utilized *autoencoders* and *bag of features* for encoding feature vectors.

As there is a great deal of research using technical analysis from financial point of view, we decided to choose another path. We start from the simple single feature **queue imbalance** [Gould and Bonart, 2015] to set a baseline. Next we introduce **previous queue imbalance** to make use of the fact that there might be more information in how the **queue imbalance** changes over time. Next we propose a new set of features based on a shape of Limit Order Book, which we extract by applying Gaussian Density Filter on a normalized Limit Order Book.

In this chapter we first describe requirements for the how we would like to represent the Limit Order Book and next we introduce two approaches for feature extraction from Limit Order Book:

1. Queue Imbalance
2. Gaussian Density Filters

5.1 Limit Order Book Representation Requirements

We wish to find representation of the Limit Order Book which can be easily used as an input for classification algorithms. The requirements for such representation are:

S.1 Limit Order Book representation has to represented by the same fixed number of features for every time t.

S.2 Limit Order Book representation has to be not easily influenced by change of in the list of orders

The requirement **S.1** needs to be fulfilled, because the input to the classification algorithms has to have fixed number of *features*, what is described in Section 3.2.

The requirement **S.2** is vague, but it needs to be considered, because classification algorithms take into account the order of *features*. We do not want incoming orders to drastically change the how the subsequent Limit Order Book is represented, since it may slow down training process.

Our first approach for representing Limit Order Book is to look at buying/selling pressure based on bid and ask prices, which we will define later as **queue imbalance**. The second approach is to look at the shape of the whole Limit Order Book, we will refer to this approach as **Gaussian Density Filter** approach.

5.2 Queue Imbalance

In this section we will define a measure for buying and selling pressure called **queue imbalance**, which was also used in [Gould and Bonart, 2015].

Let's denote sum of all volumes with price p from $bidList(t)$ which we also call *total size of active buy orders* at time t for price p as $n^{bid}(p,t)$:

$$n^{bid}(p,t) = \sum_{\{b \in bidList(t), p_b = p\}} b.v$$

Let's denote sum of all volumes with price p from $askList(t)$ which we also call *total size of active sell orders* at time t for price p as $n^{ask}(p,t)$:

$$n^{ask}(p,t) = \sum_{\{a \in askList(t), p_a = p\}} a.v$$

We define **queue imbalance** $queImb(t)$ at time t as normalized difference between *total sizes of active sell and active buy orders*:

$$queImb(t) := \frac{n^{bid}(b_t,t) - n^{ask}(a_t,t)}{n^{bid}(b_t,t) + n^{ask}(a_t,t)}$$

where $a_t = askPrice(t)$ and $b_t = bidPrice(t)$ are ask and bid prices for time t respectively.

If $queImb(t) < 0$ then there are more ask orders in a LOB, so there is higher selling pressure. If $queImb(t) > 0$ then there are more bid orders in a LOB, so there is higher buying pressure. If $queImb(t)$ is close to 0 then there is no buying or selling pressure – the LOB is balanced [Gould and Bonart, 2015].

We introduce a new feature **previous queue imbalance** $prevQueImb(t)$ at time t as:

$$prevQueImb(t) = queImb(t-1)$$

The **previous queue imbalance** is a value of **queue imbalance** on a previous time step. This way we introduce information about **queue imbalance** from the past to exploit sequential nature of the data more.

We can easily see that using **queue imbalance** or both **queue imbalance** and **previous queue imbalance** fulfills the first requirement **S.1** of Limit Order Book representation, since we represent Limit Order Book with a single number (or two numbers in case of using both features).

The requirement **S.2** is also taken into account – the **queue imbalance** feature value changes only if the orders which are the closest to **mid price** are executed.

5.2.0.1 Limitations of the approach

Approach using **queue imbalance** feature only does not take into account any other information from Limit Order Book than prices and volumes for buy and ask orders closest to **mid price**. There may be more information in the Limit Order Book, which could be relevant and help to improve predictions. To look at wider picture of Limit Order Book we introduce **Gaussian Density Filter**.

5.3 Gaussian Density Filter

To be able to use more information from the whole Limit Order Book rather than the orders closest to **mid price** only we introduce Gaussian Density Filter (GDF).

As we shown before on Figure 4.1 Limit Order Book is not a regular structure in time, so tracking each order may not be the best approach for classification algorithms. Instead, intuitively we wish to track the "shape" of the Limit Order Book rather than specific orders.

Gaussian Density Filter is a filter based on Gaussian Function (Appendix A.2). Before applying it we need to prepare Limit Order Book by normalizing the values. First we describe the process of normalization, next we show the algorithm which shows an application of Gaussian Function as a GDF filter.

5.3.1 Limit Order Book Normalization

To prepare Order Book before application of GDF first we need to normalize it. To do so, we will change its order structure, so every order represents normalized price and normalized order size instead of price and volume. We define size of order $o_i(p_i, v_i)$ as:

$$size(o_i) = p_i v_i$$

We define total capital of Limit Order Book lob at time t as:

$$totalCapital(t) = \sum_{o \in lob(t)} size(o)$$

Normalized price of order $o_i(p_i, v_i)$ at time t is

$$normPrice(t, o_i) = 1000 * \frac{p_i - midPrice(t)}{midPrice(t)}$$

We choose to normalize price using **mid price** instead of applying mean normalization or standardization, because we want normalized Limit Order Book centered around **mid price**, because the orders closest to **mid price** are more likely to be

executed. Also normalization allows us to compare normalized prices across Limit Order Books from different time steps.

We can now define normalized size of order $o_i(p_i, v_i)$ at time t:

$$normSize(t, o_i) = \frac{size(o_i)}{totalCapital(t)}$$

We construct normalized Limit Order Book $nlob(t)$ from time t by normalizing each order from Limit Order Book $lob(t)$:

$$nlob(t) = \{(normPrice(t, o), normSize(t, o)) \mid \forall o \in lob(t)\}$$

5.3.2 Applying GDF

Lets define function $f(x, \mu, \sigma^2)$ as:

$$f(x, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Parameters σ^2 and μ are corresponding to *variance* and *expectation* in Gaussian Function respectively.

For every normalized Limit Order Book $nlob(t)$ at time t, which is a set of n normalized orders $no_k = (p_k, size_k)$ for $k = 1, 2, \ldots, n$ we will calculate $gdf_i(t)$ for $i = 1, 2, \ldots, K$:

$$gdf_i(t) = r \sum_{k=1}^{n} max(f(p_k, \mu_i, \sigma), no_k)$$

The expectation μ_i of Gaussian f is:

$$\mu_i = \begin{cases} -\frac{K}{2} + i \text{ if } i < \frac{K}{2} \\ i + 1 - \frac{K}{2} \text{ if } i \geq \frac{K}{2} \end{cases}$$

To apply GDF we need to choose three parameters: K, r and σ.

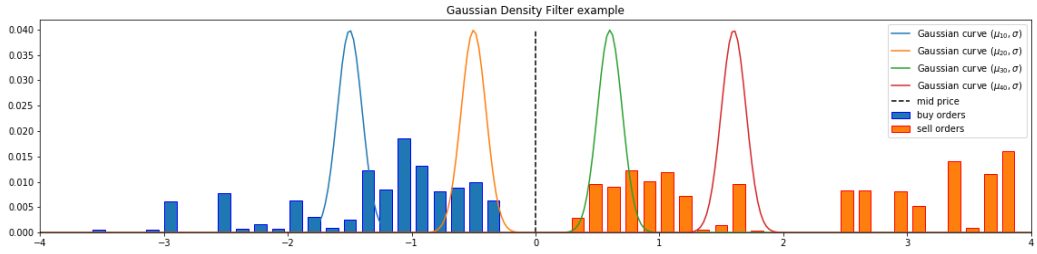

Figure 5.1: Examples of Gaussian Functions for Limit Order Book. On y-axis we denote normalized volume, on x-axis we denote normalized price. Intuitively the filter will take intersection between area under the Gaussian Curve and bars which represent the orders

We can easily see that using *gdf* features fulfill the first requirement **S.1** of Limit Order Book representation, since we represent Limit Order Book with K numbers.

The requirement **S.2** is also taken into account – by choosing parameters σ and r we can steer how sensitive the filter is.

5.3.2.1 Limitations of the approach

Applying a filter is multi-step procedure in which we first need to normalize Limit Order Book, next we need to apply the filter itself. The main obstacle in GDF feature extraction approach is choosing the initial parameters for the filter K (number of segments), σ (how wide is the filter) and r (scale, how tall is it) for data preparation. These parameters have to be chosen before performing any prediction.

This means that in total we have at least five parameters. It happens that wrong choice of feature extraction parameters K, σ and r easily leads to data set on which is very hard to find proper hyper-parameters for classifier.

5.4 Summary

In this chapter we presented our solution to the problem of selection features from Limit Order Book and also described approach used by different researchers. In addition to **queue imbalance** [Gould and Bonart, 2015], we proposed new feature called **previous queue imbalance** to be able to use the fact that stock market data is a sequence in classification algorithms, which are not designed to handle sequential data. The advantage of the **queue imbalance** is that it is a simple and yet powerful feature, which was used by other researchers with success. However it is based on orders which are the closest to **mid price** only, hence it does not utilize information from the whole Limit Order Book. That is why we proposed a new method – Gaussian Density Filters – for extracting features directly from Limit

Order Book, which allows to have more general picture of Limit Order Book. The disadvantage of this method is that it requires setting up additional parameters. However, when set up correctly, it is not prone to noise, which is an issue when using order data directly.

Chapter 6

Methodology

In the previous chapter we described how to select features from Limit Order Book. In this chapter we will describe how do we use extracted features to make a prediction of the direction of a movement of **mid price**.

Researchers use many different algorithms for making predictions of stock market prices. In [Nousi et al., 2018] authors used MLP and SVM algorithms for predicting **mid price** movements. In [Gould and Bonart, 2015] the authors were using Logistic Regression for similar purpose. In [Kercheval and Zhang, 2015] the researchers explored if SVM is suitable for capturing the dynamics of Limit Order Books. In [Tsantekidis et al., 2017] the authors decided to compare CNN with SVM and MLP on a task of predicting stock market price movement. In both [Chung and Shin, 2018] and [Li et al., 2018] the authors used LSTM. In the first work the authors proposed a method using genetic algorithms to overcome the challenge of tuning hyper-parameters of LSTM. In the second work the authors presented a novel multi-input LSTM, which is able to discard noise from the data by dual-stage attention mechanism. The benefit of our solution is that thanks to focusing more on feature selection we free ourselves from the need of applying special additions to algorithms, which reduces overall complication of the solution.

Before we describe algorithms applied for forecasting **mid price indicator** first we describe the data used for the research presented in this thesis. There is a challenge of *class imbalance* to address before using the data for prediction. Next we elaborate how we asses the predictions performed using different methods. With increasing complexity of feature space, more sophisticated classification methods are needed, therefore we use different algorithms for different sets of features. We show how we apply Logistic Regression and Support Vector Machine algorithms on **queue imbalance** to predict **mid price indicator**. Next we describe how we use Logistic Regression, MLP, LSTM and GRU on features extracted from Limit Order Book using Gaussian Density Filters.

6.1 Data

We use ultra-high frequency data from London Stock Market from 2013-09-01 – 2013-11-15 for 98 stocks. London Stock Market is open on week days between 8:00 and 16:30. We decided to skip the data before 8:30 and after 16:00, same as in [Gould and Bonart, 2015], since half an hour before closing or opening of the stock market exchange is the time when the most abnormal behaviours happen.

We will start from showing an example of raw data, then we will show **mid price** and **mid price indicator** visualizations. Next we describe how we choose stocks for this research. In the last subsection we elaborate on how we tackle problem of balancing the classes.

6.1.1 Raw Limit Order Book

We show a raw snippet from the data set of one of the stocks on Figure 6.1. For the purpose of visualization of the plain data set we needed to omit parts of the data – skipped data is marked with "...". In the data set there is one row of data per minute. Each row represents one Limit Order Book. Therefore, in each row first there is a bid list - starting from **BID** (marked by red color on Figure 6.1) - where in columns with even number $2i$ where $i = 0, 1, \ldots,$ *length of bid list* are prices and in columns with numbers $2(i + 1)$ there are corresponding volumes. We start numbering the columns from 0 from the first column which is on the right of caption **BID**. Similarly for the ask list, which starts from the cell with caption **ASK**. Notice that lengths of bid lists and ask lists can differ between Order Books at different timestamps.

2013-09-02 08:00:00	BID	550.0	10000	...	709.4	45531	709.8	2932	716.5	35755	ASK	635.5	1820	...
2013-09-02 08:01:00	BID	550.0	6364	...	688.2	4515	ASK	688.9	2730	689.0	7580	689.1	1644	...
2013-09-02 08:02:00	BID	550.0	6364	...	688.3	3774	ASK	689.0	10219	689.1	1227	689.2	2664	...
2013-09-02 08:03:00	BID	550.0	6364	...	688.5	31448	688.6	607	ASK	689.1	8918	689.6	4047	...

Figure 6.1: Order Book raw data snapshot. Prices are in light yellow cells and volumes are in light grey cells

6.1.2 Mid Price

From **mid price** plots shown on Figure 6.2 we can see that there is no obvious pattern in how **mid price** changes. Each of stocks presented on this plot belongs to a different cluster (shown on Figure 6.5). The range of **mid prices** differs for these stocks – the lower mean **mid price** the smaller range, for stock 3459 [1] the price range is between 730 and 790, for stock 9058 between 1750 and 1980 and for

[1] We refer to internal stock ids – names of stocks can be checked in Table 7.1.

stock 2748 between 4900 and 5600. For both stocks 3459 and 2748 we can see that the overall trend is rising, for 9058 it is more stable. For 2748 we can observe big jump (about 400 difference) of a **mid price** at 2013-10-30.

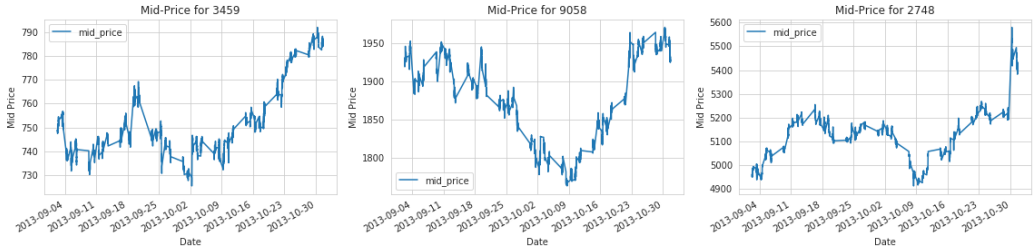

Figure 6.2: **Mid price** for three stocks between 2013-09-01 and 2013-11-01

On Figure 6.3 we can see that **mid price** distributions between stocks vary. All of the distributions are skewed, for stock 3459 it is negatively-skewed, for 9058 and 2748 it is positively-skewed.

