UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

FACULTAD DE INGENIERÍA

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

A DEEP LEARNING STATE-BASED MARKET MICROSTRUCTURE APPROACH FOR THE PRICE MOVEMENT PREDICTION TASK

A DISSERTATION SUBMITTED BY

NOEL CHAPARRO PORTILLO

IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER IN COMPUTER ENGINEERING

CHIHUAHUA, CHIH., MEXICO. FEBRUARY 2022

A deep learning state-based market microstructure approach for the price movement prediction task. A thesis submitted by Noel Chaparro Portillo in partial fulfillment of the requirements for the degree of Master in Computer Engineering, has been verified and accepted by:

M.I. Javier González Cantú
Director de la Facultad de Ingeniería

Dr. Alejandro Villalobos Aragón
Secretario de Investigación y Posgrado

M.S.I. Karina Rocío Requena Yáñez
Coordinador Académico

Dr. Jesús Roberto López Santillán
Dissertation Director

February 2022

Examination committee:

Dr. Jesús Roberto López Santillán
Dr. Manuel Montes y Gómez
Dr. Luis Carlos González Gurrola
M.I. Ana Lucia Corral Flores

© All rights reserved
Noel Chaparro Portillo
Circuito Universitario Campus II,
Chihuahua, Chih., México.
February 2022

UNIVERSIDAD AUTÓNOMA DE
CHIHUAHUA

22 de febrero de 2022.

ING. NOEL CHAPARRO PORTILLO
Presente

En atención a su solicitud relativa al trabajo de tesis para obtener el grado de Maestro en Ingeniería en Computación, nos es grato transcribirle el tema aprobado por esta Dirección, propuesto y dirigido por el director **Dr. Jesús Roberto López Santillán** para que lo desarrolle como tesis, con el título **"A DEEP LEARNING STATE-BASED MARKET MICROSTRUCTURE APPROACH FOR THE PRICE MOVEMENT PREDICTION TASK"**.

Contents

Dedication

Acknowledgments

Abstract

Contents

List of Figures

List of Tables

1: Introduction

- 1.1 Trading
- 1.2 High-frequency trading, a new trading paradigm
- 1.3 Problem statement
- 1.4 A novel LOB representation
- 1.5 Research questions
- 1.6 General objetives
- 1.7 Contributions
- 1.8 Thesis outline

2: Literature review

- 2.1 Market predictability
- 2.2 The Price movement prediction task

FACULTAD DE INGENIERÍA
Circuito No.1, Campus Universitario 2
Chihuahua, Chih., México. C.P. 31125
Tel. (614) 442-95-00
www.fing.uach.mx

3: Theoretical framework

3.1 Artificial intelligence

3.2 Machine learning

3.3 Deep learning

3.4 Loss function

3.5 Backpropagation

3.6 Sequence models

3.7 Clustering

3.8 Natural language processing

3.9 Evaluation metrics

3.10 Statistical significance tests

3.11 High frequency trading

3.12 Market microstructure

4: Methodology

4.1 Data collection

4.2 Data preprocessing

4.3 Sequence segmentation

4.4 Task description and data labeling

4.5 Model architecture selection

4.6 Model evaluation

5: The proposed limit order book sequence representation

5.1 Proposed limit order book; stationary version

5.2 Proposed limit order book; tokenized version

6: Experimental setup

6.1 Input data

6.2 Experimental design

6.3 Software and hardware

6.4 Training setup

UNIVERSIDAD AUTÓNOMA DE
CHIHUAHUA

7: Results

7.1 Experimental results

7.2 Training monitoring results

7.3 Results of experiment with the highest performance

8: Discussion, conclusions and future work

8.1 Discussion

8.2 Conclusions and future work

Referencias

Solicitamos a Usted tomar nota de que el título del trabajo se imprima en lugar visible de los ejemplares de las tesis.

A T E N T A M E N T E

"Naturam subiecit aliis"

EL DIRECTOR | **EL SECRETARIO DE INVESTIGACIÓN Y POSGRADO**

FACULTAD DE INGENIERÍA U.A.CH.

M.I. JAVIER GONZÁLEZ CANTÚ | **DR. ALEJANDRO VILLALOBOS ARAGÓN**

DIRECCIÓN

FACULTAD DE INGENIERÍA
Circuito No.1, Campus Universitario 2
Chihuahua, Chih., México. C.P. 31125
Tel. (614) 442-95-00
www.fing.uach.mx

Dedication

This thesis is dedicated to my parents who have always supported me. To my wife Nayeli who was always the first person to hear and approve every new idea. To my little daughter Ximena who was in charge of getting me to take "scheduled" breaks to play for the last two years. And to my future son Eugenio who wanted me to finish all this work before he steps by this world.

Acknowledgments

Foremost, I would like to thank God who has always been my guide. To the *Consejo Nacional de Ciencia y Tecnología* (CONACYT) from which I received a scholarship. Thanks to the *Universidad Autónoma de Chihuahua* (UACH), which gave me the opportunity of becoming part of this project and from which I received funding for acquiring the dataset used in this research.

I would like to express my sincere gratitude to my director Dr. Jesús Roberto López Santillán and to my co-director of thesis Dr. Manuel Montes y Gómez who were constantly enriching this work with their expertise.

Special thanks to the *Instituto Nacional de Astrofísica, Óptica y Electrónica* (INAOE) and particularly to Dr. Manuel Montes y Gómez who interceded so that we were given access to the INAOE's supercomputer lab. It allowed us to greatly decreased the training times and made in many ways this thesis possible.

Thanks to my professors for the knowledge I received. And lastly, I would like to thank Alberto, Leonardo and Daniel who were my classmates (now my friends) during this masters and made the process a lot more fun.

Abstract

The advancement in technology has lead to an evolution in how trading is done in the different financial markets. For instance, the access to almost instantaneous market data is now possible with the reduction of latency. The use of computers in trading made possible to reduce execution times drastically, and also allowed to create programs for automatic trading.

With all of these changes, a shift towards smaller investment horizons has been observed in the past few decades, being the extreme something known as High Frequency Trading (HFT). In HFT, such horizons are typically just a very small fraction of a second.

This shift in the trading paradigm is driven by high-frequency market microstructure data: the Limit Order Book (LOB) and the order flow. However, analyzing such information is not just a matter of speed, but a problem that presents some nonlinearities from a high dimensional space.

In this thesis, a deep learning approach is taken in order to effectively predict the mid-price movement direction at a future short-term horizon for the cryptocurrency market. Specifically, the work is focused on the biggest cryptocurrency to date as per market capitalization: Bitcoin.

High frequency transaction data from Coinbase is used to reconstruct the LOB for the pair BTC/USD from August 25th to October 25th, 2020. Two novel state-based representations of the data are introduced and compared against a simpler representation that applies a well known standardization method over the raw data. Long Short-Term Memory (LSTM) and Transformer models are trained on these representations for the task at hand.

Best prediction metrics are obtained by using some of the the representations proposed. A test for statistical significance is carried out showing that differences in the results are statistically significant.

The implications of these results directly challenge some of the well known hypothesis that state that market prices behave like a random walk, and that it is impossible to find predictable patterns that allow predicting the future price direction. Although out of scope of this dissertation, best trained models can potentially be integrated into a HFT strategy as signal generators.

Contents

1 Introduction **1**
1.1 Trading . . . 1
1.2 High-frequency trading, a new trading paradigm . . . 2
1.3 Problem statement . . . 3
1.4 A novel LOB representation . . . 4
1.5 Research questions . . . 5
1.6 General objectives . . . 5
1.6.1 Specific objectives . . . 5
1.7 Contributions . . . 6
1.8 Thesis outline . . . 6

2 Literature review **7**
2.1 Market predictability . . . 7
2.2 The price movement prediction task . . . 9

3 Theoretical framework **17**
3.1 Artificial intelligence . . . 17
3.2 Machine learning . . . 18
3.2.1 Machine learning types . . . 18
3.2.1.1 Supervised learning . . . 18
3.2.1.2 Unsupervised learning . . . 19
3.2.1.3 Reinforcement learning . . . 19
3.2.2 The perceptron . . . 20
3.2.3 Activation function . . . 21
3.2.4 Multilayer neural network . . . 21
3.3 Deep learning . . . 22
3.4 Loss function . . . 23
3.4.1 Binary cross-entropy . . . 23
3.4.2 Categorical cross-entropy . . . 24
3.4.3 Mean squared error . . . 24
3.5 Backpropagation . . . 24
3.6 Sequence models . . . 25
3.6.1 Recurrent neural networks . . . 26

3.6.1.1 Long short-term memory . . . 27
3.6.2 Transformer . . . 30
3.7 Clustering . . . 32
3.7.1 k-means . . . 33
3.8 Natural language processing . . . 34
3.8.1 Vector representation . . . 35
3.8.1.1 One-Hot encoding . . . 35
3.8.1.2 Word embeddings . . . 35
3.9 Evaluation metrics . . . 37
3.9.1 Accuracy . . . 38
3.9.2 Precision . . . 39
3.9.3 Recall . . . 39
3.9.4 F1-Score . . . 39
3.9.5 From binary to multi-class . . . 39
3.10 Statistical significance tests . . . 40
3.10.1 McNemar's test . . . 41
3.11 High frequency trading . . . 42
3.12 Market microstructure . . . 43
3.12.1 Limit order book . . . 43