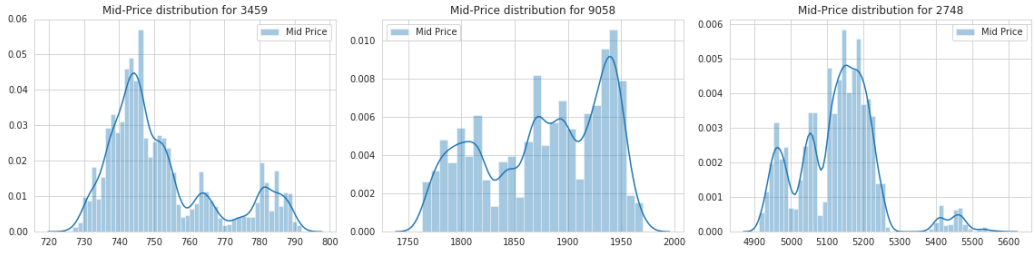

Figure 6.3: **Mid price** distribution for three stocks between 2013-09-01 and 2013-11-01

6.1.3 Mid Price Indicator

Mid price indicator corresponds to classes (also called labels) in a forecasting problem we are trying to solve. On Figure 6.4 we can see that for all exemplary stocks **mid price indicator** values are not distributed equally, which means that **mid price** raises more rarely than it is not rising. During classification we will need to balance classes, by assigning weights, an example how it is done for Logistic Regression is shown in Subsection 3.2.1.1.

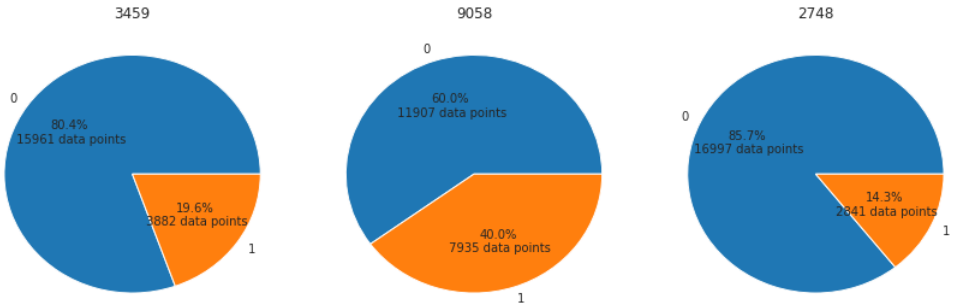

Figure 6.4: **Mid price indicator** numerical proportions for three stocks. Blue slice corresponds to **mid price indicator** equal 0 (**mid price** will not raise in next time step) and the orange one relates to **mid price indicator** equal 1 (**mid price** will raise in next time step)

6.1.4 Stocks Choice

Because we treat each stock separately, we decided to limit number of stocks used for this thesis. We applied clustering K-Means clustering on all stock data using mean **mid price** and mean spread as input. The clusters are shown on Figure 6.5. We can see that the first cluster marked with pink color is for stocks for which the mean **mid price** is below 1000 and the mean spread is below 1. For the second cluster marked with black color, the mean **mid price** is between 1000 and 2400 and the mean spread is between 0.6 and 2. For the last white cluster the mean **mid price** is between 2400 and 5128 and the mean spread is between 1 and 6. It seems that the choice of clusters depends mostly on the mean **mid price**.

Figure 6.5: Clustered stock market data based on mean **mid price** and mean spread using K-Means clustering with 3 clusters

We decided to choose 5 stocks per each cluster. The distribution of mean **mid**

price and mean spread is shown on Figure 6.6. The chosen stocks have closer to uniform distribution of the mean **mid price** and the mean spread than all stocks.

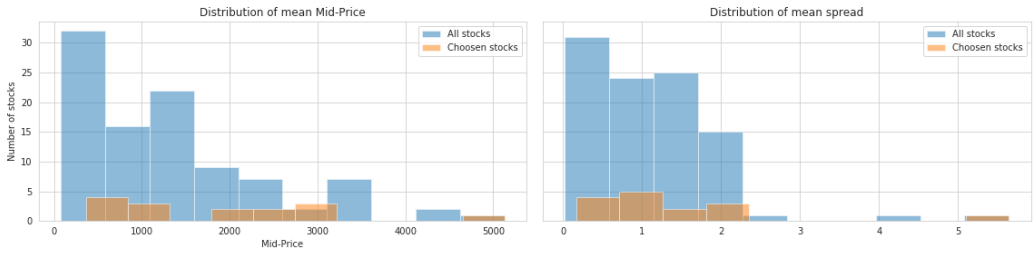

Figure 6.6: Distribution mean **mid price** and mean spread for all and chosen stocks

6.1.5 Class Balancing

As was shown on Figure 6.4 the ratio of positives to negatives of **mid price indicator** is not equal or even close to 1. Therefore, we need to balance the classes to get a meaningful predictions. Otherwise there could be that classifier which predicts that **mid price indicator** is always "0" could have for instance accuracy above 75%. We balance classes using class weights applied to loss function, which is described in Subsection 3.2.1.1, but there are other approaches to that problem as well.

To balance the classes the authors of [Gould and Bonart, 2015] adjusted the number of positive and negative samples by removing every each one of subsequent samples with the same **mid price** value. This approach removes the time dependency between subsequent samples, which makes data unsuitable as an input to any algorithm which extract information from the sequence like LSTM.

To use this approach for online learning we would need to check if the last data sample at time t had the same **mid price** as the incoming one at time $t+1$. If $midPrice(t) = midPrice(t+1)$ then we would need to just drop the same from time $t+1$.

The approach of balancing the classes in the data leads to issues with predictions on unseen data. The new data may have unbalanced classes, which also has a consequence with having different distribution of input feature – **queue imbalance**. This may cause this classifier to not generalize well on unseen data.

6.2 Assessing Predictions

To assess and compare predictions done by different classifiers, we calculate Matthews Correlation Coefficient (MCC) and ROC Area scores on testing data set. We have chosen ROC area score because it is considered one of the best measures of binary classifier performance [Bradley, 1997], but it might be prone to class imbalance, be-

cause it does not take into consideration the whole contingency table. Also there is no explicit formula to calculate it, so the interpretation is not straight-forward [Boughorbel et al., 2017]. Therefore, we also use MCC score, since it uses all measures from contingency table [Boughorbel et al., 2017] and it can be computed using one equation using all values from contingency table. After measuring and comparing scores we also apply McNemar test to check if compared classifiers have different error rates.

6.2.1 Null Hypothesis

Null hypothesis assumes that none of features we use have any influence over direction of **mid price** movement. Therefore the probability that **mid price** will raise in the next time step is exactly 0.5. The classifier which would make such predictions would achieve 0.5 ROC Area score [Gould and Bonart, 2015] and 0 using MCC score [Boughorbel et al., 2017].

6.2.2 Model Selection

To be able to evaluate predictions we need to split data for each stock into two data sets: training and testing set. On a training data set we will train classifiers. We will choose the hyper-parameters for which we got the highest score obtained during validation, if the classifier requires hyper-parameter tuning. We use 5-Fold Forward Cross-Validation method 3.6 in which validation set is chosen from a training set in a rolling window. We compare different types classifiers using scores obtained on applying them on testing set.

6.2.2.1 Training and Testing Data

In total we have around 24000 data points per stock. We use 20000 data points (80%) as training and validation data set and the rest 5000 data points (20%) as testing data set. As a training set we use data from 2013-09-02 08:30:00 to 2013-10-31 16:00:00. As a testing data we use data from about 2013-11-01 08:30:00 to 2013-11-15 16:00:00.

6.2.3 Model Comparison

For classification methods which use randomization, hence they not yield the same results every time, we run these 30 times and take a mean score obtained on testing set. Next we can compare them against each other.

6.2.4 McNemar Test

Since our data does not have a normal distribution we can apply McNemar test, which does not rely on data distribution [Stapor, 2017]. We use McNemar test to compare if two classifiers have the same error rate, which would mean that they are approximately the same. The benefit of using McNemar test is that it does not require retraining classifier multiple times, what takes much time for more complex classifiers [Dietterich, 1998].

We calculate χ^2 for contingency table obtained by comparing predictions made on testing set for both classifiers and true values as described in Subsection 3.3.3.1. If $\chi^2 > 2.706$ which corresponds to p-value equal 0.1 then the probability that classifiers have similar error rate is small enough to conclude that they are different.

6.3 Queue Imbalance

In this section we describe how we apply Logistic Regression and SVM to predict **mid price indicator** using **queue imbalance** as a feature. The approach is shown on Figure 6.7.

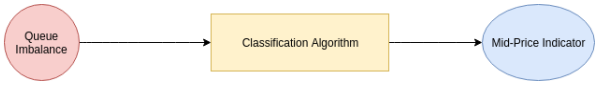

Figure 6.7: Scheme of classification algorithm for predicting **mid price indicator** using **queue imbalance**

6.3.1 Logistic Regression

To predict **mid price indicator** we train Logistic Regression on the training data, where we use **queue imbalance** as a feature. The algorithm outputs a probability p. If $p > 0.5$ then we predict that the **mid price** will raise. Otherwise we say that **mid price** will not raise. Approach with Logistic Regression on **queue imbalance** feature was also used in [Gould and Bonart, 2015]. We use this approach to set up a baseline algorithm with which we compare other algorithms.

We also tried to apply Logistic Regression algorithm on two features: **queue imbalance** and **previous queue imbalance** to use more information from the past about the **queue imbalance**.

6.3.2 SVM

To predict **mid price indicator** we train SVM on training data with **queue imbalance** as a feature. When predicting, the algorithm outputs the label **mid price**

indicator.

The main challenge in using SVM is the number of hyper-parameters, which has to be set. The most important is the choice of the kernel. We decided to try three commonly used kernels (described in Section 3.2.2.1):

1. *Linear* kernel
2. *RBF* kernel
3. *Sigmoid* kernel

For every kernel we perform hyper-parameter tuning using **grid search** approach [Syarif et al., 2016] using 5-Fold Forward Cross-Validation (described in Subsection 3.3.2.1) to find which hyper-parameters obtain the highest score during validation. As a performance metric we used Matthews Correlation Coefficient (MCC) and ROC area score. We summarized hyper-parameters for different kernels in Table 6.1. To allow grid search find the best combinations of hyper-parameters we decided to use wide range of parameters from 0.001 to 1000.0.

Table 6.1: SVM hyper-parameters for each kernel

linear	$C \in$	$\{0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0\}$
RBF	$C \in$	$\{0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0\}$
	$\gamma \in$	$\{0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0\}$
sigmoid	$C \in$	$\{0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0\}$
	$\gamma \in$	$\{0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0\}$
	coef0 $\in$	$\{0.001, 0.01, 0.1, 1.0, 10.0, 100.0\}$

We also tried to apply SVM algorithm on two features: **queue imbalance** and **previous queue imbalance** to use more information from the past about the **queue imbalance**.

6.4 Gaussian Density Filter

We select features from Limit Order Book by applying Gaussian Density Filter (GDF) with $K = 50$.

We also add **queue imbalance** to the feature vector, so for each time t we have a vector v of features: $v = [gdf_1, gdf_2, \ldots, gdf_K, queImb(t)]^T$. We reduce the dimensionality of feature vector by applying PCA. We will refer to this approach as GDF+PCA.

We use GDF+PCA features for prediction of **mid price indicator** using different algorithms. We started from Logistic Regression for which the approach is

straight-forward. The natural next step is to try more sophisticated algorithms like MLP or LSTM.

6.4.1 Choice of GDF Parameters

We have chosen GDF parameters empirically. We used grid search using parameters $r \in (0.01, 0.1, 0.25)$ and $s \in (0.1, 0.25, 0.5)$.

On Figures 6.8 and 6.9 we show examples how the features extracted using Gaussian Density Filter look like for different parameters r and σ. We can see that for bigger r and σ the GDF values are smoother.

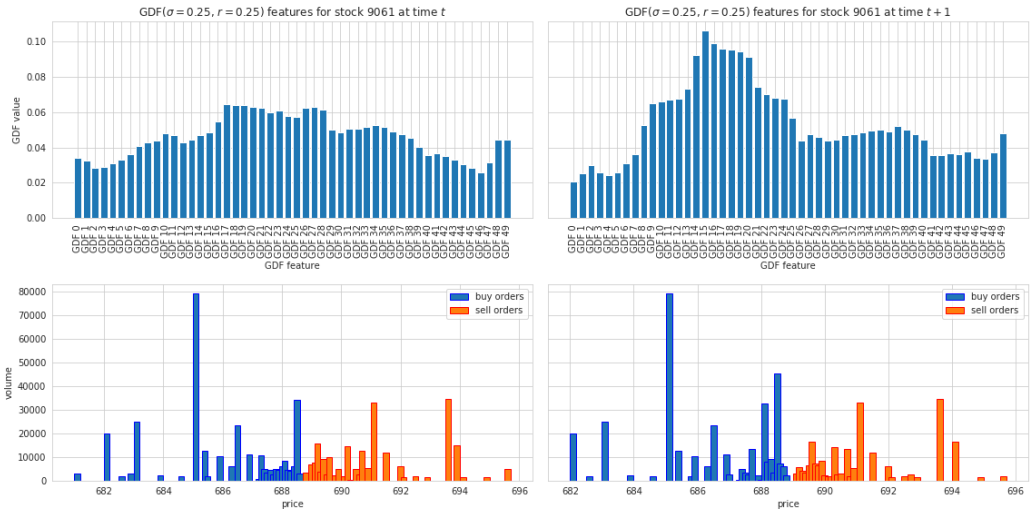

Figure 6.8: Example of GDF features(top) and corresponding Limit Order Book for $r = 0.25$ and $\sigma = 0.25$

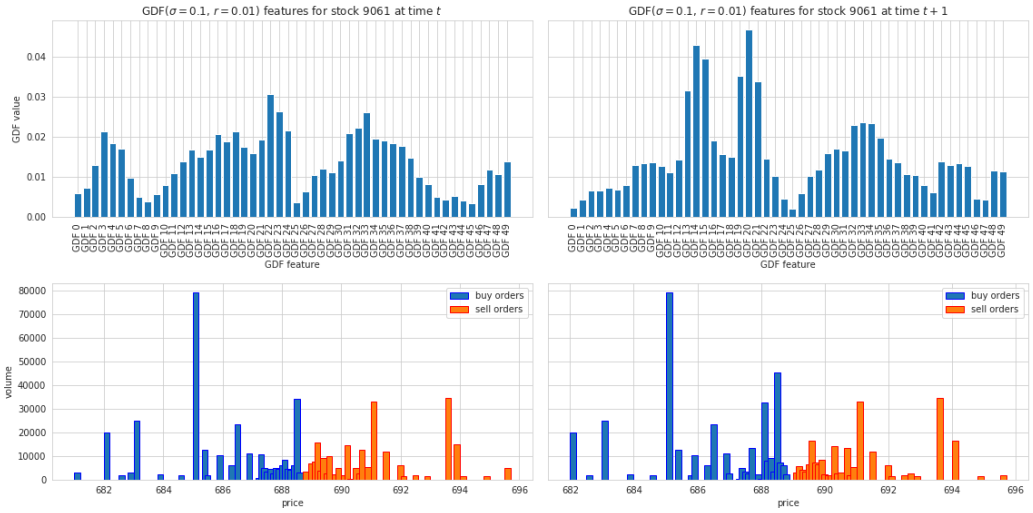

Figure 6.9: Example of GDF features(top) and corresponding Limit Order Book for $r = 0.01$ and $\sigma = 0.1$

6.4.2 Feature Number Reduction

Using all *gdf* features might not be practical, because the orders which have price far from **mid price** are not so relevant. That is because the closest orders to **mid price** have much bigger probability of being executed. We tried three approaches of choosing the features:

1. Limiting number of features
2. Principal Component Analysis applied on all *gdf* features
3. Principal Component Analysis applied on all *gdf* features and **queue imbalance**

Simple limiting number of features could work as follows. For n as a number of features we want to chose we use features:

$$gdf_{\lfloor \frac{K}{2} \rfloor - \lfloor \frac{n}{2} \rfloor}, gdf_{\lfloor \frac{K}{2} \rfloor - \lfloor \frac{n}{2} \rfloor + 1}, \ldots, gdf_{\lfloor \frac{K}{2} \rfloor + \lfloor \frac{n}{2} \rfloor}$$

On Figure 6.10 we can see general idea of applying Principal Component Analysis algorithm to limit the number of the features. We simply input all the features to PCA and use output from PCA as input for classification algorithm.