4 Methodology **46**
4.1 Data collection . . . 46
4.1.1 The raw dataset . . . 47
4.1.2 Limit order book reconstruction . . . 49
4.2 Data preprocessing . . . 51
4.3 Sequence segmentation . . . 52
4.4 Task description and data labeling . . . 53
4.5 Model architecture selection . . . 55
4.5.1 LSTM-based architecture . . . 56
4.5.2 Transformer-based architecture . . . 57
4.6 Model evaluation . . . 58

5 The proposed limit order book sequence representation **59**
5.1 Proposed limit order book: stationary version . . . 59
5.2 Proposed limit order book: tokenized version . . . 63
5.2.1 Proposed sequence contrast score . . . 64
5.2.2 k-means clustering . . . 66

6 Experimental setup **68**
6.1 Input data . . . 68
6.1.1 Data feeding . . . 69
6.2 Experimental design . . . 70
6.3 Software and hardware . . . 71
6.4 Training setup . . . 72

6.4.1 Objective function 72
6.4.2 Optimizer 72
6.4.3 Hyperparameter tuning 73
6.4.4 Model selection 73

7 Results **74**
7.1 Experimental results 74
7.1.1 Statistical test 77
7.2 Training monitoring results 78
7.3 Results of experiment with the highest performance 79

8 Discussion, conclusions and future work **82**
8.1 Discussion 82
8.2 Conclusions and future work 88

References **91**

List of Figures

3.1 The Perceptron 20
3.2 Common Activation Functions 21
3.3 Multilayer Perceptron 22
3.4 Schematic diagram of a basic RNN 27
3.5 LSTM Cell 28
3.6 The Transformer architecture 31
3.7 Evolution of k-means algorithm 34
3.8 Word Embeddings (2D projection) 36
3.9 Word2Vec architectures 36
3.10 Relationships learned with word embeddings 37
3.11 Contingency table 41
3.12 Schematic representation of a LOB 44

4.1 General overview of the methodology 47
4.2 Events file provided by Kaiko 48
4.3 Event types, source: https://docs.cloud.coinbase.com 48
4.4 Transformer and LSTM architectures 55

5.1 Sequence representation processes pipeline 60
5.2 Sequence Contrast Scores for some k values 66
5.3 Example of a tokenized sequence 67

7.1 F1-Score values for different horizons 76
7.2 McNemar's Test results 78
7.3 McNemar's Test results for additional experiments 78
7.4 Confusion Matrix of best model 80
7.5 Movement direction predictions plot 80
7.6 Learning curves for best model 81

8.1 Cramér's V association rule results 86
8.2 Probability distribution of the F1-Score on a test day 87

List of Tables

4.1 Features list 51

6.1 Class distribution of dataset splits 69
6.2 Models' parameters 73

7.1 Experimental results for a prediction horizon $k = 10$ 75
7.2 Experimental results for a prediction horizon $k = 20$ 75
7.3 Experimental results for a prediction horizon $k = 50$ 75
7.4 Experimental results for a prediction horizon $k = 100$ 76
7.5 Additional experimental results for a prediction horizon $k = 20$ . . . 77
7.6 Number of epochs trained 79
7.7 Classification report for best model 79

Chapter 1

Introduction

1.1 Trading

Trading has existed for as long as humans have needed to exchange goods. First forms of trading date back to ancient times in the form of bartering, where services or goods like food, weapons, materials, etc., were traded without the use of money. As time went by, this type of trading evolved to the concept of financial markets as we know them today.

From an economical perspective, a financial market is a place that brings together those who have money to invest with those who need some funding [1]. In this sense, investors go to a financial market in order to exchange financial instruments [2]. Some examples are the stock market, the bond market, the commodities market, and more recently the cryptocurrency market, where digital currencies are traded [3]. The driving force behind financial markets is the underlying motivation that exists in investors to earn a return on their investments, either from a yield or capital gains [4].

Financial markets started as physical places where participants interacted face-to-face with others. Then, market dynamics changed with the appearance of the

Electronic Communication Network (ECN) (an automatic matching engine for buys and sells) in 1969, and the creation of the first electronic stock market in 1971, NASDAQ (National Association of Securities Dealers Automated Quotation), that allowed instantaneous communications between participants, and the update of quotations as new information was arriving [5].

Since then, the electronic trading received a big boost, allowing the establishment of the automatic trading (also called algorithmic trading) [6], and the transition from a quote-driven market to an order-driven market [7]. Now financial markets are more of a virtual place, where traders does not even need to meet in order for a trade to happen.

1.2 High-frequency trading, a new trading paradigm

As the electronic trading increased, the order processing time decreased, leading to high volume data, and the surge of High-frequency trading (HFT) [8]. HFT is a type of algorithmic trading that uses sophisticated technological tools, high volume of orders and extremely short holding periods (milliseconds down to nanoseconds) [6].

In longer time horizons, fundamentals around an asset in question are studied in an effort to predict the future's asset price or direction. On the other hand, HFT is microstructure-based, i.e., the trading information itself is used, e.g., the order flow and the structure of the Limit Order Book (LOB) [9]. The LOB is simply a table-like registry where all incoming limit orders (orders that are not fulfilled immediately) are recorded [10].

Research around high frequency data can be grouped into statistical modeling and machine learning based models. However, statistical models impose fixed rules and make predefined assumptions, hardly reflecting the real mechanics behind [8].

On the other hand, Machine Learning (ML) or Deep Learning (DL) models have been implemented to better capture inherent relationships on the data [11, 12, 13].

On several research using high-frequency data, DL models have been outperforming simpler linear models, suggesting that non-linear relationships are present on this type of data [14, 15].

1.3 Problem statement

Recurring events and patterns are exploited from high-frequency data in order to spot predictable short-term responses. These are in turn incorporated into a trading strategy to generate profits.

Different techniques have been proposed to represent the high-frequency data in order to allow a ML/DL model to extract information and predict the price movement direction of an asset at a future time horizon. Before feeding the data to the models, most research have only applied common preprocessing techniques like z-score, min-max, etc. [16, 17, 18, 19, 20, 21, 22, 23].

Others have applied kernels to determine a non-linear mapping from the raw data, then, this mapped data was fed to a classifier [24]. Some feature engineering has been also attempted in order to extract time-sensitive and time-insensitive features [8, 25], or create stationary features [26].

Market data has been input to a Convolutional Neural Network (CNN) on different channels (Order Book, Trades, Orders, Deletions) [27]. A Bag-of-Features (BoF) was applied in [28, 29] to extract short-term and long-term representations of the data. Convolutional filters have been used as feature extractors for capturing the spatial nature of the LOB [30].

Despite the many different techniques being utilized on previous research, no efforts have been made as to understand the LOB dynamics as a transition of scale

invariant LOB states. Nevertheless, the transition between LOB states was studied in [31], and it was statistically proved that such transitions are not random.

1.4 A novel LOB representation

Framing the LOB high-frequency data as a sequence of states, would allow to identify where exactly in the state space, a particular LOB state is located. By doing so, future transitions could be learned conditional on the actual and previous states.

In the present thesis, a novel preprocessing strategy applied to high-frequency data is presented. Two different representations based on the idea of shaping the high frequency data contained in the LOB as a collection of states are introduced.

The first LOB representation is a state space representation of scale invariant features for prices and volumes. While the second, explores the idea of simplifying the LOB state space size by clustering the space, then, reduce the state space dimensionality onto a lower dimension by applying a commonly used technique borrowed from the Natural Language Processing (NLP) area, Word Embeddings (WEs).

Effectiveness of the proposed representations is evaluated on the mid-price movement prediction task by using some of the most performing sequence models: the Long Short-Term Memory (LSTM) [32] and the Transformer [33]. Cryptocurrency market data is used for this task. Finally, the value added when incorporating data augmentation and information related to order flow and transaction data history is evaluated as well.

1.5 Research questions

This thesis focus is to answer the following research questions:

- Does a scale invariant state-based representation of the high-frequency data provides improved results on the mid-price movement prediction task?
- Do the intention and perception of market participants can be captured and synthesized, through a modeling of the high-frequency data using a WE representation and perform competitively on prediction tasks?
- Is the cryptocurrency market predictable at very short-time horizons?

1.6 General objectives

- To create a DL scale invariant state-based model for decision making on the mid-price movement prediction task for the cryptocurrency market.
- To implement a novel representation for high-frequency data for modeling high-frequency dynamics.

1.6.1 Specific objectives

- To reconstruct market event order data flow into historical sequential LOB and order flow representations.
- To design a preprocessing framework for a scale invariant state-based high-frequency data representation.
- To design a suitable DL architecture for the mid-price prediction task.