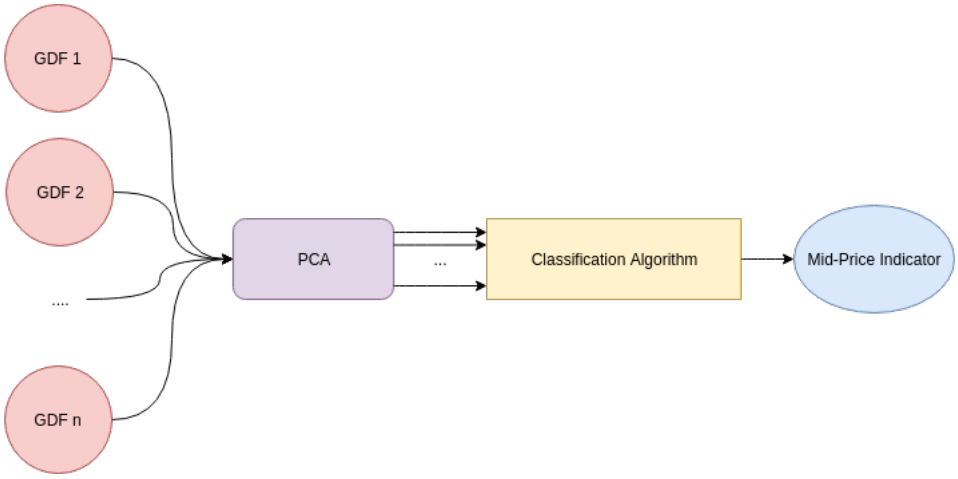

Figure 6.10: Approach with using GDF+PCA approach to predict **mid price indicator**

6.4.2.1 Choice of number of PCA components

When using PCA it is important to choose number of components, to which the data set will be reduced. We decided to use smallest number $k \leq 10$ of components for which 99% of variance is retained.

We empirically chose 10 as maximal number of PCA components, because we noticed that for most of the data sets of the stocks numbers between values 2-5 are usually sufficient.

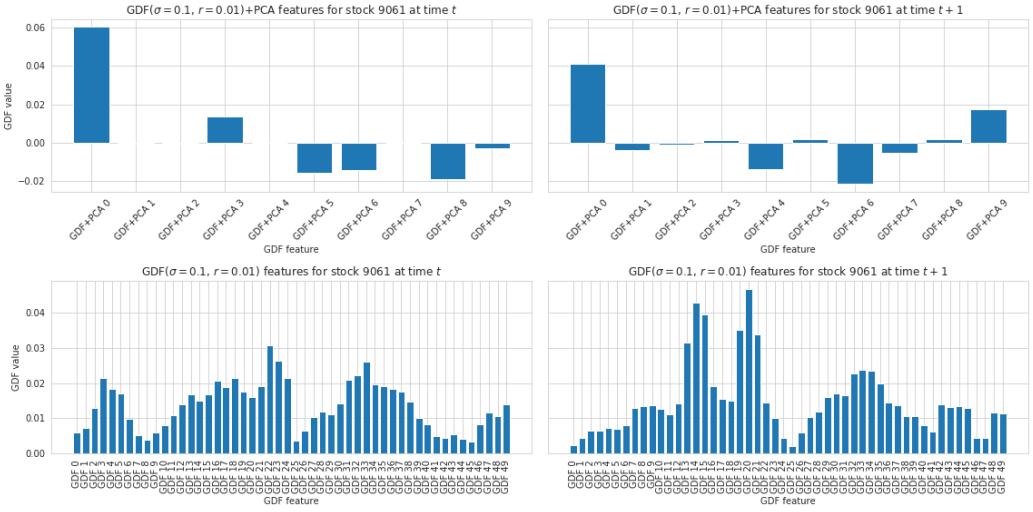

Figure 6.11: PCA-reduced GDF features at time t and $t+1$ on top plots. On bottom plots there are corresponding LOBs

6.4.3 MLP

After making a choice of which *gdf* features to use we can apply classification algorithm in order to predict **mid price indicator**. In this subsection we will describe the approach of applying MLP as classifier.

We use various architectures of MLP, with different numbers of hidden layers from one to up to three with $2, 4, 8,$ or 16 neurons. For all layers except the output layer we use *tanh* activation function (Appendix A.1.3). For output layer activation function we use *sigmoid* activation function (Appendix A.1.2). As a loss function we use weighted binary crossentropy (described in Subsection 3.2.1.1), because classes in the problem we solve are not balanced.

To reduce overfitting we use l2 regularization.

We use mini-batch gradient descent with batch size 512. We optimize it using Adaptive Moment Estimation algorithm (Adam), which stores an exponentially decaying average of past gradients and squared gradients [Ruder, 2016] for computing adaptive learning rates for each parameter.

6.4.4 LSTM and GRU

When the algorithm becomes more complex it is more challenging to prevent it from overfitting. It also has more hyper-parameters to tune. There are many techniques

which can prevent overfitting, but they also increase number of hyper-parameters. The other challenge is computation time – for complex models it takes more time to train and validate algorithm.

All RNN algorithms like LSTM or GRU are even more challenging than MLP in regarding how many parameters have to be set, starting from the architecture.

As input RNN takes n time steps with features prepared using GDF filter and reduced by PCA. From the last unit of RNN we output probability that **mid price** will not raise in next time step using *sigmoid* activation function (Appendix A.1.2).

For both LSTM and GRU in a single unit, which corresponds to one time step, we used many-to-one architecture shown on Figure 6.12. The outputs from the last RNN unit is feed to neural network with one input and output layer. The neural network outputs **mid price indicator prediction** using *sigmoid* function (Appendix A.1.2).

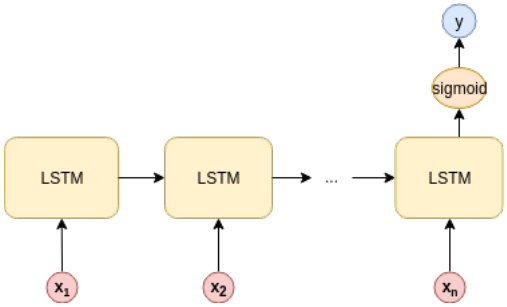

Figure 6.12: Architecture of LSTM network. $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are feature vectors in subsequent n time steps

For neural network units inside LSTM or GRU units we try different activation functions like *tanh* and *relu*. We tried different number of time-steps: 1, 2, 4, 8 and 16. We use 1, 4, 8, 16 or 32 cells in one unit.

We prefer to use mini-gradient descent instead of regular gradient descent to make algorithm converge faster, we use batch size 512 for mini gradient descent.

We also tried to use stacked RNNs with ANN, but it is more challenging to not overfit. We use techniques like dropout and l2 regularization to prevent overfitting [Srivastava et al., 2014].

6.4.4.1 Challenges in training LSTM and GRU

The main challenge in training LSTM or GRU networks is the number of parameters, which can be chosen, starting from learning rate to different activation functions applied on different components of LSTM unit. With GDF+PCA approach we have to pick r and σ parameters as well. Choosing the right architecture for RNN

networks is also challenging, since it is very easy to cause overfittig. There are several methods which can be used to prevent larger networks to overfit, but they introduce another sets of parameters as well. It is sometimes possible to learn these parameters using for instance genetic algorithms like in [Chung and Shin, 2018], but we did not use them during this research.

6.5 Summary

In this chapter we described how we use classification algorithms and extracted features during feature extraction for forecasting the direction of **mid price** movement.

We decided to start from Logistic Regression, because this algorithm is easy to train and because of its simplicity it converges quickly. For simple features like **queue imbalance** it might be even enough, but we decided to try SVM for these as well. SVM are able to capture more complex dependencies than Logistic Regression [Bishop, 2006]. For more complex features obtained by applying GDF+PCA we also started from Logistic Regression, but as Logistic Regression is known from suffering *dimensionality curse* [Bishop, 2006] we decided to try MLP. None of the algorithmic approaches utilizes the fact that the data is sequential. Of course it is possible to craft features in that manner (we try that with for instance **previous queue imbalance**), but it could be better if the algorithm could understand time dependencies. Since recently RNN algorithms are used more often for forecasting time series data we decided to use LSTM and GRU. To our best knowledge GRU was not yet used for this application.

Chapter 7

Results

In this chapter we show results obtained for features selected from Limit Order Book used in different classification algorithms for forecasting **mid price indicator** (Section 4.1). We compare differences in quality of predictions by comparing MCC and ROC area scores obtained for classifiers (Subsection 6.2.2). To determine if classifiers are different we will also execute McNemar test (Subsection 3.3.3.1).

In our experiments we use different algorithms for different sets of features, since the complexity of feature space differs between feature extraction approaches:

1. Queue Imbalance feature
    - Logistic Regression – QUE+LOG
    - SVM – QUE+SVM
2. Queue Imbalance and Previous Queue Imbalance features
    - Logistic Regression – PREVQUE+LOG
    - SVM – PREVQUE+SVM
3. GDF+PCA features
    - Logistic Regression – GDF+PCA+LOG
    - MLP – GDF+PCA+MLP
    - LSTM – GDF+PCA+LSTM
    - GRU – GDF+PCA+GRU

For algorithms which require hyper-parameter tuning (SVM, MLP, LSTM, GRU) we use 5-Fold Forward Cross-Validation (Subsection 3.3.2.1) using MCC score

in order to measure the performance of classifier with chosen hyper-parameters. We choose classifiers with hyper-parameters which give the highest MCC score during validation per stock. To measure how well classifier is generalizing we record MCC and ROC area scores obtained on unseen data (test data set). Also we use testing data set to compare chosen classifiers – for algorithms which include randomization steps (MLP, LSTM, GRU), we run each experiment 30 times and take a mean score obtained on unseen data.

7.1 Data

In Table 7.1 we show internal ids, stock market symbols, mean spread and mean **mid price** and total arrivals (total number of volumes) of stocks used for this research from 2013-09-01 – 2013-11-15. We have chosen five stocks per cluster as described in Section 6.1.4, the clusters are based on mean **mid price** and mean spread.

Table 7.1: Stocks chosen for research. Stocks are grouped based on cluster to which they belong to

id	symbol	mean spread	mean mid price	total arrivals
9061	HSB	0.167577	686.151230	$7.295451\times10^{+9}$
3459	NGG	0.646266	752.117731	$1.507767\times10^{+10}$
9761	ARM	0.939125	973.908411	$1.042543\times10^{+10}$
4851	UU	0.769056	696.995186	$9.448016\times10^{+9}$
2730	LAND	0.776235	934.176676	$9.675409\times10^{+9}$
9062	RIO	0.966438	3120.336399	$1.061201\times10^{+10}$
3879	ULVR	1.200665	2456.725912	$3.700911\times10^{+9}$
1431	SDR	2.329788	2553.911820	$2.683938\times10^{+9}$
2748	NXT	5.644103	5128.616356	$6.317837\times10^{+9}$
4320	WTB	1.890978	3150.772656	$1.526888\times10^{+9}$
2822	VED	1.901099	1102.788002	$1.414043\times10^{+9}$
11583	DGE	0.780740	1984.517965	$1.206506\times10^{+10}$
4799	FRES	1.743972	1041.757428	$6.349821\times10^{+9}$
10470	WEIR	1.805948	2314.327218	$1.805229\times10^{+9}$
9058	BLT	0.743928	1877.535943	$3.354497\times10^{+10}$

7.2 Experiments Setup and Benchmarks

Experiments are run using Python 3.6 with *sklearn* (for Logistic Regression and SVM algorithms) and *Keras* (for MLP, LSTM, GRU algorithms) libraries. We use 64-bit Fedora 29 on CPU Intel(R) Core(TM) i7-6820HQ CPU @ $2.70GHz$ with 16 GB of RAM machine to run experiments on.

Execution times are various for different approaches. Table 7.2 shows the mean time of single training for different approaches. QUE+LOG is trained in significantly less time than other algorithms, which is less than 1 second. The other approaches need on average more than 20 seconds to be trained. The training time raises with complexity of an algorithm. GDF+PCA+GRU and GDF+PCA+LSTM needs the most time – their execution times are comparable.

Table 7.2: Mean time of training (in seconds) per approach

approach	time (sec)
QUE+LOG	0.900386
QUE+SVM	22.785398
GDF+PCA+MLP	36.645407
GDF+PCA+GRU	38.455034
GDF+PCA+LSTM	38.620579

We show training times for QUE+SVM for different kernels in Table 7.3. QUE+SVM with *linear* kernel needs around 12 seconds to be trained, on the other hand QUE+SVM with *RBF* needs more than 30 seconds, which is comparable to execution time of approaches with GDF+PCA.

Table 7.3: Mean time of training (in seconds) per kernel for QUE+SVM

kernel	time (sec)
linear	12.724253
sigmoid	25.204549
RBF	30.427392

7.3 Queue Imbalance

Queue imbalance feature has statistical relationship with **mid price** movement. We can see it on Figure 7.1 – the orange *violins* are getting wider for the higher values of **queue imbalance**, what implies that the higher value of **queue imbalance**, the more often **mid price indicator** is positive, hence the more often **mid price** raises in next time step. From the location of modes of blue *violins* we can see that the lower value of **queue imbalance** the more often **mid price indicator** indicates that **mid price** does not raise in next time step – it is especially visible for stock 2748 [1]. Notice that for positives median of **queue imbalance** is above 0 and for negatives it is below 0.

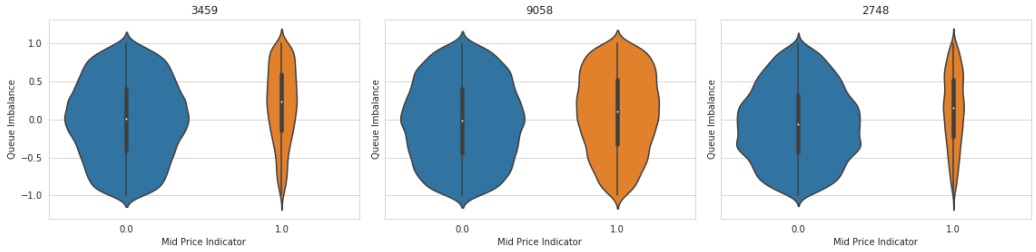

Figure 7.1: Violin Plots of **queue imbalance** vs **mid price indicator** for three stocks on training data. Value 0 (negative) of **mid price indicator** denotes that **mid price** does not raise in next time step. Value 1 (positive) – **mid price** raises in next time step

7.3.1 Logistic Regression

We apply Logistic Regression on **queue imbalance** to predict **mid price indicator**. We refer to this approach as QUE+LOG.