1.7 Contributions

The main contributions of the present thesis are as follows:

- A novel approach for representing high-frequency data for modeling market dynamics tasks.
- A novel quality measure for measuring clustering effects on multivariate sequential data, known as the Sequence Contrast Score (SCS).
- An empirical study of market predictability at very short-time horizons on cryptocurrency markets.
- A large ready-to-use high-frequency dataset.

1.8 Thesis outline

This thesis is organized as follows. Firstly, in **Chapter 2**, the current knowledge about predictability in markets and previous research on the mid-price prediction task are presented. **Chapter 3** includes an overview of the key concepts, covering both market microstructure as well as ML/DL concepts. **Chapter 4** provides a description of the methodologies used from the data acquisition till model evaluation.

Then, in **Chapter 5** the proposed high-frequency data representations are introduced. In **Chapter 6**, a detailed description of the experiments performed is presented as well as some details on the data used. After that, in **Chapter 7**, results are evaluated and first comparisons are made. Finally, in **Chapter 8**, an in-depth exploration of the results is carried out, final conclusions about the research are drawn, and at the end, future research directions are considered.

Chapter 2

Literature review

In this chapter a literature review is presented, it covers the current knowledge about the market predictability and this thesis main task; the prediction of the market price direction and the relevant research around it. By the end of the chapter, the reader will have a better understanding on how the problem has been addressed, and how this research proposes a novel way of solving it.

2.1 Market predictability

Several hypotheses have been suggested claiming that an asset price or direction behave like a random walk, and thus, it cannot be predicted on the basis of past history [34, 35].

The Efficient Market Hypothesis (EMH) [36], states that asset prices fully reflect all available information and expectations, so current prices are the best approximation of a company's intrinsic value. In other words, such implication would mean that it is impossible for a market participant to consistently beat the market. However, many empirical studies have shown that financial markets are, to some extent, predictable [37]. From a scientific point of view, in the work presented by Campbell

and Yogo in [38], it was found predictability evidence in stock market returns.

In time series, a forecast consists of making predictions of future values based on information collected up to the present [39]. Several forecasting techniques have been developed, from which, two main approaches can be distinguished: technical analysis and fundamental Analysis.

The first approach focuses on studying historical data, mainly price and volume. It focuses on finding trading opportunities by identifying statistical trends or price patterns. This information is mainly extracted from charts [40]. The second approach focuses on estimating the intrinsic value of an asset at any point in time [35]. It makes so, by studying fundamentals such as: past records of a company, income statements, balance sheets, valuation, investment plans, future growth, return on equity, interest rates, overall state of the economy, news, etc [41]. Potential gains are exploited whenever the estimated intrinsic value deviates from its current market price.

A different categorization approach for market prediction techniques is about its final output: price prediction or trend prediction. Depending on this output, the problem could be categorized as a regression or as a classification problem. According to Sezer et. al in [12], it was concluded that researchers considered the direction of the price movement as a more crucial study area compared to the price itself.

As of today, published research has not come to an agreement, and despite the implications of the published hypotheses not supporting market's predictability, the scientific community has proposed different techniques to predict the financial markets.

2.2 The price movement prediction task

Financial time series forecasting is undoubtedly the most popular application in finance and in the investment industry. Not just because it could yield some profits, but also because of the intellectual challenge associated with it.

There have been many different techniques applied to time series, ranging from classical time series forecasting techniques to ML, and recently, DL techniques [42, 43, 44, 37, 11, 12, 45, 46, 47, 48]. In this review however, a focus is made on those research that use high frequency-data and some ML or DL technique for predicting the future price direction of the market.

Few studies meet these criteria, such is the case of [24], where the authors proposed the use of Fisher Kernels along with the SimpleMKL technique. These kernels are incorporated into a Support Vector Machine (SVM) model to predict the direction of the price movement for the EUR/USD currency pair over multiple time horizons. This was the first published research paper that used volumes from a LOB as a predictor for making market forecasts.

Another SVM based approach is the one presented by Kercheval & Zhang in [8]. In this work, a framework to automate the prediction process for a LOB is proposed. A collection of attributes that can be derived directly from the LOB and the message book is presented. Those attributes were divided into three sets: basic, time-insensitive and time-sensitive. Results showed that their multi-class SVM model significantly outperforms a ZeroR classifier for the mid-price metric.

Their work was the first to demonstrate the effectiveness of using a multi-class SVM on equity LOB forecasting. However, it could be noted that just 1500 data points were used in the training phase, even though they randomly sampled 1500 data points datasets for training several times, the dataset was very limited.

An extension of the previous work was made by Han et. al. in [25]. Using the same set of features as in [8], a Random Forest (RF) model outperformed a multi-

class SVM in terms of accuracy for the mid-price prediction metric.

The vast amount of information contained in the LOB, and the dynamics found in transitions between time steps leave without a doubt repeated patterns. These patterns, if detected correctly, could improve the predictions. Convolutional Neural Networks (CNNs) are well known for being very good at extracting such features from the input data.

Doering et. al. in [27] trained a CNN based upon the architecture of CafeNet and GoogleNet. Raw recorded market messages from the London Stock Exchange (Barclays PLC stock) were used to reconstruct the LOB for the first 30 levels. Three more data representations were proposed focused on the order flow: recently happened trades, incoming buy and sell orders, and transmitted order deletions. These four data representations were used as different input channels for the model. The architecture proposed performed significantly better than a random classifier. It was also observed that combining all input channels performed better than just using a subset of them.

In [20], authors investigated the CNN's capacity for predicting the mid-price movement of 5 Nordic stocks. The results were compared against a Linear SVM and a Multilayer Perceptron (MLP) model. It was concluded that CNNs are better suited for capturing the sequential nature of the LOB as well as the microstructure of the market. The dataset is relatively larger than those used in previous research, more than 4 million data points were used, giving a better estimate in the studied metrics.

Financial time series are a type of sequential data, the dynamics of the LOB itself represent a multivariate sequential type of data. Some models like Recurrent Neural Networks (RNNs) or the improved LSTM model are better suited for dealing with sequential data.

Dixon in [49], used a RNN model to predict the next event price-flip as a tool

for reducing the likelihood of adverse price selection. As in [14], a spatio-temporal representation of the LOB data was used along with the history of market orders, similar to what Kercheval & Zhang did on [8]. RNNs resulted to be suitable for LOB sequential data, however, the model struggled when correctly classifying the up or down movements. This might indicate that it is even harder to predict the next event price-flip, and so, very short event horizons as compared to relatively longer event horizons.

Similar to [20], a comparison of a LSTM model against a Linear SVM and a MLP performance was studied in [21]. The final reported metrics showed that it is far easier for the LSTM models to get better results as compared with simpler models like a Linear SVM and MLP. This is because these models do not natively handle sequential data as compared to a LSTM. By using a LSTM, a significant improvement is seen when comparing the performance of the CNN in [20] and the LSTM on [21] for all the metrics reported.

New architectures have been proposed to better deal with the spatial nature of the LOB. Sirignano in [14], created a new neural network architecture referred as a spatial neural network. The proposed architecture allowed a more effective use of information deep in the LOB. The spatial neural network models the joint distribution of the state of the LOB. The model was trained on a massive amount of LOB data, a total of 489 stocks (from the S&P 500 and the NASDAQ-100), this increased the robustness of the results. The out-of-sample validation showed that the spatial neural network has a lower error and higher accuracy when compared to a standard neural network. It also outperformed the logistic regression model and the naive empirical model.

With the surge of interest in applying ML/DL to high-frequency data, and the almost null availability of public datasets of such type, the first publicly available benchmark dataset for mid-price prediction was created and published by Ntakaris

et. al. in [16], called the FI-2010 dataset. The data comes from the Nasdaq Nordic stock market for a period of 10 consecutive days. Five stocks from different industry sectors are included. The raw dataset is about 4 million events, however the final representation by the authors led to a dataset of about half a million points. Baseline results were provided by applying a linear and a nonlinear regression models.

It is worth noticing that some research using this dataset as input for their models take each time step as one event, while others use each point in the sequence as a summary of 10 events as it was originally intended by authors. However, results are still being compared, but a fair comparison can not be guaranteed.

Once the benchmark dataset was published [16], more studies and state-of-the-art methodologies were tested. One of the first works that made use of this dataset is the one presented by Tran et. al. in [17], where the authors compared tensor-based machine learning models against those that use a vector-based representation. Experimental results favored those with a tensor representation where time corresponded to one of the tensor orders.

Attention based mechanisms are designed as to allow deep neural networks architectures focus on some important parts of the input while ignoring some others. A new Bilinear Layer (BL) and a Temporal Attention-Augmented Bilinear Layer (TABL) were proposed in [18]. New state-of-the-art performance was obtained on the benchmark dataset [16]. Even better results were obtained than those attained by CNN models in [20], or LSTM models in [21]. Due to the noisy nature of the time series contained in the LOB, it was proved that some microstructure signals can be better discriminated by implementing attention mechanisms in the model.

Some inspiration has been taken from other areas of research like image processing and NLP. The BoF, originally inspired in the Bag-of-Words (BoW) from NLP, was used in [28, 19, 29]. In the work presented by Passalis et. al. in [28], and then further improved by same authors in [29], a Temporal Bag-of-Features (T-BoF) that

extracts short-term and long-term representations of the LOB time series was proposed. Each feature vector was quantized using two previously constructed codebooks. The proposed model gave better performance results compared to several feature learning models. This approach reveals something about the nature of the LOB time series, and it is the fact that even though there is a lot of chaos in the data, some repetitive patterns exists in the data generating process.

In order to give a performance boost to already tested architectures, a hybrid model formed by CNN-LSTM modules was introduced by Zhang et. al. in [30]. The proposed architecture made use of convolutional filters and LSTM modules to capture the spatial structure and the time dependencies on the LOB data. An Inception module is also contained in the architecture as to allow a better automatic feature extraction. Their model was tested on the benchmark dataset [16] and it outperformed all existing state-of-the-art models, including [18]. The authors tested the robustness of ther architecture training on a one year dataset extracted from the London Stock Exchange (LSE) for five of the most liquid stocks. Similar results were obtained. Finally, inspired by the work of Sirignano and Cont in [50], which suggests that universal features can be extracted from the LOBs, the model that was trained on the one year dataset was directly evaluated to some other stocks not part of the training data. Results showed that transfer learning is actually possible, and the hypothesis of universal features is in some way supported.

Most of the research only use asset specific data to solve the problem of the price movement forecast, however, some studies suggest that advantages can be taken by using data not specific to the asset at hand.

The work by Sirignano and Cont in [50] tested a 3-Layer LSTM architecture across 1000 US stocks. Main finding suggests universality in the captured features by the DL model. The next price movement forecast performance was better when using a model trained over several different stocks than those stock-specific trained

models, even for stocks not part of the training sample. Thus, opening doors to the use of transfer learning on finance applications. It was also found out that models are stable not just across stocks and sectors, but also, across time.

A similar experiment setup was proposed in [19] by Nousi et. al. where a hold-out stock (i.e. not part of the training data) was evaluated after having trained on four different stocks. The experiment was executed 5 times, for five different hold-out stocks. Results demonstrated that a model trained on other stocks can still be utilized for making predictions on a stock not seen on the training step.

It is well known that the quality of the representation of a time series is directly reflected on the performance of the model. For LOB time series data, some scaling and normalization techniques have been explored. Tsantekidis et. al. in [26] proposed a new set of stationary features that can be extracted directly from the raw LOB data, i.e., prices and volume. These new features are intended to solve any distribution shift on the series, and also, improve generalization to unseen data. A hybrid model CNN-LSTM was also proposed and compared against their individual counterparts, i.e., CNN or LSTM based models alone. Some simpler models like SVM and MLP were also evaluated for comparison. Best results were achieved by the hybrid model, and the models trained with the proposed stationary features obtained significantly better results than those with just raw data.

Prices at each price level were made stationary by computing the percentage difference against the mid-price. However the proposed volume features are not stationary, they are computed as a cumulative sum over the different levels on the LOB, and could still suffer from shifts. Not to mention that the model can not be directly transferred to some other assets with different volume sizes scales.

An adaptive normalization scheme is proposed by Passalis et. al. in [51] where the best normalization to be used on the raw data was dynamically estimated. The method is fully differentiable and can be trained in an end-to-end fashion while

the rest of the model's parameters are also learned. It was compared against some competitive normalization schemes like: z-score, instance normalization and batch normalization. Results favored the proposed method.

Most of the models studied for predicting the mid-price movement are LOB-driven, i.e., based on features derived from the LOB. Just very few works make use of the order flow. Some examples can be found in [27, 15].

The work by Zhang et. al. in [15] used Market by Order (MBO) data for directly predicting the mid-price, i.e., without the need of reconstructing the LOB. Results showed that LOB-based models slightly outperform those MBO-based models. However, MBO-based models seemed to capture some different and less correlated signals. For this reason, models using both LOB and MBO information were also evaluated, giving in general, better results that their counterparts alone.

In the work mentioned above however, some messages were disregarded for a fair comparison with a ten levels LOB representation. Even though this messages did not impacted those first LOB levels, they might indicate market hesitation in sending orders closer to the mid-price, or some other intention. Thus, the models are trained with missing information in regards of the natural order flow.

Complex RNNs or CNNs had been the state-of-the-art in most applications handling sequential data until the appearance of the Transformer architecture proposed in [33]. The transformer model has set new state-of-the-art results in many applications, given that in general, transformers can be applied to any sequential task. The most popular DL model for financial time series forecasting applications in the last years is the LSTM [12]. However, with the emergence of transformers, this statement could change in the near future.

A transformer based architecture was introduced by Wallbridge in [22] to predict the mid-price direction using the FI-2010 dataset provided by [16]. The model used dilated causal convolution over the LOB data followed by transformer blocks

with masked multi-head self attention layers. The proposed architecture achieved state-of-the-art results on the FI-2010 dataset.

Literature about cryptocurrencies and bitcoin is now vast due to their recent popularity. Some predictive models have been also presented, however, very few deal with high-frequency data for predicting the mid-price direction. In [23], Jha et. al. trained a temporal CNN to solve the mid-price movement prediction task. They used 100ms LOB snapshots from the pair BTC/USD, and considered just the first ten levels to be informational for the predictive model. Results demonstrated strong predictive accuracy on a two seconds horizon, especially for the stationary class.

After this extensive literature review, some key points can be drawn. First, in general, DL models outperform linear models, suggesting the existence of non-linear features living on the LOB time series. Second, there is some prediction capacity in the LOB data, and finally, time series representation is as important as the model selection is. Another fact is that tensor based representations or sequence models are better suited for the mid-price movement prediction task.

High-frequency data is in general difficult to get, and sometimes costly. For this reason, most of the works reviewed used an already preprocessed dataset leaving no room for more advanced preprocessing steps. Besides, most of the datasets used by the works cited on this literature review are a subsampled version of the original data flow from the LOB. The research around this thesis, makes use of the original data flow as it occurred, i.e., no subsampling is made.

Even though, some feature extraction models or preprocessing methods have been explored, no attempt has been made to represent the LOB data in to a finite number of possible states. This alone, addresses the prediction problem from a totally novel perspective. A detailed introduction around this research's proposal would be presented in the chapters to come.

Chapter 3

Theoretical framework

To better understand the research being conducted, key fundamental concepts are presented in the following sections. The main idea behind this research, is the application of DL techniques to a specific financial market data type. Thus, relevant information to ML/DL and market microstructure are to be presented. The contents on this chapter will be giving support throughout all research stages.

3.1 Artificial intelligence

The term Artificial Intelligence (AI), was first used in 1956 by John McCarty during a workshop at Darthmouth College. The main research objective was to find how to build machines what were able of solving problems that until that moment, were considered only being able to be solved by humans [52].

From that moment till now, distinct definitions had been given to the term AI, but to stay with just one of them and still preserve its original nature, the definition as given by [53, p. 350] is presented here, which says the following: "Automatic processing on a computer capable of performing operations that would seem to be exclusively relevant to human intelligence".

3.2 Machine learning

Another key concept that results natural to mention is ML, which is sometimes confused with AI as being the same thing, but they are not. ML is a branch of AI that uses algorithms in order to extract knowledge from the data, and by doing so, being able to take decisions or make predictions without being explicitly programmed.

ML apply concepts from statistics, computer science and AI. Some popular definitions for ML are very well accepted by the community as the one given in [54, p. 2]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". In other words, a machine that is able to improve at some specific task while learning from some input data.

3.2.1 Machine learning types

There are several techniques of ML, however most of them would fall into one of these four categories: supervised, unsupervised, semisupervised and reinforcement learning. These four categories are divided according to the level of human supervision required [55].

3.2.1.1 Supervised learning

In supervised learning, the model receives some input data, usually called training data, and each record or example in the data includes also a label, or expected output. The main purpose of this kind of learning is to learn a good mapping between the input and the output.

Typical problems that can be addressed from this approach include: classification problems, for which the output is a set of finite values, usually called categories, and regression problems, where the output is a continuous numerical value [56].

Semisupervised learning just refers to those algorithms capable of dealing with data that is partially labeled, with the majority of examples being unlabeled.

As the data to be used in this work is labeled, a supervised approach is to be used. Several learning methods are employed for a supervised learning, however, for this particular thesis, a neural network approach is made based on the idea of the perceptron. For this reason, from section 3.2.2 onwards, the information presented is focused on this same approach.

3.2.1.2 Unsupervised learning

On the other hand, the unsupervised learning is a type of learning on which the training data has no labels or expected outputs, i.e., the records are unlabeled. Unlike supervised learning, the models using this approach are intended to learn patterns on the data without an explicit feedback [57].

One of the main tasks that this type of learning solves is clustering, whose objective is to separate or cluster the input data by groups, in such a way, that the examples on each group are very similar between them, and very different with respect to the rest of the clusters [58].

In this research, an unsupervised learning is to be applied in the preprocessing steps when clustering the LOB states to create a tokenized representation of the data. Details will be given in Chapter 5: The proposed limit order book sequence representation.