On Figure 7.2 we can see that using Logistic Regression as a classifier leads to a very simple decision boundary. The decision boundary divides data into two parts – all samples with **queue imbalance** below certain value are classified as "0" ("**mid price** will not raise in next time step" class), the rest of data samples are labeled with "1". The classes are separated by a constant value, we will refer to its value as a *pivot*. We can see that for all stocks from Figure 7.2 the pivot is above 0.

[1] We refer to internal stock ids – names of stocks can be checked in Table 7.1.

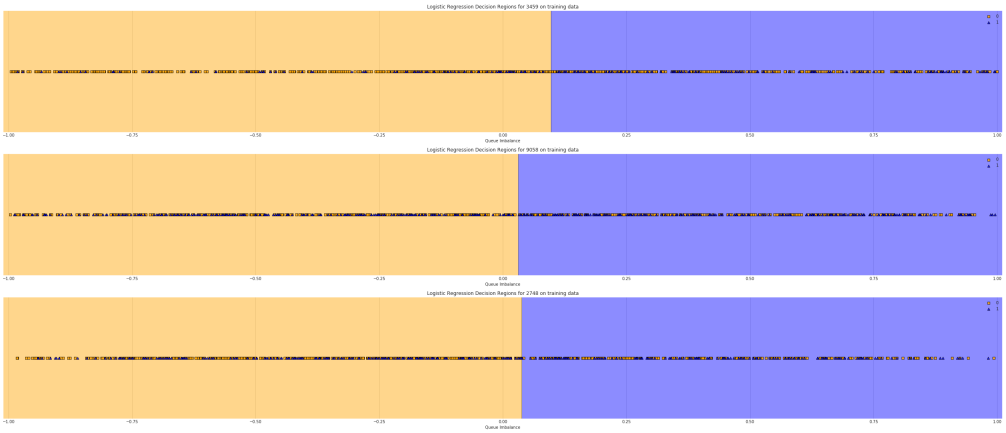

Figure 7.2: Decision boundary for QUE+LOG on 3 example stocks. The pivot for 3459 is 0.097, for 2748 is 0.038 and for 9058 is 0.031

Histogram 7.3 shows that Logistic Regression classifier on **queue imbalance** feature is always better than *null hypothesis*, since the lowest MCC score around 0.05 and the lowest ROC area score is above 0.52. Training, validation and testing scores have a very similar distributions, what implies that QUE+LOG has approximately similar performance on unseen data as on training data.

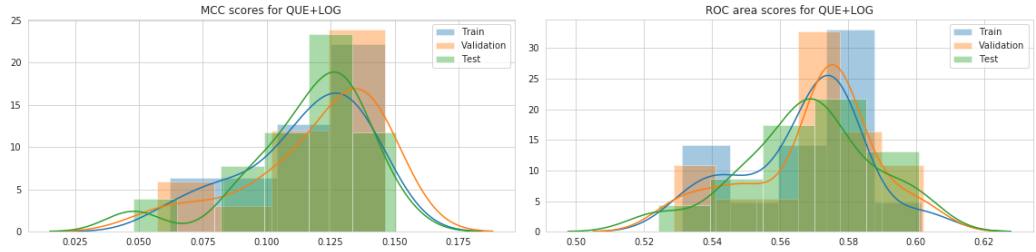

Figure 7.3: Histogram of MCC and ROC area scores for QUE+LOG

In Table 7.4 we can see that QUE+LOG can lead to $4.7 - 15.0\%$ improvement on unseen data comparing to *null hypothesis* using MCC score and $2.4 - 10.0\%$ using ROC area score. On both training and validation set improvement is bigger – for MCC score it's between $5.7 - 14.6\%$, with similar standard deviation. The mean MCC score for all training, validation and testing is between $11.4 - 11.9\%$.

The mean ROC area score is 56% for training, validation and testing data sets. For training data set the standard deviation is the lowest – 1.84% and for testing data set it is the highest 1.99%. Mean, median, minimum and maximum ROC area scores for all data sets are similar. This means that the classifier is neither underfitting or overfitting, which leads to conclusion that it found optimal decision boundary.

Table 7.4: Distribution of MCC and ROC area scores for QUE+LOG

	MCC score			ROC area score		
	Training	Validation	Test	Training	Validation	Test
mean	0.114799	0.119750	0.115945	0.566758	0.569270	0.567689
std	0.024526	0.026611	0.025105	0.018425	0.019578	0.019918
min	0.062033	0.057241	0.047951	0.531175	0.528676	0.524117
median	0.120549	0.129605	0.123012	0.574360	0.573640	0.570149
max	0.146122	0.146427	0.150420	0.601755	0.602119	0.600747

7.3.2 SVM

In this subsection we show results of applying SVM on **queue imbalance** feature to predict **mid price indicator**. We refer to this approach as QUE+SVM.

The decision boundary is a pivot shown on Figure 7.4, which splits the the data into two parts. Below certain value of **queue imbalance** all data samples will be labeled as "0", the data samples at the other side with "1". It is similar behaviour as for QUE+LOG, only the value of the pivots differ. For stock 3459 the difference between pivots obtained for QUE+LOG and QUE+SVM is about 0.04, for 2748 is around 0.12. For stock 9058 the difference is smaller – around 0.02.

Figure 7.4: The decision boundary of QUE+SVM for 3 example stocks. For 3459 the pivot is around 0.046, for 2748 it is 0.158 and for 9058 it is 0.013

In Table 7.5 we can see hyper-parameters chosen during validation for three example stocks. Stock 2748 for which SVM with *RBF* kernel was chosen during model selection (described in Section 6.2.2) is the one with very different value of pivot than the pivot obtained using QUE+LOG approach. For the other two stocks SVM with *sigmoid* kernel is chosen.

Table 7.5: Parameters for QUE+SVM classifiers for three example stocks

stock	kernel	C	γ	coef0
3459	sigmoid	10.0	0.010	1.0
9058	sigmoid	10.0	0.001	1.0
2748	RBF	1000.0	0.010	-

We show the results obtained for the selected models using model selection described in Section 6.2.2. We are using two approaches:

1. treating SVM kernel as hyper-parameter
2. not treating SVM kernel as hyper-parameter

In the first approach during model selection we chose only one SVM classifier per stock with the best MCC score obtained during validation. In the second one we chose three models per stock, each one have different kernel (*linear*, *RBF* or *sigmoid*).

On Figure 7.5 there are distributions of MCC scores obtained for all chosen stocks for each data set. For training and validation set all the obtained scores are above 0. Distributions of scores for each data sets are similar, hence the model generalizes well.

Both median and mode of the scores for all training, validation and test data sets are above 0.1. The IQR is higher for results obtained on validation data set, than for both training and test data sets. When looking at modes shown on distribution plot on the left, for the validation data set results we reach the higher density of scores, than for training or test data sets. This is also an indication that the chosen models will have a good generalization properties.

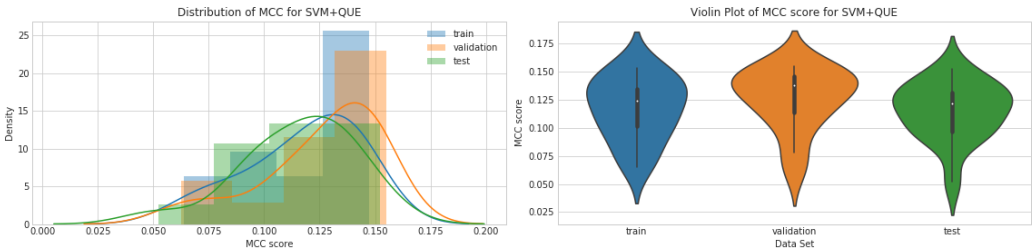

Figure 7.5: Histogram and Violin Plot of MCC scores for QUE+SVM

When we choose the best classifier for each stock by kernel, instead of treating type of kernel as hyper-parameter, we can see some similarities between the scores obtained for *linear* and *RBF* based SVMs. Figure 7.6 shows the distributions of MCC scores per kernel. We can see that for *sigmoid* kernel there is at least one

stock for which MCC score on test data set is below 0, which means it the chosen model for that stock is worse than *null hypothesis*. For other kernels the scores for testing data sets have similar range as for training or validation data sets, hence it is likely that they will generalize well.

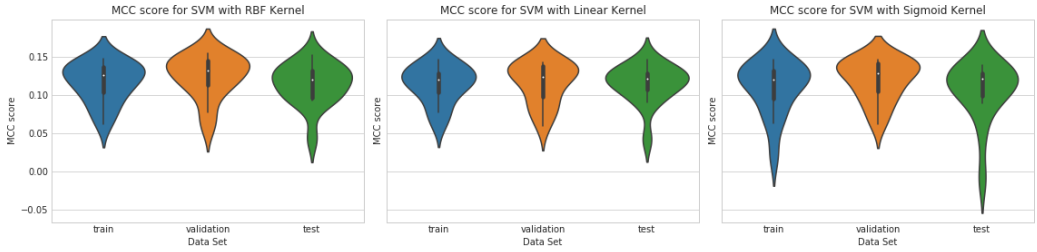

Figure 7.6: Violin Plots of MCC scores for QUE+SVM per kernel

Table 7.6 shows the summary of MCC scores chosen SVMs with *RBF* and *linear* kernels. Both classifiers have very similar average score on unseen data. On validation set SVM with *RBF* kernel have a higher mean score by 0.7% than for training data set, which may indicate very slight overfitting. When we look at median score for *RBF* kernel it is also higher for validation data set than for other data sets, therefore this slight overfitting might be for more than one stock.

Table 7.6: Distribution of MCC scores QUE+SVM with *RBF* and *linear* kernel

	RBF Kernel			Linear Kernel		
	Train	Val.	Test	Train	Val.	Test
mean	0.117134	0.124851	0.114705	0.114789	0.116256	0.114962
std	0.025720	0.027684	0.026943	0.024797	0.026978	0.025253
min	0.062020	0.058888	0.043992	0.061270	0.059446	0.042593
median	0.126633	0.131952	0.120548	0.120379	0.124565	0.121280
max	0.147411	0.155044	0.152323	0.146167	0.143231	0.146225

The mean scores for *sigmoid* kernel shown in Table 7.7 for testing data set are worse by about 1% than for classifiers with other kernels, despite having good average MCC score on validation data set. It is due to negative minimum score on testing data set, which is by more 4% less than for other kernels. This leads to conclusion that SVM with *sigmoid* kernel does not generalize well for this classification problem or the hyper-parameters could be improved.

Table 7.7: Distribution of MCC scores QUE+SVM with *sigmoid* kernel

	Sigmoid Kernel		
	Train	Validation	Test
mean	0.109992	0.120235	0.105829
std	0.034344	0.026814	0.038864
min	0.021764	0.062361	-0.008427
median	0.121163	0.129108	0.121606
max	0.147290	0.146541	0.140110

We take a closer look at the decision boundaries of the worst scored stock for QUE+SVM with *sigmoid* kernel on Figure 7.7. We can see that there is more than one pivot for *sigmoid* kernel, which splits the decision surface into four parts. On the other hand SVMs with *linear* and *RBF* kernel have similar decision boundaries.

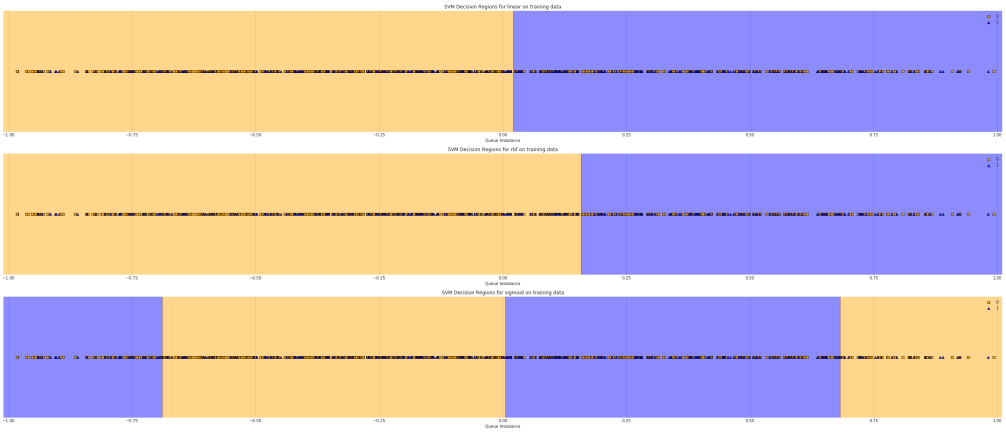

Figure 7.7: Decision boundaries for *linear*, *RBF* and *sigmoid* kernels for stock 2748, for which the score for QUE+SVM with *sigmoid* kernel on test data set was below 0

On Figure 7.4 we noticed that SVMs with *linear* kernel could have a similar decision boundary as QUE+LOG. On Figure 7.8 we can see that this is true for only a few stocks, like 4799 and 3459. We can also see that for *sigmoid* kernel, not always single pivot could be found, for instance for stocks 9061 and 2748. Also QUE+SVM with *sigmoid* kernel is the only classifier which has pivots below 0 (for stocks 9062 and 4851). We clearly see decision boundaries vary between different approaches.

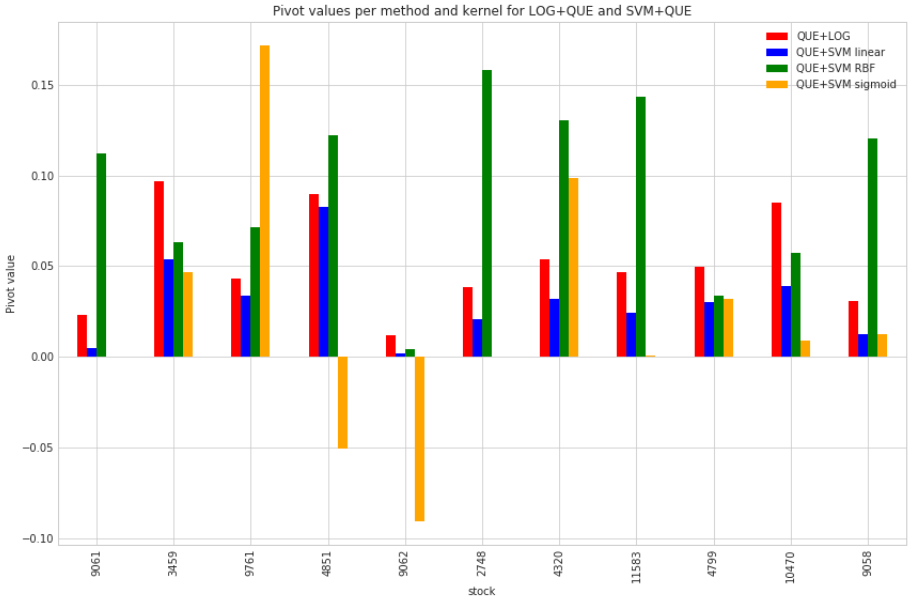

Figure 7.8: Bar plot of pivots (decision boundaries) for QUE+SVM with *linear*, *RBF* and *sigmoid* kernels and QUE+LOG

7.3.3 Comparison of QUE+SVM and QUE+LOG

In this subsection we will compare QUE+SVM with QUE+LOG. Table 7.8 shows the differences between MCC and ROC area scores between best QUE+SVM with kernel treated as hyper-parameter and QUE+LOG. For training and test data set average MCC score difference is below 0, which means that on average LOG+QUE yields better results on unseen data. On the other hand, because median of difference of MCC score on test data set is positive, for at least half of the stocks QUE+SVM gives better results. We took QUE+SVM with all kernels into consideration, knowing that SVM with *sigmoid* kernel (chosen for 6 stocks) does not generalize well in our case, which might be the cause of that result.