3.2.1.3 Reinforcement learning

In this context, the learning model is called the agent. This agent observes and extracts information from the environment and take decisions from it. This decision making within the context of reinforcement learning is called policy.

According to the problem being addressed, a reward or penalty is given to the

agent for each action taken, this allows that, as time goes on, based on trial and error the agent learns to make better decisions, thus, maximizing its total reward [59]. On this thesis however, no reinforcement learning approaches are taken.

3.2.2 The perceptron

The simplest neural network someone can find is called the perceptron, also known as the basic processing element [59]. It was popularized by Frank Rosenblatt in 1957. It is a computational model formed by a single neuron, and follows a feed-forward model in the sense that some input values are injected in the neuron, processed and just one output value is returned. A graphical representation is shown in Figure 3.1.

Figure 3.1: The Perceptron [59]

Here $x_j, w_j, y \in \mathbb{R}$, which are the inputs, weights and output respectively. In the image above y is simply represented as $y = \sum_{j=1}^{d} w_j x_j + w_0$. However in a more general case the output y is calculated by a function called activation function ϕ, which is a function of the weighted sum or dot product $W^T X$, where $\mathbf{w} = [\mathbf{w_0}, \mathbf{w_1}, ..., \mathbf{w_d}]^\mathbf{T}$ and $\mathbf{x} = [\mathbf{1}, \mathbf{x_0}, \mathbf{x_1}, ..., \mathbf{x_d}]^\mathbf{T}$.

$$y = \phi(\mathbf{w^T x}) \tag{3.1}$$

3.2.3 Activation function

For the case of the original perceptron, the activation function used was a simple step function, when that is the case, this single neuron is called a Threshold Logic Unit (TLU) [55]. However, some other activation functions had been used, and depending on the application some activation functions are preferred over the rest.

The process of applying an activation function to a unit can be divided into two steps. From equation 3.1, each unit within a network calculates first the weighted sum of the inputs (which can be the original inputs, or some outputs from another units), after that, a non-linear function is applied on that result to finally produce an output [57].

The nonlinearity of the activation function is what makes neural networks possible to approximate and learn some real complex functions. The most common activation functions along with their derivatives are shown in Figure 3.2.

Figure 3.2: Common Activation Functions [55]

3.2.4 Multilayer neural network

In particular, a perceptron does not necesarily need to be formed of a single unit, it might be composed of a single layer with several TLU units, where every input is connected to each TLU unit.

There are some problems that can be solved by using a perceptron, but by doing so, just approximate linear functions of the input can be obtained [59]. However, most real life problems are much more difficult to solve, and cannot be simply modeled by a single layer. For this matter, an Artificial Neural Network (ANN) known as the MLP is applied.

A MLP is created by stacking multiple perceptrons, i.e., the architecture is formed by several layers. Here again, a function is learned to map the input values to the output values, and under the hood, this function is composed of many simpler functions [60].

A simple representation of a MLP is shown in Figure 3.3. The layer on the bottom is called the input layer, the layer at the top is called output layer, and any layer in between is called hidden layer.

Figure 3.3: Multilayer Perceptron [59]

3.3 Deep learning

As mentioned before, more complex problems can be addressed by stacking several layers of neurons, here the term DL emerges, which refers in a simple definition, as

a machine learning model using multiple layers of simple adjustable elements [57].

This simple adjustable elements, i.e., neurons, process the information from the input layer to the output. As information propagates deeper and deeper, its representation becomes more and more abstract, thus, gaining the ability of solving really complex problems. Some examples are natural language understanding, sentiment analysis, language translation, image recognition, speech recognition, and many more [61].

DL models learn by using an algorithm called backpropagation. This algorithm, allows the model to learn the parameters or weights between layers in order to optimize an objective function, normally called loss function [62].

3.4 Loss function

The loss fuction, cost function, or error function measures how well the estimate or predicted value matches the true value [63]. The type of loss function to use is problem specific. Here are some of the most common types.

3.4.1 Binary cross-entropy

Used for binary classification problems, the binary cross-entropy is defined as follows [64]:

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}(y_i log(\hat{y}_i) - (1 - y_i)log(1 - \hat{y}_i)) \tag{3.2}$$

where y_i is the true value, $\hat{y}_i$ is the predicted value, and n is the number of training examples.

3.4.2 Categorical cross-entropy

Also called Multi-class Cross entropy and it is used for classification problems with more than two categories. For a single training example this loss function is defined as follows [64]:

$$L(y, \hat{y}) = -\sum_{k}^{K} y^{(k)} log \hat{y}^{(k)} \tag{3.3}$$

where $y^{(k)}$ is the true value (0 or 1), $\hat{y}^{(k)}$ is the predicted value, and K is the total number of classes.

3.4.3 Mean squared error

For a regression type problem, a Mean Squared Error (MSE) loss function is commonly used, and it is defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{3.4}$$

where y_i is the true value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

3.5 Backpropagation

The learning process of a neural network can be reduced to the learning of the best possible weight values of each layer, in a way that the objective function is optimized. For many years researches in the area of machine learning had struggled in finding algorithms that could train architectures with multiple layers [55]. It was not until 1986 that in [65], a new algorithm was proposed to train MLPs. It was called backpropagation.

The backpropagation algorithm computes the gradient of an objective function

with respect to the weights, and it is the same as applying the chain rule used in calculus to get the derivative of a function [61]. As presented by [55], the backpropagation algorithm works as follows:

1. The whole dataset or a subset (mini-batch) is fed to the network.
2. This batch is passed to the first input layer, and the computations of the input layer are passed to the next layer. The process is repeated until, the batch travels all the way through till the output layer. This process is called forward pass (prediction).
3. Once the output is obtained, an objective function or loss function is computed in order to measure the error between the expected and the actual output.
4. Once the error is known, the chain rule is applied in order to find out the individual contribution of each one of the learnable weights in the network. This step is called backward pass, and it continues until the input layer is reached.
5. Finally, once the gradients are obtained, a gradient descent step is applied to each one of the weights in order to update their values and reduce the error. The degree of how much the weights are updated is controlled by a parameter called the learning rate.

The above algorithm is repeated until the error is minimized to an acceptable value, some other methods have also been proposed to know when to stop the learning process [66].

3.6 Sequence models

The main characteristic of the data to be used in this research is that it is sequential. This type of data is normally analyzed using sequence models. A sequence model

is just a model whose inputs have a sequential dependence. The most basic neural network sequence model is the RNN, and it is based on the work presented by David Rumelhart in [65].

3.6.1 Recurrent neural networks

The purpose of the RNN is to model sequences of tensors [67], and it is also considered to be a generalization of Feedforward Neural Networks (FNNs) to sequences [68]. RNNs are very well suited to process sequential data such as text, speech or time-series data [56].

For RNNs, the input is a sequence of data $(x_1, ..., x_T)$, where T represents the length of the sequence, and for each input of this type, a sequence of outputs is computed $(y_1, ..., y_T)$. The output is obtained by applying the following equations at each time step [68]:

$$h_t = sigm(W^{hx}x_t + W^{hh}h_{t-1}) \tag{3.5}$$

$$y_t = W^{yh}h_t \tag{3.6}$$

where h_t is the hidden state at time t, x_t is the input at time t, and h_{t-1} is the hidden state of the previous layer at time $t-1$, or the initial hidden state at time 0. W^{hx}, W^{hh} and W^{yh} represents the weight matrix that multiply the input x_t, the previous hidden state h_{t-1} and h_t respectively.

On the left side of Figure 3.4, a simplified version (folded version) of a RNN is shown. The loop indicates that the computed value at any time step can be fed to the next time step. On the right side, the unfolded version of a RNN is presented, here U, W and V correspond to W^{hx}, W^{hh} and W^{yh} respectively. As it can be seen,

these weight matrices are shared along all time steps.

Figure 3.4: Schematic diagram of a basic RNN [60]

Once that all outputs are computed, a loss function is evaluated, the errors are retro-propagated by using the backpropagation algorithm, that within this context, is called Backpropagation Through Time (BPTT) [55].

The key feature of RNNs is that some information is shared to posterior time steps, this gives RNNs a type of memory that can be used to better understand a sequence [63].

3.6.1.1 Long short-term memory

As already pointed out, RNNs are very good when dealing with sequential data, however there is a little problem, they suffer with long term dependencies in a sequence, i.e., the RNN model fails to establish the relationship between two distant data points or steps in a sequence [69]. This is known as the vanishing gradient problem.

The basic idea of this problem is that for long sequences in a RNN the gradient might get very small, and so, the changes when updating the weight values. In the worst case, this makes stopping the learning way before something is actually

learned.

The problem was confirmed by Hochreiter in [70], on which experiment results showed that conventional learning algorithms for RNNs cannot learn long time lag problems in a reasonable time. On the same study, it was shown that LSTM models performed well in long time lag problems of up to 1000 steps.

The LSTM model was proposed in [32] as a solution to the vanishing gradient problem and to handle long-term dependencies in the sequence. The architecture of a LSTM Cell is shown Figure 3.5.

Figure 3.5: LSTM Cell [69]

The governing equations of the LSTM are given as follows:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \tag{3.7}$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \tag{3.8}$$

$$g_t = tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \tag{3.9}$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \tag{3.10}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{3.11}$$

$$h_t = o_t \odot tanh(c_t) \tag{3.12}$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, h_{t-1} is the hidden state of the previous layer at time $t-1$, or the initial hidden state at time 0. i_t, f_t, g_t, o_t are the input, forget, cell, and output gates respectively. The sigmoid function is represented by σ, and $\odot$ represents the Hadamard product, or element-wise product.

In order to analyze what is inside a LSTM cell, a focus needs to be made on two principal things: the cell state c_t and the hidden state h_t. h_t can be seen as being a short-term state, and c_t as being a long-term state [55].

At time t, the LSTM cell receives three inputs: the previous cell state c_{t-1}, the previous hidden state h_{t-1}, and the input x_t. As shown in Figure 3.5, h_{t-1} and x_t are fed to four layers (gates). From left to right, the purpose for each gate is described as follows [55]:

1. First, the forget gate f_t is computed, h_{t-1} and x_t go through a fully connected layer with a sigmoid activation function, this makes the output to be in the range between 0 and 1. This gate controls what information should be removed from c_{t-1}.

2. The input gate i_t controls which information should be added to c_{t-1}, it has again a sigmoid activation function, which makes this a controlling gate with values between 1 and 0.

3. The cell gate g_t, has a similar function to what a RNN cell would do, it takes h_{t-1} and x_t, and computes an output by applying a $tanh$ activation function.

4. Once i_t and g_t are computed, a point-wise element multiplication is performed between the two, in other words, i_t chooses which information to keep from

g_t.

5. The output gate o_t, controls what information should be used to generate the output in response to the input h_{t-1} and x_t [56].

Finally the cell state c_t is computed by removing and adding some new information. Information is removed by a point-wise element multiplication between c_{t-1} and f_t, and new information is added once it was selected via point-wise element multiplication between i_t and g_t.

In summary, a LSTM model is capable of carrying information by learning long-term dependencies as well as to deal with short-term memory at the same time.

3.6.2 Transformer

In 2017, a group of Google researchers proposed a novel architecture called the *Transformer*, it was presented on a paper called *Attention is all you need* [33]. This new architecture only uses attention mechanisms, totally dispensing of recurrence or convolutions. Another advantage is that Transformers can significantly improve parallelization as well as to reduce the training time.

Before the Transformer, RNNs or CNNs, or a combination of both (with or without attention mechanisms) had been setting state-of-the-art results for a variety of tasks, mainly in NLP and Computer Vision (CV). Thus far, transformers have achieved state-of-the-art results in these two areas of application, take for instance the results from [33, 71, 72].

In Figure 3.6, the transformer model architecture is shown. It follows an encoder-decoder structure, on the left-hand side, the encoder module is shown, decoder is on the right. On the original Transformer, both encoder and decoder modules are a composition of a stack of $N = 6$ layers.

For the encoder, each layer has two sub-layers: a multi-head self-attention mech-

anism, and a position-wise fully connected feed-forward network. For the decoder, each layer has three sub-layers: same layers as the encoder, plus a multi-head attention that computes attention on the encoder stack output.

Between each sub-layer in either the encoder or decoder, residual connections followed by a layer normalization are applied. As shown in Figure 3.6 on the decoder module, the multi-head self-attention is masked in order to prevent putting attention on upcoming positions.

Figure 3.6: The Transformer architecture.

The multi-head self-attention layer contains eight parallel heads, yielding differ-

ent linear projections queries, keys and values used later by the attention function. On the original paper, this attention function is called *"Scaled Dot-Product Attention"*, such function is shown below [33]:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.13}$$

Where Q, K and V are the queries, keys and values respectively. For the input in both the encoder and decoder, learned embeddings are used to map input tokens onto a dense representation of size d_{model}, which for the original Transformer it was 512. Just right after the input is mapped to an embedding space, a positional encoding is added. The positional encoding vector can be computed with equations 3.14 and 3.15.

$$PE_{(pos,2i)} = sin(pos/10000^{\frac{2i}{d_{model}}}) \tag{3.14}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{\frac{2i}{d_{model}}}) \tag{3.15}$$

By adding a positional vector to the embedding, vector information in regards of the ordering of the sequence is given, otherwise the order would be lost since no recurrence is made as in the LSTM model.

For the purpose of this research, no decoder stack is needed, outputs from the encoder stack are taken and post-processed towards a final classification.

3.7 Clustering

One of the most used unsupervised ML approaches is clustering. In this research, clustering is applied directly to the reconstructed LOB and preprocessed data in order to obtain a reduced representation of the LOB state space.

Clustering is a type of ML algorithm which main objective is to divide the data

into groups (clusters), in such a way that members of the same group are similar, and members of different groups are not [58]. Some often used applications of clustering are: data analysis, search engines, customer segmentation, image segmentation. dimensionality reduction [55].

3.7.1 k-means

k-means is perhaps the most popular clustering technique, it was proposed by Stuart Lloyd in 1957, but it was not made public until 1982 in [73]. The main purpose of k-means algorithm is to divide the dataset into k different groups or clusters. The number of clusters k needs to be provided as an input to the algorithm. In case that no information about the number of clusters is known in advance, some methods exist to estimate k [74].

The k-mean algorithm is very simple, it iteratively perform two steps until convergence is reached [60]. The algorithm initializes by assigning k different D-dimensional centroids $\{\mu^{(1)}, ..., \mu^{(k)}\}$ with some inital values, where D is the dimension of the data points.

In the first step of the algorithm, each data point is assigned to a cluster i, where i is taken from the closest centroid $k^{(i)}$ to the data point; euclidean distance is commonly used in this step. Once all data points are assigned to a cluster, the second step follows, here the values of $\mu^{(i)}$ are updated by computing the mean of all data points assigned to cluster i.

The k-means algorithm finishes when there are not further changes in the cluster assignments or some maximum value of iterations has been reached [75]. A graphical representation of the first 3 iterations of the k-means algorithm is shown in Figure 3.7.

Figure 3.7: Evolution of k-means algorithm [59]

One of the applications of k-means is the mapping of a continuous space to a discrete space, this is known as vector quantization [59]. In the context of this research, a quantization step is performed when clustering the LOB state space.

3.8 Natural language processing

NLP comprises a set of techniques dealing with the understanding of language [76]. Nowadays NLP is immerse in our daily lives in different applications such as: web search engines, social networks, text-to-speech services, translations, sentiment analysis, etc [77].

3.8.1 Vector representation

The understanding of language for solving real-world tasks, involves transforming text into a representation that a computer can understand, e.g., numerical vectors. Popular vector representations in NLP are: One-hot encoding and WE.

3.8.1.1 One-Hot encoding

From a NLP perspective, a One-Hot encoding is obtained once the vocabulary has been constructed. Then each word can be represented as a vector of zeroes of equal dimension to the vocabulary size, and just a "1" is added in the position where the word is found within the vocabulary [78]. This is sometimes referred as a sparse representation, and contrary to WEs, the One-Hot encoding lacks of a semantic interpretation.

3.8.1.2 Word embeddings

While working in time series data, and inspired in what is done in NLP, the use of WEs occupies an important part of this research. WEs are considered to be one of the most powerful and widely used techniques to represent words, characters, and some other objects in a sequence [79]. They are also sometimes attributed to be the reason of DL success in NLP [67].

WEs are dense vector representations of words. The word dense comes from the comparison between a One-Hot vector representation of a word, being high dimensional (equal to the vocabulary size) and sparse (with most values being zero), while WEs are real valued lower dimensional vectors [80].

Word embeddings encode each word and represent it in such a way that words close to each other in this new vector space are expected to have semantic similarity [56]. A 2D projection of WEs is shown in Figure 3.8, it can clearly be seen that similar words are close to each other.

Figure 3.8: Word Embeddings (2D projection) [57]

Word2Vec, one of the most popular WE techniques was proposed by a research team at Google [81, 82]. In word2vec, WEs are learned by neural networks, by two different model architectures, CBOW and Skip-gram. The former predicts the current word based on the context, while the latter predicts surrounding words given the current word, both model architectures are shown in Figure 3.9.

Figure 3.9: Word2Vec architectures [81]

WEs are not limited to the mapping of similar words to a close space, but there are some other properties that result fascinating and almost unexplainable. Us-

ing algebraic operations on learned WEs had shown that some relationships are also learned, for instance, the algebraic operation: $vector(King) - vector(Man) + vector(Woman)$ results in a vector that is very close to $vector(Queen)$[81].

A similar relationship between countries and capital vectors is shown below in Figure 3.10 .

Figure 3.10: Implicit relationships learned with WEs [82]

WEs are not only useful for English, but for any sequence of symbols where the appearance in the sequence has some representative meaning [83].

In this work, a WE encoding of each LOB state is learned along with the mid-price movement prediction task, i.e., the numerical values of the WE vectors are learned during the training phase (more details about how WEs are learned are given in Chapter 4: Methodology).