Table 7.8: Differences for MCC and ROC area scores between best QUE+SVM for all kernels and QUE+LOG

	Differences in MCC score			Differences in ROC area score		
	Train	Val.	Test	Train	Val.	Test
mean	0.002338	0.006138	-0.000807	0.000667	0.002430	-0.000873
std	0.003944	0.003013	0.005843	0.002741	0.002701	0.003950
min	-0.001979	0.000596	-0.016729	-0.003619	-0.005594	-0.010784
median	0.001357	0.005222	0.001009	0.000526	0.002529	0.000883
max	0.012028	0.010775	0.004147	0.007462	0.006645	0.002235

Table 7.9 shows the differences for MCC and ROC area scores between QUE+SVM

with *sigmoid* kernel and QUE+LOG. We can see that on unseen data the average score is negative, hence QUE+LOG on average gives better results. However, still for at least half of stocks the score on unseen data is positive, because the median is above 0. The minimum score differences on test data set are very low – -14.2 for MCC and -10.2 for ROC area. For training and validation data sets the minimum scores are much higher than for validation data set, what implies that for some of the stocks QUE+SVM with *sigmoid* kernel is not generalizing well.

Table 7.9: Differences for MCC and ROC area scores between best QUE+SVM with *sigmoid* kernel and QUE+LOG

	Differences in MCC score			Differences in ROC area score		
	Train	Val.	Test	Train	Val.	Test
mean	-0.004808	0.000484	-0.010115	-0.003906	-0.000060	-0.007607
std	0.022348	0.014798	0.036938	0.015629	0.010536	0.026503
min	-0.084091	-0.052157	-0.142697	-0.059589	-0.037599	-0.102648
median	0.000591	0.004229	0.000997	-0.000083	0.002185	0.000485
max	0.008193	0.009127	0.004492	0.004210	0.005552	0.002547

The results for QUE+SVM with *RBF* and *linear* kernel compared with QUE+LOG are shown in Table 7.10. QUE+SVM for test data set is only slightly worse than QUE+LOG, because mean difference is negative, but close to 0. For QUE+SVM with *linear* kernel median MCC score on test data set is negative, therefore it is worse than QUE+LOG for at least half of the stocks. On the other hand for *RBF* kernel the median is positive. The minimum score differences for both kernels are close to 0, so these approaches are only slightly worse than QUE+LOG.

Table 7.10: Differences for MCC between best QUE+SVM with *RBF* kernel or *linear* kernel and QUE+LOG

	RBF Kernel			Linear Kernel		
	Train	Val.	Test	Train	Val.	Test
mean	0.002335	0.005101	-0.001240	-0.000010	-0.003494	-0.000983
std	0.003898	0.003560	0.005904	0.002015	0.016065	0.003295
min	-0.001979	0.000596	-0.016729	-0.002369	-0.060775	-0.006146
median	0.001013	0.004235	0.001237	-0.000587	0.000599	-0.000322
max	0.012028	0.010775	0.004062	0.005056	0.004122	0.005597

7.3.3.1 McNemar Test for QUE+SVM and QUE+LOG

To check if QUE+SVM differs from QUE+LOG we apply McNemar test. Figure 7.9 shows the results of McNemar test for comparing QUE+SVM with different kernels and QUE+LOG. For QUE+SVM with *RBF* kernel we obtained p-value above threshold for three stocks: 1431, 9061 and 9062. For QUE+SVM with *linear*

kernel p-value is above threshold for one additional stock. For *sigmoid* kernel p-value is above threshold for 9062, 2748 and 9061, for which *sigmoid* kernel has no single decision boundary or a very low pivot.

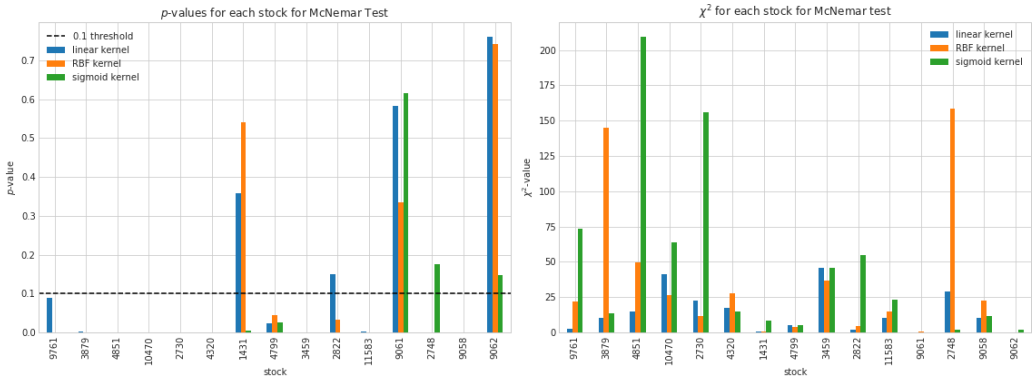

Figure 7.9: P-values from McNemar test between QUE+SVM per kernel and QUE+LOG

For approach which treats kernel as a hyper-parameter the McNemar test results are shown on Figure 7.10. For only two stocks QUE+SVM approach has p-value above threshold – 9061 and 9062. For these stocks we cannot reject McNemar test *null hypothesis* that QUE+LOG and QUE+SVM have similar error rate.

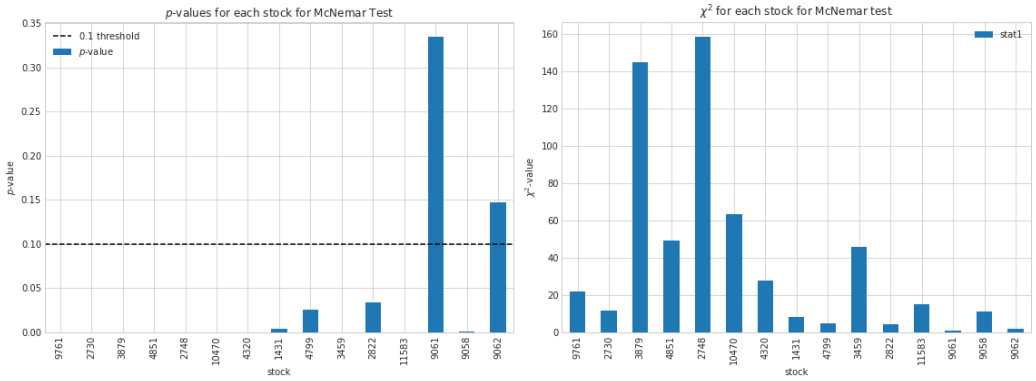

Figure 7.10: p-values from McNemar test between QUE+SVM per kernel and QUE+LOG

7.3.4 Summary

In this section we shown results for **queue imbalance** feature for prediction of **mid price indicator** using Logistic Regression and SVM. Our finding is that QUE+LOG has better average scores than QUE+SVM and has better generalization properties. QUE+SVM approach is more prone to overfitting, especially if *sigmoid* kernel is used. However QUE+SVM yields better results for more than half of the stocks,

one have to be very careful with hyper-parameters. According to McNemar test results QUE+SVM have different error rates than QUE+LOG for almost all stocks, so we know that for these QUE+SVM approach has different performance than QUE+LOG.

7.4 Queue Imbalance and Previous Queue Imbalance

In this section we briefly present our findings regarding using both **queue imbalance** and **previous queue imbalance** for predicting **mid price indicator** using Logistic Regression (PREVQUE+LOG) and SVM (PREVQUE+SVM).

In Table 7.11 we can see that on average PREVQUE+LOG yields very small improvement on unseen data. On test data set mean MCC score difference is only 0.2%. For more than half of the stocks the mean difference between both MCC and ROC area scores is positive, therefore introducing **previous queue imbalance** slightly improves the quality of predictions, but not for all stocks, since minimum difference is negative. The standard deviation of the difference between the scores is never higher than 0.7%, what suggests that PREVQUE+LOG has very similar performance to QUE+LOG.

Table 7.11: Difference for MCC and ROC area scores between PREVQUE+LOG and QUE+LOG

	Differences in MCC score			Differences in ROC area score		
	Train	Val.	Test	Train	Val.	Test
mean	0.000003	-0.000726	0.002391	0.000001	0.000429	0.001478
std	0.000093	0.005419	0.006109	0.000050	0.002913	0.003570
min	-0.000138	-0.010448	-0.011217	-0.000072	-0.004783	-0.006107
median	-0.000020	-0.001500	0.002183	-0.000014	0.000560	0.001369
max	0.000206	0.007166	0.015575	0.000114	0.004883	0.009133

On Figure 7.11 we can see that decision boundaries for three different stocks look similar. The **previous queue imbalance** is less significant feature, because the decision boundary only slightly changes due to different values of **previous queue imbalance**. The **queue imbalance** seems to have the biggest impact over the decision boundary, because the decision boundary line has small slope.

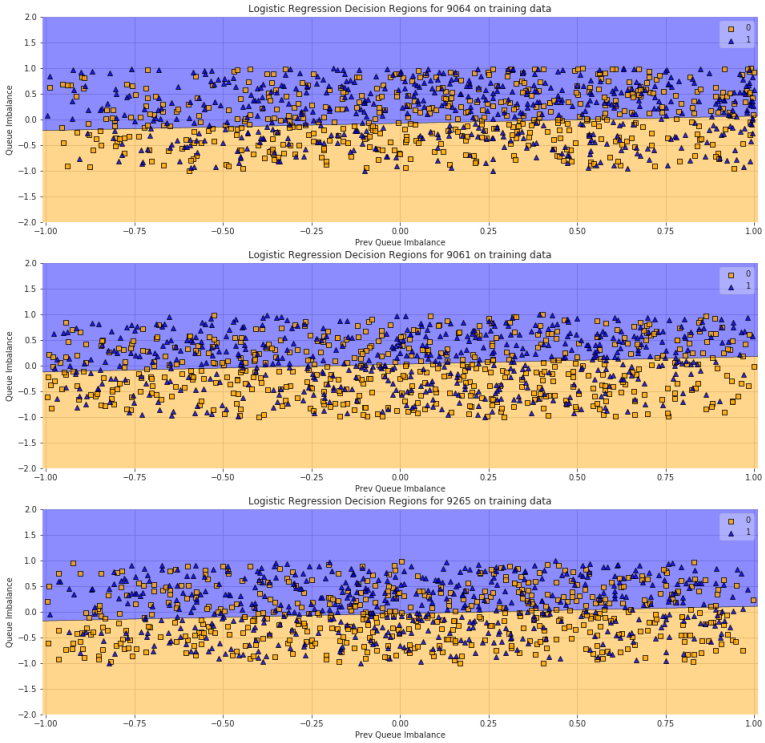

Figure 7.11: Decision boundary for PREV+QUE+LOG on 3 example stocks

We present comparison between PREVQUE+SVM and baseline in Table 7.12. We can see that according to both MCC and ROC area score PREVQUE+SVM performs worse than QUE+LOG, by about 1% on test data set. We consider here SVMs with all kernels, hence the minimum difference is very big 8%, because we include *sigmoid* kernel in hyper-parameter list. It is possible that with limiting kernels to *RBF* or *linear* would improve PREVQUE+SVM results significantly as we observed for QUE+SVM approach.

Table 7.12: Difference between scores for PREVQUE+SVM and QUE+LOG

	Differences in MCC score			Differences in ROC area score		
	Train	Val.	Test	Train	Val.	Test
mean	-0.014709	0.004052	-0.017222	-0.008780	1.729077e-03	-0.010351
std	0.047510	0.003757	0.045408	0.028852	2.522757e-03	0.026951
min	-0.129963	-0.000369	-0.130774	-0.081691	-8.095128e-04	-0.081522
median	0.000426	0.004068	-0.000868	0.000107	1.048351e-03	-0.000468
max	0.034882	0.013937	0.006190	0.024518	8.829469e-03	0.002954

The decision boundary obtained using *sigmoid* kernel is radically different than for *RBF* kernel as shown on Figure 7.12.

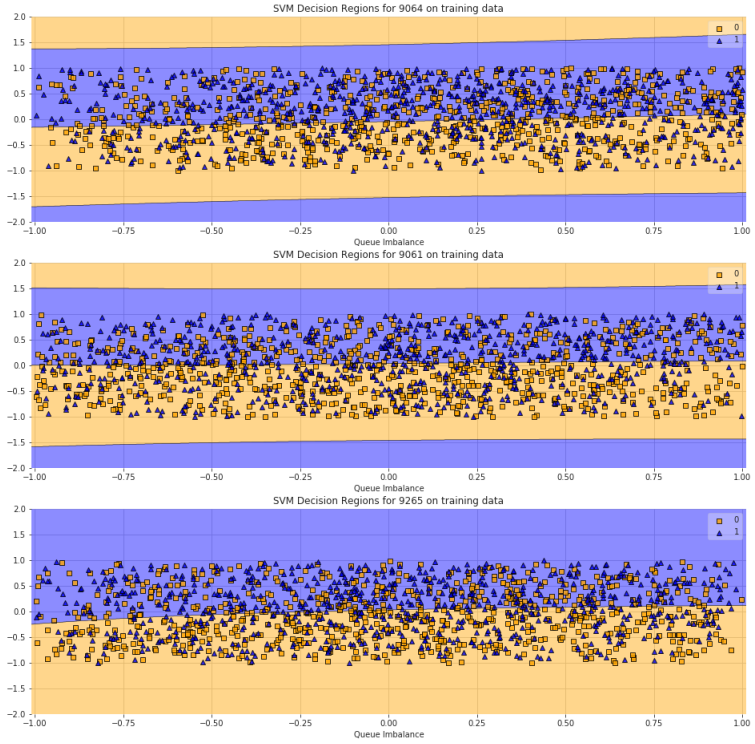

Figure 7.12: Decision boundary for PREV+QUE+SVM on 3 example stocks. The two top ones are obtained using SVM with *sigmoid* kernel, the bottom one is obtained using SVM with *RBF* kernel

Introducing **previous queue imbalance** feature does not bring improvement over predictive powers of classifiers. It might be that if for PREVQUE+SVM we exclude *sigmoid* kernel from hyper-parameter list, PREVQUE+SVM could perform better. On the other hand the decision boundary obtained for PREVQUE+LOG suggests that **previous queue imbalance** has small impact on the result of the prediction.

7.5 Gaussian Density Filter

In this section we describe the classification results obtained by using features extracted from Limit Order Book using Gaussian Density Filter and reduced using PCA as input for Logistic Regression (GDF+PCA+LOG), MLP (GDF+PCA+MLP), LSTM (GDF+PCA+LSTM) and GRU (GDF+PCA+GRU) algorithms. First we will show a few visualizations of features obtained using that approach.

On Figure 7.13 we can see the distribution of GDF features for on of the stocks. We can see that the middle GDF features (24 and 25) have the smallest values. That could be, because the orders with the highest selling or buying pressure have the price closest to **mid price**. These orders have lower number of volumes than

the rest of Limit Order Book.

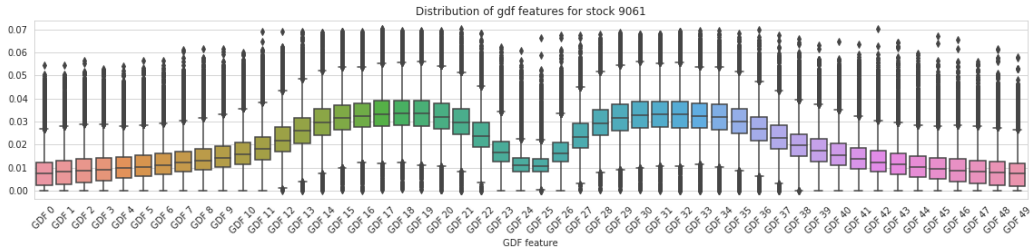

Figure 7.13: Box plot of GDF features

Box plot 7.14 presents distribution of values of GDF features reduced by PCA with 4 components. We can see that the first component (number 0 on the picture) has biggest range of values. The higher number of the component the lower range of its values. We can also see that the median is close to 0 for all the components.

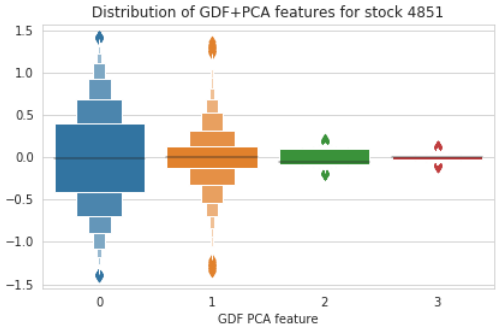

Figure 7.14: GDF features distribution after applying reduction PCA with 4 components

7.5.1 Logistic Regression on GDF+PCA without QUE

First we tried to apply Logistic Regression on GDF+PCA features without **queue imbalance** added to feature space before applying PCA. We briefly present comparison with QUE+LOG in Table 7.13. We can see that for all data sets score differences are negative, hence GDF+PCA without **queue imbalance** feature space might be too complex for Logistic Regression. It is not shown in Table, but this approach performed worse than QUE+LOG for 12 out of 15 stocks.

Table 7.13: Differences for MCC and ROC area scores between GDF+PCA+LOG without queue imbalance and QUE+LOG

	MCC score difference			ROC area score difference		
	Train	Validation	Test	Train	Validation	Test
mean	-0.000940	-0.025107	-0.020054	0.000085	-0.016756	-0.012798
std	0.033434	0.018450	0.027543	0.019580	0.010804	0.020011
min	-0.044104	-0.065570	-0.076952	-0.022250	-0.041686	-0.067335
median	-0.003918	-0.023622	-0.016012	-0.002754	-0.015006	-0.009908
max	0.111479	-0.000514	0.027931	0.067157	-0.000713	0.021243

We decided to add **queue imbalance** to the GDF feature space before applying PCA reduction, since we already have seen that it has slight statistical relationship with **mid price indicator**. We will check if it brings improvement over QUE+LOG.

7.5.2 Logistic Regression on GDF+PCA with QUE

We use GDF+PCA features with **queue imbalance** added to the feature space as an input to Logistic Regression algorithm to predict **mid price indicator**. Table 7.14 shows that GDF+PCA+LOG is on average better by 0.2% on test data set when looking at MCC score and 0.1% better when considering ROC area score. For both scoring methods minimum difference is negative, so there are stocks for which QUE+LOG performed better by at around 1%. Mean scores obtained on training data sets are on average at least 0.5% better than these obtained during validation. It may imply that GDF+PCA+LOG is underfitting. It is not shown in Table, but GDF+PCA+LOG approach performed worse than QUE+LOG for 7 out of 15 stocks.

Table 7.14: Differences for scores between GDF+PCA+LOG and QUE+LOG

	MCC score difference			ROC area score difference		
	Train	Validation	Test	Train	Validation	Test
mean	0.013713	0.006409	0.002673	0.008519	0.003760	0.001097
std	0.031174	0.007547	0.009843	0.020929	0.004972	0.006141
min	-0.000073	-0.000329	-0.016091	-0.000038	-0.000163	-0.009532
median	0.006159	0.003556	0.001498	0.003453	0.001964	0.000510
max	0.125476	0.026575	0.025824	0.083765	0.018114	0.017033

It might be that Logistic Regression algorithm is not sophisticated enough to use the full potential from GDF+PCA features. Therefore we try more complex classification algorithm like MLP.

7.5.3 MLP on GDF+PCA

We use MLP on GDF+PCA with **queue imbalance** added to feature space to predict **mid price indicator**. Histogram 7.15 shows that MLP classifier on GDF+PCA features is always better than Null Hypothesis, since for testing data set the lowest MCC score is 0.05 and the lowest ROC area score is 0.525. The spread is the biggest for training set, because there is at least one stock for which training score is above 0.2 using MCC or 0.64 using ROC area score. This might suggest overfitting for at least one stock. For all data sets the mode is between $0.12-0.15$ MCC or $0.55-0.58$, hence for most of the stocks GDF+PCA+MLP generalizes well.

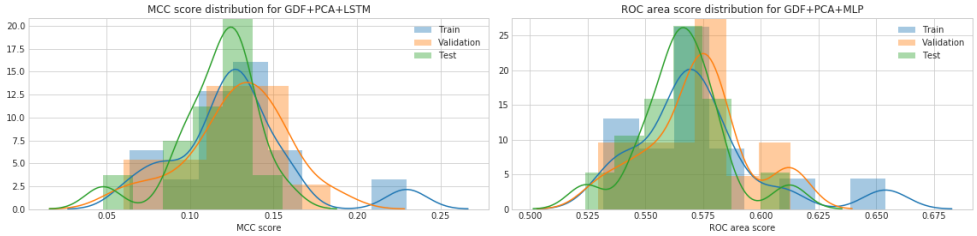

Figure 7.15: Histogram of scores obtained by GDF+PCA+MLP

In Table 7.15 we show that GDF+PCA+MLP features can lead to $2.9-16.1\%$ improvement comparing to Null Hypothesis using MCC score and $1.9-11.4\%$ using ROC area score. For all data sets mean and median of MCC scores is between $12.1-13.2\%$ what confirms that in general GDF+PCA+MLP generalizes well.

Table 7.15: Distribution of scores for GDF+PCA+MLP

	MCC score			ROC area score		
	Training	Validation	Test	Training	Validation	Test
mean	0.127496	0.124227	0.121182	0.574643	0.571350	0.569692
std	0.041848	0.033867	0.029092	0.030806	0.023367	0.021724
min	0.039851	0.040740	0.039844	0.520022	0.519999	0.520039
median	0.127807	0.132029	0.125039	0.576860	0.575429	0.568468
max	0.227642	0.173214	0.161993	0.653209	0.607803	0.614215

7.5.3.1 Comparison GDF+PCA+MLP and QUE+LOG

In Table 7.16 there are shown differences for MCC and ROC area scores between GDF+PCA+MLP and QUE+LOG. We can see that mean of the difference on test set when using MCC score is above 0.05%, which means that GDF+PCA+MLP on average performed better than QUE+LOG. For training data set the difference is bigger – 0.12%, which could mean that with further hyper-parameter tuning MLP classifier could perform even better on other data sets according to MCC score.

Table 7.16: Distribution of differences for MCC and ROC area scores between GDF+PCA+MLP and QUE+LOG

	MCC score difference			ROC area score difference		
	Train	Validation	Test	Train	Validation	Test
mean	0.012697	0.004476	0.005238	0.007885	0.002080	0.002002
std	0.031388	0.010306	0.008649	0.019970	0.005400	0.006779
min	-0.022182	-0.016501	-0.008107	-0.011154	-0.008678	-0.013138
median	0.008681	0.003440	0.004505	0.004877	0.001789	0.001731
max	0.121787	0.029587	0.020675	0.078111	0.014228	0.013467

Minimum difference for testing set is -0.8%, which means that there is at least one stock for which QUE+LOG performs significantly better. Maximum difference for testing set is 2%, hence GDF+PCA+MLP has ability to perform better than QUE+LOG for certain stocks. The maximum difference is significantly bigger for training data set – 12%, therefore QUE+LOG most probably does not overfit for any stock whereas GDF+PCA+MLP does.

For ROC area score the mean of the difference on test set is 0.2%, what confirms better average performance for QUE+PCA+MLP. Minimum ROC area score differences are comparable to MCC ones – they are also negative but not less than -2%. Positive median suggests that GDF+PCA+MLP performed better than QUE+LOG for at least half of the stocks.

Using MLP on GDF+PCA features brings a slight improvement compared to QUE+LOG, but we possible can improve more by using an algorithm which can use the fact that we are classifying patterns from time series data.

7.5.4 LSTM on GDF+PCA

We use LSTM on GDF+PCA with **queue imbalance** added to feature space to predict **mid price indicator**. As described in Section 6.4.4 after applying GDF filter to Limit Order Book we reduce the dimensionality of the extracted features using PCA. Figure 7.16 shows the number of PCA components used for each stock. The approach of choosing the number of PCA components is described in Section 6.4.2.1 – in essence, we use the minimal number of components which keep 99% of explained variance.

On Figure 7.16 we can see that the choice of PCA components differs for each stock, which tells about how many variables are needed to explain the variance of GDF features. The most frequent component number is 2. Component numbers 3, 6 and 8 are assigned for one stock each. Components numbers 5, 9 and 10 were not chosen for any of the stocks. For stock 2748 the highest number of PCA components was used – it is a stock with the highest mean spread as shown in Table 7.1.

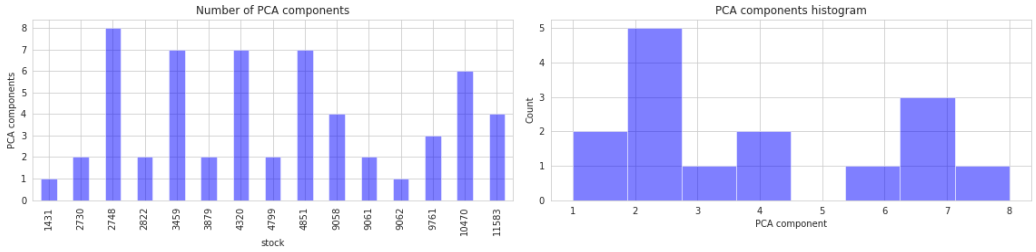

Figure 7.16: Bar plot and histogram of number of used PCA components used in GDF+PCA+LSTM

7.5.4.1 Comparison with Null Hypothesis

Histogram 7.17 shows that LSTM classifier on GDF features reduced by PCA (GDF+PCA+LSTM) is always better than *null hypothesis*, since the lowest MCC score is 0.05 and the lowest ROC area score is 0.525. For both MCC and ROC area scores the mode for training, validation and test data sets has similar values. Wider range of scores for training data sets may imply that there is at least one stock for which GDF+PCA+LSTM is overfitting, since testing and validation data score spreads are different from training data set spread. On the other hand similar spreads of validation and testing suggest that GDF+PCA+LSTM generalizes well for unseen data.

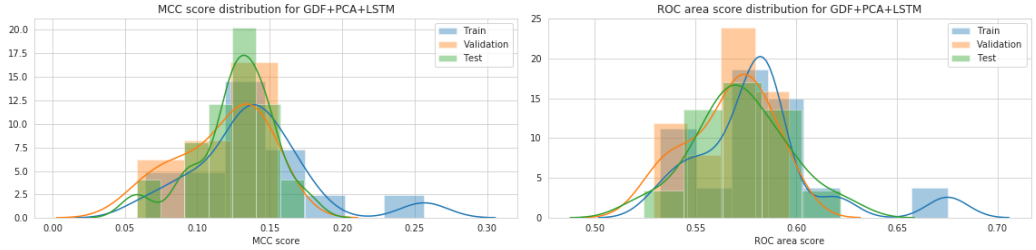

Figure 7.17: Histogram of MCC and ROC area scores obtained by GDF+PCA+LSTM

In Table 7.17 we can see that GDF+PCA+LSTM approach can lead to 5.8 – 17.3% improvement on unseen data comparing to *null hypothesis* when looking at MCC score and 2.4 – 12.2% on ROC area score. On training data set mean improvement is bigger than during validation and testing. We can also see that mean scores obtained on testing data set are higher than validation scores, therefore we can conclude that GDF+PCA+LSTM in overall generalizes well. Standard deviation for both MCC and ROC area scores on training data set is higher than for other data sets, what may suggest that there is at least one stock, which may not generalize well. Minimum and median scores are similar between data sets, but maximum scores are much higher for training data sets (10.1% using MCC score, 7.9% using ROC area score) than for the rest of data sets, what may suggest overfitting

for at least one stock. However we do not observe issues with generalization when comparing maximum MCC score on validation set to testing set.

Table 7.17: Distribution of MCC and ROC area scores for GDF+PCA+LSTM

	MCC score			ROC area score		
	Training	Validation	Test	Training	Validation	Test
mean	0.138902	0.116735	0.126441	0.581095	0.566306	0.572502
std	0.044075	0.030138	0.028127	0.033552	0.020367	0.024093
min	0.064279	0.058467	0.058114	0.532260	0.529052	0.524187
median	0.137956	0.127198	0.128872	0.581057	0.571098	0.570503
max	0.256568	0.155287	0.173571	0.675172	0.596208	0.622194

On Figure 7.18 we plot scores for each stock. We can see that for stock 2748 GDF+PCA+LSTM may overfit – this is the stock which has the highest score on training data set. We reach highest score on testing data set for different stock – 4851. The differences between MCC scores are bigger than for ROC area scores for the stocks which do not have comparable scores on different data sets.

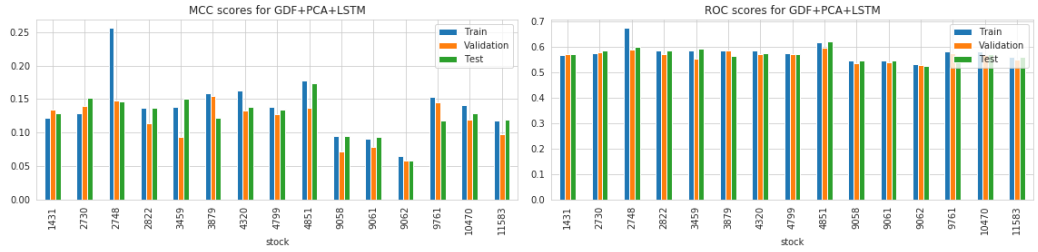

Figure 7.18: Bar plot of MCC and ROC area scores obtained by GDF+PCA+LSTM

To check if GDF+PCA+LSTM is overfitting for stock 2748 we plot *loss* function on validation and training data sets on Figure 7.19. The validation loss is lower than loss on training data set, so the GDF+PCA+LSTM classifier we have chosen is underfitting instead of overfitting. To fix underfitting we can increase complexity of the model by introducing more LSTM units or by stacking LSTM networks.