3.9 Evaluation metrics

One key thing to keep in mind whenever a model is created is to find a way to measure its performance. Performance metrics give a diagnostic onto how the model is doing, and also, it allows to compare the model with some other models in a

standardized way.

The final goal of a model is to generalize well on unseen data, i.e., data where the model has not been trained on [67]. This is the reason why performance metrics should be taken from unseen datasets.

As some other things in ML, evaluation metrics are task specific. In this work, given the nature of the task (classification), evaluation metrics such as accuracy, precision, recall and F1-score are used.

3.9.1 Accuracy

Accuracy can be simply understood as the proportion of correctly predicted observations [58]. For a binary classifier (e.g., positive or negative), accuracy is calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.16}$$

where, TP and TN, are the number of true positives and true negatives respectively, and FP and FN, are the number of false positives, and false negatives respectively.

However, when dealing with an imbalanced dataset, using accuracy as evaluation metric is in general inadequate. For example, for a dataset of 1000 records, with 950 being of class A, and the remainder of class B, it is not fair to measure the model's performance by simply computing the accuracy. Under this scenario, a model that always predicts the majority class (class A) would have very high accuracy but would lack of predictive power [63]. In such cases, it is advised to use: precision, recall and the F1-score, or some other metric that take into account the imbalance.

The following metrics are set to work on a binary classifier, however their use can also be extended to a multi-class classifier by treating the data as a set of binary classifiers.

3.9.2 Precision

Precision is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{3.17}$$

Which put into words would be something like: for those examples that were predicted to be positive, what is the proportion that actually are.

3.9.3 Recall

Recall is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{3.18}$$

Which put into words would be something like: for those examples that are actually positive, what is the proportion that the model identified as such.

3.9.4 F1-Score

The F1-Score, also called F-Score or F-measure is the harmonic mean of precision and recall [58]. The value of this metric will only be high if both recall and precision are high [55]. It is defined as:

$$F_1 score = 2\frac{(Precision)(Recall)}{Precision + Recall} \tag{3.19}$$

3.9.5 From binary to multi-class

As mentioned before, the above metrics are used to evaluate binary classifiers, however, their use can also be extended to a multi-class setting. In order to do so, the problem is treated as a collection of C binary classifiers, where C is the number of different classes.

Each binary classifier is formed by assigning the positive class to be one of the

C classes, while the rest are assigned to be the negative class. Per-class metrics (e.g. precision, recall or F1-Score) are then averaged by using one of the following approaches [63, 58, 55]:

- **macro**: the arithmetic mean of the binary metrics is computed. It gives equal weight to each class.
- **weighted**: the weighted arithmetic mean of the binary metrics is computed. The weight given for each class is proportional to the appearance of the class in the data, i.e., its support.
- **micro**: it computes the total number of TP, TN and FP across classes. Then, precision, recall and F1-Score can be determined from these quantities. It tends to ignore the majority class if present.

For the the multi-class problem at hand, a macro average approach is taken in order to equally weight each movement in the market: Up, Down or Stationary. More details about the model evaluation and metrics used are given in Chapter 4: Methodology.

3.10 Statistical significance tests

Statistical significance tests are used to estimate the probability that a particular observed relationship in the data/results can be merely attributed to pure luck. In order to do so, hypothesis testing analysis is applied. Two mutually exclusive hypothesis are presented at the beginning of the analysis, namely the null hypothesis H_0 and the alternative hypothesis commonly known as H_1 or H_A [84].

An α value, also known as the significance level is given before applying the statistical test. This α value quantifies the probability of rejecting the null hypothesis

given that it is actually true, i.e., a type I error [59]. The smaller the α value the less room for errors of this type.

The α value is then get compared against a p-value computed from the statistical test itself. The p-value is defined as the probability of obtaining the observed effects in the data given that the null hypothesis is true. In other words, the smaller the p-value the stronger the evidence against accepting the null hypothesis.

In the case that the computed p-value is less than or equal to the α value, then, the null hypothesis is rejected and the results are considered to be statistically significant. Otherwise the null hypothesis is accepted and results are not statistically significant.

In this work, a statistical significance test is to be performed to analyze the different performances for several classifiers and baseline models. There exist a variety of tests that could be utilized, however, for reasons described in Chapter 7: Results, the McNemar's test was considered to be the most suitable statistical test to be applied.

3.10.1 McNemar's test

In the context of this work, the McNemar's test is used to compare the classification performance of two given models. Models are trained on the training set and evaluated on the test set. Predictions of both models are compared against the ground truth and a contingency table is populated as shown in Figure 3.11.

e_{00}: number of examples misclassified by both	e_{01}: number of examples misclassified by 1 but not 2
e_{10}: number of examples misclassified by 2 but not 1	e_{11}: number of examples correctly classified by both

Figure 3.11: Contingency table summarizing errors count for both models [59].

For the McNemar's test the null hypothesis is that both models have the same

error rate, i.e., $e_{01} = e_{10}$. With a large enough number of misclassified examples ($e_{01} + e_{10}$), the equation 3.20 [85] is equivalent to a chi-squared distribution with one degree of freedom.

$$\frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}} \sim \chi_1^2 \tag{3.20}$$

Same rules for acceptance or rejection of the null hypothesis apply as explained previously.

3.11 High frequency trading

HFT is a type of algorithmic trading whose trading strategies are characterized for making automatic transactions at very low latency, and investments with very short time horizons. This type of trading has been significantly increased during the last 10 years, with an estimate of about 55% of the total stock market volume in the United States, and 40% on European markets [86].

Although the term HFT is not clearly defined, the Securities and Exchange Commision (SEC), through the Concept Release, identified 5 main characteristics that the term HFT is frequently related to: 1) the use of high speed programs that generate and execute orders, 2) the use of co-location to reduce latency and network delays, 3) very short time investment horizons, 4) a big volume of sent and canceled orders, and also, 5) ending the trading day without opened or unhedged positions [87].

According to [10], HFT strategies are divided in 4 main classes: arbitrage, directional event-based trading, automated market making and liquidity dectection. Even though, this research is centered in a type of data that is usually consumed by HFT traders to generate trading strategies, there is not aiming at generating a trading strategy *per se*, however, research's findings might be helpful for those who do practice it.

3.12 Market microstructure

Most of the economics literature does not consider the microstructure behavior of the markets, in the words of Maureen O'Hara [88], the market microstructure is defined as "the study of the process and outcomes of exchanging assets under explicit trading rules".

This research is focused in market microstructure data such as the LOB and the order flow. At this granularity level, fundamental data is not of great use.

3.12.1 Limit order book

A large part of the dataset used in this work is formed by data extracted from the first 10 price levels of each side of the LOB, containing prices and volumes at each price level. This makes the LOB a core component of this research.

A LOB is a register of all outstanding limit orders pending to be completed in a LOB driven market. The orders that are added to this register are called limit orders, which are set to be filled at a specific price or better [14].

A limit order lets you set the amount or size you want, and also the price you are willing to buy or sell an asset (Bitcoin in this context). On the other hand, a market order is a type of order sent to the market with the intention to be executed immediately, it can either be a buy or sell order, and it is filled at any cost, i.e., it will be filled at the best current price [89]. Just limit orders are added to the LOB if there is not an immediate match before.

As seen in Figure 3.12 (and as it is normally represented), the LOB is divided in two major components the bid side and the ask side, which correspond to the opened buy limit orders and opened sell limit orders respectively.

Both sides are organized by price levels (horizontal axis in Figure 3.12) as follows, for the bid side, the first price level corresponds to those buy limit orders

Figure 3.12: Schematic representation of a LOB [90].

whose price is the highest (best bid), and for the ask side, the first level corresponds to those sell limit orders whose price is the lowest (best ask).

Orders can be sent to the market at discrete prices, and the lowest permissible price difference between two orders is called *tick size* [90], which is a value set by the market on which the trades are taking place. All limit orders must then arrive at a price that is a multiple of the tick size. Analogous to the tick size for the price is the *lot size*, which is the smallest amount or size an order can have, all orders sizes must then be a multiple of the lot size.

According to Alridge in [10], most LOBs have in common the following set of rules:

1. A limit order remains in the LOB until matched or canceled.

2. Prices in the bid side are organized in a descending way, starting with the first level being the highest price (best bid), and continuing with lower price levels.

3. Prices in the ask side are organized in a ascending way, starting with the first level being the lowest price (best ask), and continuing with higher price levels.

4. A price level can have more than one limit order, i.e., all limit orders with the same price are aggregated on the same price level.

5. When a limit order is sent to the market, canceled or matched, the corresponding price level is created, removed or updated accordingly.

6. When a market order arrives to the market, it is immediately filled and will never be opened in the LOB.

7. If a market order arrives, then, the incoming order would be filled by consuming the remaining opened limit orders until it is completely filled.

8. If a limit order arrives, it will be opened in the LOB if not immediate match is found, otherwise, it will be partially or completely filled. If partially filled, the remaining size will be opened in the LOB.

9. All limit orders are governed by the First-In, First-Out (FIFO) policy, this applies for cases where multiple orders are at the same price level.