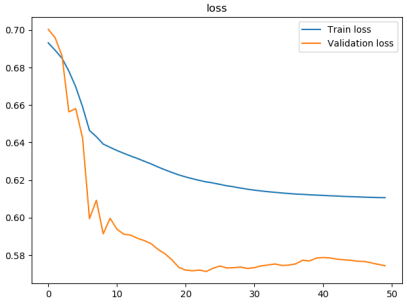

Figure 7.19: Loss function for stock 2748

7.5.4.2 Comparison GDF+PCA+LSTM with QUE+LOG

To compare results obtained for GDF+PCA+LSTM and QUE+LOG we look at distributions of scores differences between these approaches shown in Table 7.18. On average GDF+PCA+LSTM approach is better by 1.0% on unseen data according to MCC score difference. For ROC area score the difference is smaller 0.4%. For more than half of the stocks the GDF+PCA+LSTM is more than $0.8 - 0.9\%$ better than QUE+LOG when looking at MCC score.

Minimum difference between MCC and ROC area scores obtained during training and validation are negative, therefore QUE+LOG performed better on these, however the minimum score for testing data set is still positive, which may imply that GDF+PCA+LSTM approach still works better on unseen data. The reason behind lower validation scores for GDF+PCA+LSTM might be that because of the complexity of the model it requires more data to be trained properly. Since we used 5-Fold Forward Cross-Validation (Subsection 3.3.2.1) for hyper-parameter tuning, the model was trained on smaller data set for each validation group. It might be that it would be more beneficial to train models using less number of folds, so we train them on more data. However cons of that approach are that there would be fewer validation results and the obtained results might not indicate well enough if the classifier generalizes on unseen data.

Maximum difference is about 15.0% for MCC score obtained on training data set, but for validation and testing data sets, the maximum difference we obtained is only $1-3.4\%$. Similar tendency is visible when looking at maximum score differences for ROC area score – for training there is 10% of maximum difference, but for testing and validation it is $1 - 2\%$ only.

Table 7.18: Difference for MCC and ROC area scores between GDF+PCA+LSTM and QUE+LOG

	MCC score			ROC area score		
	Training	Validation	Test	Training	Validation	Test
mean	0.024102	-0.003015	0.010497	0.014336	-0.002964	0.004812
std	0.036417	0.010991	0.009834	0.024347	0.007092	0.006704
min	-0.003132	-0.027324	0.001086	-0.001983	-0.021056	0.000070
median	0.016754	-0.000713	0.010163	0.009008	-0.001935	0.001409
max	0.150713	0.017933	0.034891	0.100074	0.010326	0.021446

To compare partial results of GDF+PCA+LSTM and QUE+LOG on unseen data we use bar plot shown on Figure 7.20. We can see that for stock 4851, 2822 and 3459 the differences between both MCC and ROC area scores were the highest.

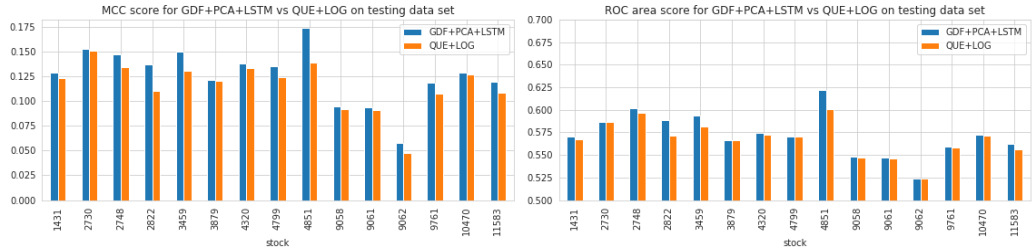

Figure 7.20: Bar plot of MCC and ROC area scores obtained by GDF+PCA+LSTM and QUE+LOG

7.5.4.3 McNemar Test

We perform McNemar test to compare if GDF+PCA+LSTM and QUE+LOG are similar classifiers. For each stock *null hypothesis* during McNemar test assumes that GDF+PCA+LSTM and QUE+LOG have similar error rate. We calculate χ^2 statistics and corresponding p-values, which are shown on Figure 7.21. For stocks 9058, 4320, 2748 and 3459 we cannot reject *null hypothesis* from McNemar test, because the p-values are above threshold 0.1.

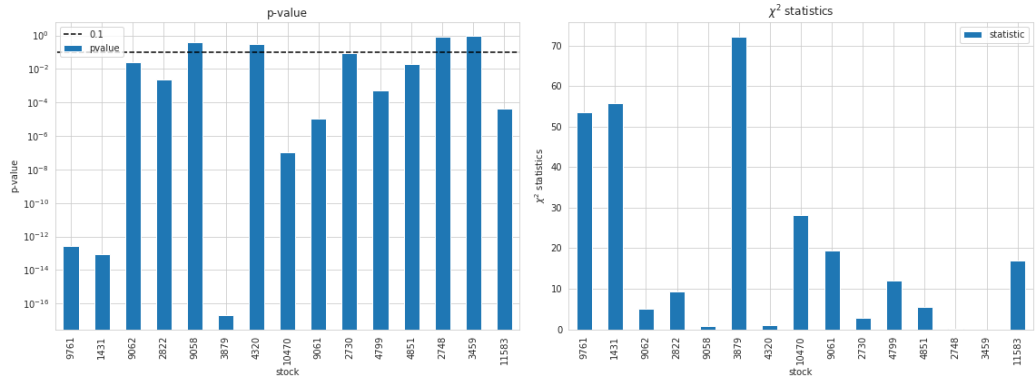

Figure 7.21: p-value and χ^2 statistics for each stock

Table 7.19 shows distributions of p-values and χ^2 statistics. The mean p-value is 0.17, which is bigger than threshold 0.1, also the maximum p-value is very high – 0.93. However the median is much below threshold. We saw that for four stocks p-value was above threshold, therefore for 75% stocks there is a strong evidence which allows us to reject Null-Hypothesis. For the rest of the stocks the evidence is too weak to reject Null-Hypothesis from McNemar test. Usually GDF+PCA+LSTM classifiers have different error rate than QUE+LOG, hence they are usually different.

Table 7.19: Distribution of p-values and χ^2 statistics

	p-value	χ^2 statistic
mean	0.1768630	18.824402
std	0.3206039	23.330439
min	1.957944×10^{-17}	0.006653
median	2.357731×10^{-3}	9.247842
max	0.9349928	72.186461

GDF+PCA+LSTM reached the scores on average higher by 1% than QUE+LOG and for most of the stocks has different error rate. Therefore we can conclude that GDF+PCA+LSTM approach usually works better. The results for GDF+PCA+LSTM we obtained could be improved by further hyper-parameter tuning, since we did not try all possible parameters.

7.5.5 GRU on GDF+PCA

Similarly as for GDF+PCA+LSTM for GDF+PCA+GRU we use PCA to reduce the dimensionality of the data. On Figure 7.22 we show the component numbers. Similar as for GDF+PCA+LSTM the most often 2 PCA components are used. For 2730 we use the highest number of components, where in GDF+PCA+LSTM we used only 2. Hence the number of used components differ between these two approaches.

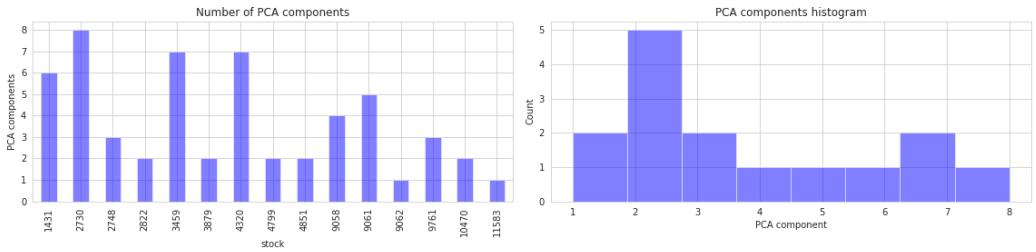

Figure 7.22: Bar plot and histogram of number of used PCA components by GDF+PCA+GRU

From Table 7.20 we can see that MCC score is always higher than 0, which means GDF+PCA+GRU is always better than *null hypothesis*. Similar as for GDF+PCA+LSTM the maximal result for training data set is much higher than other results, which results in higher mean score on testing data set as well.

Table 7.20: Distribution of scores for GDF+PCA+GRU

	MCC score			ROC area score		
	Training	Validation	Test	Training	Validation	Test
mean	0.140834	0.129516	0.124911	0.582206	0.574047	0.572531
std	0.046199	0.032159	0.028992	0.034428	0.022109	0.023380
min	0.063412	0.058509	0.050159	0.531857	0.529132	0.525227
median	0.139917	0.133453	0.131989	0.584202	0.580344	0.573492
max	0.266273	0.187762	0.167083	0.677755	0.611680	0.619784

On Figure 7.23 we see that similarly as for GDF+PCA+LSTM maximum training score is obtained for stock 2748 and the highest testing score is for 4851. For 2748 the best chosen GDF+PCA+GRU model has 32 GRU units with *tanh* activation function, *gdf* parameters: $r = 0.01$, $\sigma = 0.1$ and no kernel regularization for the GRU layer. For GDF+PCA+LSTM the chosen model is different, it contains 16 LSTM units with *tanh* activation function with kernel *l2* regularization 0.01 and *gdf* parameters: $r = 0.1$ and $\sigma = 0.1$. The GDF+PCA+GRU model obtained slightly worse score on testing data set for that particular stock. For both GDF+PCA+LSTM and GDF+PCA+GRU approaches the space of hyper-parameters we tried appears to not be sufficient for this stock. On the other hand for other stocks the issues with overfitting or underfitting are not so severe.

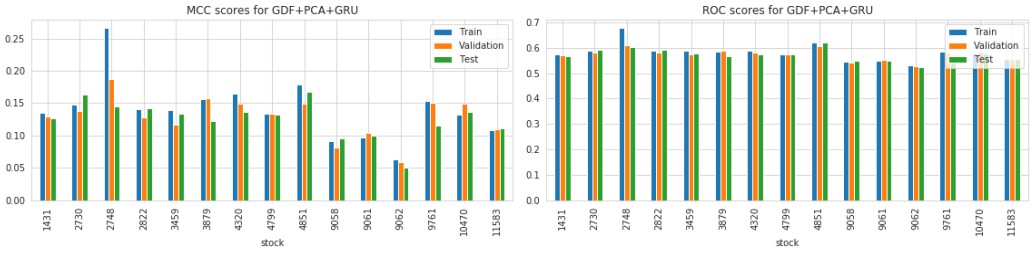

Figure 7.23: Bar plot of MCC and ROC area scores obtained by GDF+PCA+GRU

7.5.5.1 Comparison GDF+PCA+GRU with QUE+LOG

We compare GDF+PCA+GRU with baseline QUE+LOG and show differences between scores in Table 7.21. On average GDF+PCA+GRU is 0.8% better on unseen data when looking at MCC score. For ROC area score the difference is only 0.4%, but for stock market predictions even small difference is significant. For training data set GDF+PCA+GRU have much higher scores than QUE+LOG. Possibly it is due a few stocks for which GDF+PCA+GRU is overfitting, especially because the median of difference for MCC score on training data set is smaller than 2%.

Table 7.21: Difference for scores between GDF+PCA+GRU and QUE+LOG

	MCC score difference			ROC area score difference		
	Training	Validation	Test	Training	Validation	Test
mean	0.026035	0.009765	0.008967	0.015448	0.004777	0.004842
std	0.038195	0.011648	0.009267	0.024666	0.005779	0.006444
min	0.001380	-0.003757	0.001739	0.000682	-0.003556	-0.002432
median	0.018009	0.007519	0.007546	0.009857	0.003981	0.004065
max	0.160418	0.044135	0.031753	0.102657	0.019237	0.020187

On Figure 7.24 we compare scores obtained by GDF+PCA+GRU and QUE+LOG. For all stocks GDF+PCA+GRU has higher scores except for 3459 ROC area score. For both approaches ROC area scores are very similar for most of the stocks, that is why the mean ROC area score difference is only 0.4%.

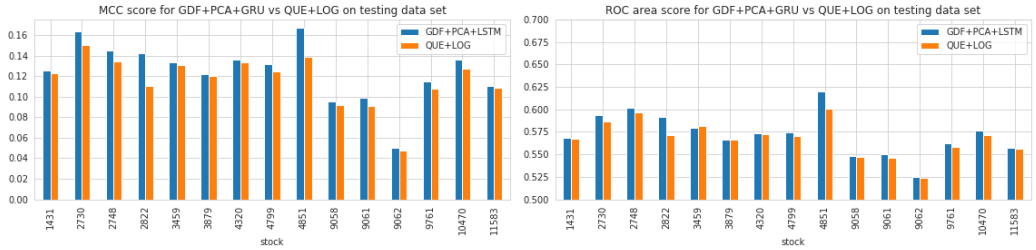

Figure 7.24: Bar plot of partial scores for for GDF+PCA+GRU and QUE+LOG

7.5.5.2 McNemar Test

We perform McNemar test to compare if GDF+PCA+GRU and QUE+LOG are similar classifiers. For each stock *null hypothesis* during McNemar test assumes that GDF+PCA+GRU and QUE+LOG have similar error rate. We calculate χ^2 statistics and corresponding p-values, which are shown on Figure 7.25. For stocks 4320, 9061, 11583, 4799, 9062 we cannot reject *null hypothesis* from McNemar test, because the p-values are above threshold 0.1.

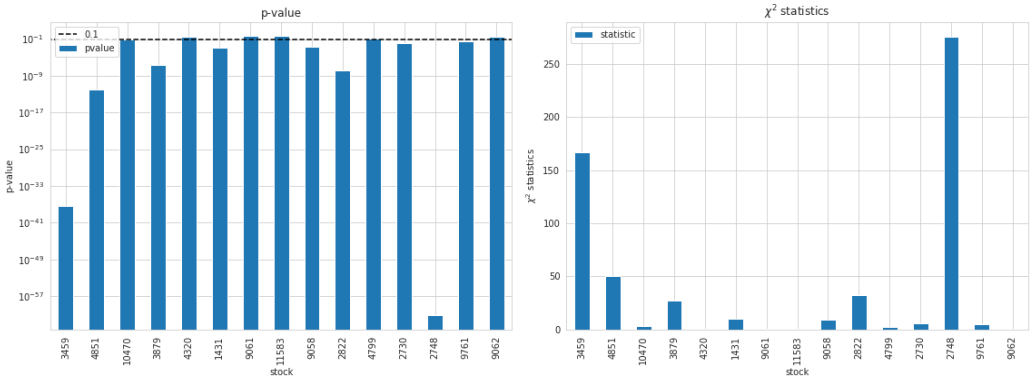

Figure 7.25: p-value and χ^2 statistics for each stock

For more than half of the stocks GDF+PCA+GRU is different than QUE+LOG. Also on average it has better scores on unseen data, even if it has issues with overfitting. Therefore we conclude that GDF+PCA+GRU approach is better than QUE+LOG, but requires more attention regarding hyper-parameters. Unfortunately the parameters have to be tuned for each stock separately, where QUE+LOG gives good results without any tuning.

7.5.6 Summary

In this section we presented results for four classifiers: Logistic Regression, MLP, LSTM and GRU on GDF+PCA features. GDF+PCA+LOG was not better than baseline QUE+LOG, most probably because of *dimensionality curse* of Logistic Regression algorithm. Using MLP slightly improved the predictions. Using RNN algorithms: LSTM and GRU yield the best results – they are better than QUE+LOG by 1% and 0.8% respectively on unseen data. This also implies that using GDF+PCA features can bring improvement if more complex classification algorithm is used. On the other hand GDF features require more work with data preparation. Also GDF+PCA+LSTM and GDF+PCA+GRU approaches require a lot hyper-parameter tuning, which is time consuming.

For GDF+PCA+LSTM and GDF+PCA+GRU approaches McNemar test results did not allow to reject McNemar's test *null hypothesis* for all stocks. This does not necessarily mean that Null Hypothesis is true. The test may fail due to variation of differences between training and testing data sets for these stocks. Also McNemar test is based on accuracy measure, therefore it is prone to class imbalance. In our data set we have much more negative samples than positive ones, which might skew the result of McNemar test. That is why we rely more on MCC and ROC area score for comparison of methods for price movement prediction shown in this thesis.

Chapter 8

Conclusions

In this thesis we approached solving financial problem using Machine Learning methods and proved that they are suitable for stock market price movement prediction. Since Limit Order Books used for executing trades on stock market have not uniform shape, we needed to choose the meaningful representation of the data. We set our baseline on **queue imbalance** feature used by [Gould and Bonart, 2015] and we successfully introduced new features: **previous queue imbalance** and normalized Limit Order Book filtered using Gaussian Density Filter and reduced by Principal Component Analysis. We used these features as inputs to multiple Machine Learning algorithms to predict direction of next **mid price** movement.

Using Gaussian Density Filter to extract information about shape of Limit Order Book proved to be valuable addition to **queue imbalance** feature, which improved quality of predictions by used classification algorithms. We noticed that approach with GDF+PCA+LSTM gave 1% improvement on unseen data than QUE+LOG. That improvement may not seem to matter much, but it is enough to be significant, because stock market data is highly unpredictable [Gould et al., 2010]. Since it is proven that LSTM algorithm can be applied to stock market forecasting problem [Chung and Shin, 2018] [Li et al., 2018] [Fischer and Krauss, 2018] it was worth to try more novel RNN algorithm – GRU [Chung et al., 2014]. GRU algorithm is suitable for time series forecasting with comparable performance to LSTM, but lower complexity [Petneházi, 2019]. GDF+PCA+GRU approach had slightly shorter computing time than GDF+PCA+LSTM with only slightly worse results. When we consider training time and number of hyper-parameters to tune, QUE+LOG still provides a good trade-off between required effort and quality of predictions.

Out of the approaches we tried, some of them were not effective, but they allowed us to discover different ways of solving the problem. Introducing **previous queue imbalance** feature barely increased prediction power of used classifiers, however, it led to an idea of using more sophisticated algorithms, which can utilize information from sequences in time series. Using Logistic Regression and MLP

algorithms on GDF features only proved that more complex Machine Learning algorithm is needed to handle the subtleties of filtered Limit Order Book data. Also using SVM over Logistic Regression on **queue imbalance** feature only increased computation time, but did not improve the quality of the result.

During the research we encountered and solved multiple problems. The first issue we needed to cope with was how to make classification algorithms perform well on unbalanced data-sets in which classes ratio is different for every stock. We overcame that issue by using weighted versions of loss functions in the algorithms, instead of using sampling solution incorporated by [Gould and Bonart, 2015]. Validation of time series data is problematic, because the data cannot be randomly sampled without loosing time dependencies. However, our hyper-parameter tuning methodology using 5-Fold Forward Cross-Validation allowed to choose classifiers, which generalize well on unseen data.

Similarly as many other researches, we found that the most difficult issue was the stock market data itself. We formulated the idea of eliminating the noise from data by applying Gaussian Density Filters. Because features obtained using that filter require more complex algorithms, the computation time became longer. Nonetheless, we successfully decreased calculation time by reducing the number of dimensions in the data using PCA.

In order to make use of GDF features, we combined them with **queue imbalance**, yet it would be also possible to introduce information about **mid price** directly as additional order in LOB on which we apply GDF. To make the orders that are close to the **mid price** more relevant, it could be a good idea to assign weights to GDF features depending on how far they are from the **mid price**. The disadvantage of that solution might be that it would introduce new set of hyper-parameters which would need to be adjusted during the training phase.

To reduce LOB dimension we chose PCA, which is a linear algorithm [Bishop, 2006]. Since we did not try any other unsupervised learning algorithm which would be suitable for dimension reduction, one could try PCA with non-linear kernel. It would also be worth to explore more sophisticated non-linear dimensionality reduction techniques such as t-SNE or its novel alternative UMAP [Becht et al., 2018]. Unfavourably, higher complexity of the reduction algorithm could possibly increase the computation time and moreover make the features even harder to interpret.

The techniques showed in this thesis could be applied for online learning to make prediction on actual stock market. For the stock for which we wish to make forecasting, we could train and tune hyper-parameters of the algorithm on the most recent data to perform live predictions. To keep the parameters of the model up-to-date we would need to retrain the algorithm periodically. Fortunately it is easy to verify the quality of predictions on instant, since LOBs are executed with high-frequencies in real time.

Bibliography

[Adhikari and Agrawal, 2013] Adhikari, R. and Agrawal, R. K. (2013). An introductory study on time series modeling and forecasting. *CoRR*, abs/1302.6613.

[Andersen, 2000] Andersen, T. G. (2000). Some reflections on analysis of high-frequency data. *Journal of Business & Economic Statistics*, 18(2):146–153.

[Batuwita and Palade, 2013] Batuwita, R. and Palade, V. (2013). *Class Imbalance Learning Methods for Support Vector Machines*, volume 83, pages 83–99.

[Becht et al., 2018] Becht, E., Dutertre, C.-A., Kwok, I. W., Ng, L. G., Ginhoux, F., and Newell, E. W. (2018). Evaluation of umap as an alternative to t-sne for single-cell data. *bioRxiv*.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[Boughorbel et al., 2017] Boughorbel, S., Jarray, F., and El-Anbari, M. (2017). Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PLOS ONE*, 12(6):1–17.

[Bradley, 1997] Bradley, A. P. (1997). The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recogn.*, 30(7):1145–1159.

[Brockwell and Davis, 2002] Brockwell, P. J. and Davis, R. A. (2002). *Introduction to time series and forecasting*. Springer-Verlag Inc, Berlin; New York.

[Chung and Shin, 2018] Chung, H. and Shin, K.-s. (2018). Genetic algorithm-optimized long short-term memory network for stock market prediction. *Sustainability*, 10(10).

[Chung et al., 2014] Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[Dietterich, 1998] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923.

[Domingos, 2012] Domingos, P. (2012). A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87.

[Engle et al., 2012] Engle, Nguyen, G. R., Fleming, M. J., and Ghysels, E. (2012). Liquidity and volatility in the U.S. treasury market. Staff Reports 590, Federal Reserve Bank of New York.

[Fischer and Krauss, 2018] Fischer, T. and Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654 – 669.

[Gerig, 2012] Gerig, A. (2012). High-frequency trading synchronizes prices in financial markets. *SSRN Electronic Journal*.

[Gould and Bonart, 2015] Gould, M. D. and Bonart, J. (2015). Queue Imbalance as a One-Tick-Ahead Price Predictor in a Limit Order Book. *arXiv e-prints*, page arXiv:1512.03492.

[Gould et al., 2010] Gould, M. D., Porter, M. A., Williams, S., McDonald, M., Fenn, D. J., and Howison, S. D. (2010). Limit Order Books. *arXiv e-prints*, page arXiv:1012.0349.

[Hastie et al., 2001] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[Kercheval and Zhang, 2015] Kercheval, A. N. and Zhang, Y. (2015). Modelling high-frequency limit order book dynamics with support vector machines. *Quantitative Finance*, 15(8):1315–1329.

[King and Zeng, 2001] King, G. and Zeng, L. (2001). Logistic regression in rare events data. *Political Analysis*, 9:137–163.

[Li et al., 2018] Li, H., Shen, Y., and Zhu, Y. (2018). Stock price prediction using attention-based multi-input lstm. In Zhu, J. and Takeuchi, I., editors, *Proceedings of The 10th Asian Conference on Machine Learning*, volume 95 of *Proceedings of Machine Learning Research*, pages 454–469. PMLR.

[Nousi et al., 2018] Nousi, P., Tsantekidis, A., Passalis, N., Ntakaris, A., Kanniainen, J., Tefas, A., Gabbouj, M., and Iosifidis, A. (2018). Machine learning for forecasting mid price movement using limit order book data. *CoRR*, abs/1809.07861.

[Ntakaris et al., 2018] Ntakaris, A., Kanniainen, J., Gabbouj, M., and Iosifidis, A. (2018). Mid-price prediction based on machine learning methods with technical and quantitative indicators.

[Ntakaris et al., 2017] Ntakaris, A., Magris, M., Kanniainen, J., Gabbouj, M., and Iosifidis, A. (2017). Benchmark dataset for mid-price prediction of limit order book data.

[Petneházi, 2019] Petneházi, G. (2019). Recurrent neural networks for time series forecasting. *CoRR*, abs/1901.00069.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[Stapor, 2017] Stapor, K. (2017). Evaluating and comparing classifiers: Review, some recommendations and limitations. pages 12–21.

[Syarif et al., 2016] Syarif, I., Prugel-Bennett, A., and Wills, G. (2016). Svm parameter optimization using grid search and genetic algorithm to improve classification performance. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 14:1502.

[Tsantekidis et al., 2017] Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., and Iosifidis, A. (2017). Forecasting stock prices from the limit order book using convolutional neural networks. pages 7–12.

A.1 Activation Functions

A.1.1 ReLU

ReLU function is given by:

$$f(x) = \max(0, x)$$

.

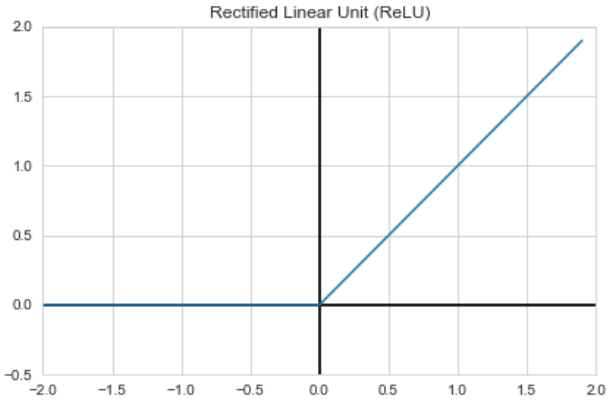

Figure 1: ReLU function

A.1.2 Sigmoid Function

Sigmoid function is given by:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

.

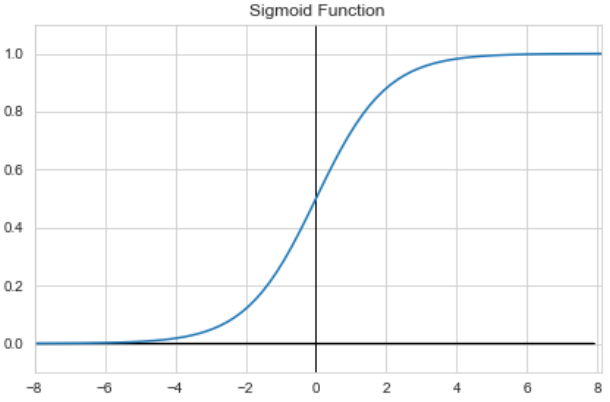

Figure 2: Sigmoid function

A.1.3 Tanh Function

Tanh function is given by:

$$f(x) = tanh(x)$$

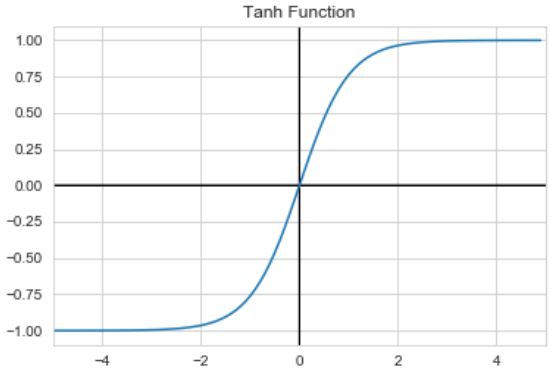

Figure 3: Tanh Function

A.2 Gaussian Distribution

Gaussian Distribution is defined as:

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

where σ^2 is a *variance* and μ is an *expectation*. Gaussian Distribution is a probability distribution [Bishop, 2006], since $\mathcal{N}(x; \mu, \sigma^2) > 0$ and it is normalized:

$$\int_{-\infty}^{\infty} \mathcal{N}(x; \mu, \sigma^2) dx = 1$$

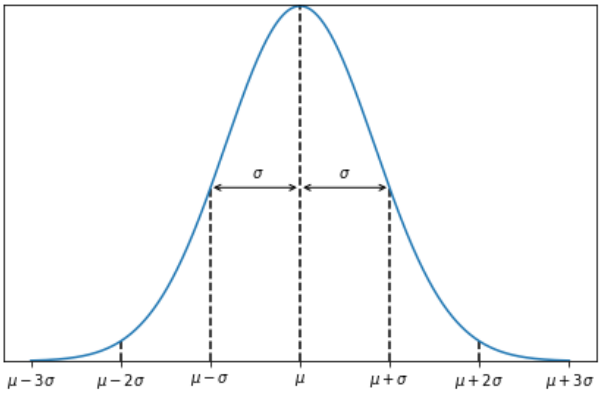

Figure 4: Gaussian Distribution

A.3 Lagrange Multipliers

Lagrange Multipliers are used to find stationary points (local minima or maxima) of a function subject to one or more constraints [Bishop, 2006].

For a problem of maximizing or minimizing function $f : \mathbb{R}^d \to \mathbb{R}$, $f(\boldsymbol{x})$ subject to constraint $g : \mathbb{R}^d \to \mathbb{R}$, $g(\boldsymbol{x}) = 0$ we introduce parameter $\lambda \neq 0$ called **Lagrange Multiplier** used in Lagrangian Function:

$$L(x, \lambda) = \nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x})$$

.

To find local maxima or minima of function f(x) subject to constraint $g(x) = 0$ we need to find stationary points of corresponding Lagrangian Function $L(x, \lambda)$ with respect to both $\boldsymbol{x}$ and λ. Therefore we need to solve $\nabla L(\boldsymbol{x}, \lambda) = 0$, which leads to solving $d + 1$ equations, from which we can eliminate λ [Bishop, 2006].

If we are looking for local maxima of a function f, but due to inequality constraints $g(\boldsymbol{x}) \geq 0$, we need to optimize Lagrange function $L(\boldsymbol{x}, \lambda) = \nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ and λ subject to *Karush-Kuhn-Tucker* (KKT) conditions [Bishop, 2006]:

$$g(\boldsymbol{x}) \geq 0$$
$$\lambda g(\boldsymbol{x}) \geq 0$$
$$\lambda \geq 0$$

When there are multiple equality constraints $g_i(\boldsymbol{x}) = 0$ for $i = 1, 2, \ldots, k$ and inequality constraints $h_i(\boldsymbol{x}) \geq 0$ for $i = 1, 2, \ldots, m$, subject which we want to maximize function $f(\boldsymbol{x})$ then the Lagrangian function is given by [Bishop, 2006]:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i=1}^{k} \lambda_i g_i(\boldsymbol{x}) + \sum_{i=1}^{m} \mu_i h_i(\boldsymbol{x})$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are ordered sets of respectively k and m Lagrangian Multipliers. We optimize Lagrangian Function subject to $\mu_k \geq 0$ and $\mu_i h_i(\boldsymbol{x}) = 0$ for $i = 1, 2, \ldots, m$.