Chapter 4

Methodology

A detailed description of the followed methodology is presented in this chapter. It covers all the steps taken at each research stage, as well as a brief explanation of why some methods are preferred over others for the problem at hand. It should also provide enough information to the reader as to allow reproducibility if desired.

A general overview of the methodology applied is shown in Figure 4.1, details at each step are provided in the following sections.

4.1 Data collection

This thesis' task is mainly focused on very short-term predictions and HFT. The most popular source of information for such a task is found on high-frequency datasets in the form of LOB and messages of events.

However, public high-frequency datasets are very scarce, some examples are referenced in [16], but most of them are uniformly sampled, i.e., at a specific sampling rate, but by doing so, some information could be missing, and that could be important for the general model causality. It has been shown that subsampling time series data can cause problems for discovering the system's causal structure [91].

Figure 4.1: General overview of the methodology

For this reason, and to ensure that the input data on this research was as complete as possible, a new dataset was created. In general, the more granular the market information is, the more expensive it gets. Considering this, and the fact that most of the top cryptocurrency exchanges provide an API for information gathering at no extra cost, high-frequency data was acquired from the Coinbase exchange.

However, for quality purposes and the technological requirements that need to be fulfilled in order to get reliable data, the data was obtained from a third party, Kaiko, a popular cryptocurrency market data provider. Descrition of the raw dataset is provided below.

4.1.1 The raw dataset

The raw dataset consists of a stream of messages sent to the market for the pair BTC/USD from August 25th to October 25th, 2020 (62 days in total). The data was extracted from the cryptocurrency exchange Coinbase by the Kaiko team. They collected details on every order sent, changed, canceled or filled as it happened in

real time. This type of data is the most granular data in the industry.

Two separate files were provided by Kaiko per day: events and snapshots. A single day of data is comprised of one events file and one snapshots file. A sample from the events file is shown in Figure 4.2, where every row can contain one of the event types shown in Figure 4.3.

159831381	side:"buy"	product_i	time:"202	sequence	order_id:"	order_typ	size:"0.12	price:"117	client_oid:"5dc88504-d115-483c-88a4-51642631459c"}
159831381	side:"buy"	product_i	time:"202	sequence	price:"117	order_id:"	remaining_size:"0.12415067"}		
159831381	side:"sell"	product_i	time:"202	sequence	order_id:"	reason:"ca	price:"117	remaining_size:"1.273"}	
159831381	side:"sell"	product_i	time:"202	sequence	order_id:"	order_typ	size:"1.27	price:"117	client_oid:"6aa5a4b0-d168-42a6-9c7e-3a4cf3ff919e"}
159831381	side:"sell"	product_i	time:"202	sequence	price:"117	order_id:"	remaining_size:"1.272"}		
159831381	side:"buy"	product_i	time:"202	sequence	order_id:"	order_typ	size:"0.5"	price:"117	client_oid:"16d8519b-bbf5-446c-b802-dbb7184b12f4"}
159831381	side:"buy"	product_i	time:"202	sequence	price:"117	order_id:"	remaining_size:"0.5"}		
159831381	side:"sell"	product_i	time:"202	sequence	order_id:"	reason:"ca	price:"117	remaining_size:"0.47584933"}	
159831381	side:"buy"	product_i	time:"202	sequence	order_id:"	reason:"ca	price:"117	remaining_size:"0.02808021"}	
159831381	side:"buy"	product_i	time:"202	sequence	order_id:"	reason:"ca	price:"117	remaining_size:"0.01730478"}	
159831381	side:"buy"	product_i	time:"202	sequence	order_id:"	reason:"ca	price:"117	remaining_size:"0.01730523"}	
159831381	side:"buy"	product_i	time:"202	sequence	order_id:"	reason:"ca	price:"117	remaining_size:"0.0238309"}	

Figure 4.2: Events file provided by Kaiko


```
{
  "type": "received",
  "time": "2014-11-07T08:19:27.028459Z",
  "product_id": "BTC-USD",
  "sequence": 10,
  "order_id": "d50ec984-77a8-460a-b958-66f114b0de9b",
  "size": "1.34",
  "price": "502.1",
  "side": "buy",
  "order_type": "limit"
}
```

```
{
  "type": "received",
  "time": "2014-11-09T08:19:27.028459Z",
  "product_id": "BTC-USD",
  "sequence": 12,
  "order_id": "dddec984-77a8-460a-b958-66f114b0de9b",
  "funds": "3000.234",
  "side": "buy",
  "order_type": "market"
}
```

```
{
  "type": "open",
  "time": "2014-11-07T08:19:27.028459Z",
  "product_id": "BTC-USD",
  "sequence": 10,
  "order_id": "d50ec984-77a8-460a-b958-66f114b0de9b",
  "price": "200.2",
  "remaining_size": "1.00",
  "side": "sell"
}
```

```
{
  "type": "done",
  "time": "2014-11-07T08:19:27.028459Z",
  "product_id": "BTC-USD",
  "sequence": 10,
  "price": "200.2",
  "order_id": "d50ec984-77a8-460a-b958-66f114b0de9b",
  "reason": "filled", // or "canceled"
  "side": "sell",
  "remaining_size": "0"
}
```

```
{
  "type": "match",
  "trade_id": 10,
  "sequence": 50,
  "maker_order_id": "ac928c66-ca53-498f-9c13-a110027a60e8",
  "taker_order_id": "132fb6ae-456b-4654-b4e0-d681ac05cea1",
  "time": "2014-11-07T08:19:27.028459Z",
  "product_id": "BTC-USD",
  "size": "5.23512",
  "price": "400.23",
  "side": "sell"
}
```

```
{
  "type": "change",
  "time": "2014-11-07T08:19:27.028459Z",
  "sequence": 80,
  "order_id": "ac928c66-ca53-498f-9c13-a110027a60e8",
  "product_id": "BTC-USD",
  "new_size": "5.23512",
  "old_size": "12.234412",
  "price": "400.23",
  "side": "sell"
}
```


Figure 4.3: Event types, source: https://docs.cloud.coinbase.com

The events file contains all the events or actions that were sent to the market and recorded during the whole day. While the snapshots file contains 24 snapshots of the full LOB at a particular event number (24 snapshots are provided as to match the 24 hours of a day). Both files are comma-separated values (CSV), and by using both, it is possible to reconstruct the LOB at any point in time as will be explained next.

4.1.2 Limit order book reconstruction

The LOB state at some time t is believed to be extremely important for practitioners, it allows to gauge the trading activity and the underlying economic motivation [92]. By reconstructing LOB historical states, participants can better assess market demand and support, look for arbitrage opportunities, price impact analysis, train machine learning algorithms, and more.

In order to rebuild the historical LOB states, two things are needed: a past LOB state, and the stream of updates from that point and forward, namely, the snapshots file and events file. A simplified version of the LOB states reconstruction is shown below in a pseudocode form:

Algorithm 1 Historical LOB Reconstruction

```
procedure RECONSTRUCTION(Events, Snapshot)
    LOB ← Snapshot                                  ▷ Initial LOB state
    S ← Snapshot sequence number
    for event in Events do
        N ← event's sequence number
        if N = S + 1 then
            LOB ← update(LOB, event)                ▷ Update LOB state as per event type
            S ← S + 1
        end if
    end for
end procedure
```

The above procedure was used to reconstruct the LOB states for the entire raw

dataset. Due to some missing events on the data, the reconstruction was done hour by hour, i.e., from every snapshot event number till the next snapshot event number. Thus, for a single day, 24 files of reconstructed LOB were created.

At the same time, this allowed to verify that the reconstruction was correct by verifying that the last LOB reconstructed state for a single file was the same as the next snapshot of the LOB in the snapshots file.

During the reconstruction, just the first 10 price levels (on both sides: sells and buys) were kept, following what was done in previous works [16, 93], and the findings in [23], that suggested that the most informative features are within the first 10 levels as it was empirically proved. Some other considerations were made, details are explained next:

- All message events having the same timestamp, i.e., that happened at exactly the same time, were recorded as a single event by aggregating the individual contributions per message. This happens when multiple orders are sent at the same time by the same participant, or because the matching engine protocol split one order into multiple artificial orders till the incoming order gets filled or partially filled [94]. Not following this consideration would end up in a reconstruction of some LOB states not part of the natural order sequence flow, i.e., artificial LOB states.

- Also, whenever an event from the LOB states sequence was missing, the reconstruction stopped, and continued from the next available snapshot onwards.

- All messages that ended up updating the LOB state were considered regardless if the update happened at any of the first 10 levels or not.

The reconstruction phase was a lengthy process since it had to loop over every message. In order to speed up the process, the 24 files per day were reconstructed

at the same time, considerably reducing the amount spent per day of data. The total reconstruction time was about 150 hours. More than 1.1 billion events (messages) were processed during that time.

Apart from recording volume and size at each price level, and in order to extract as much information as possible during the reconstruction, information related to the order type and time elapsed between events was also recorded. A detailed description of the final features contained per sample is shown in Table 4.1.

Feature name	Number of Features	Feature type
Price per price level (asks)	10	Numerical
Price per price level (bids)	10	Numerical
Volume per price level (asks)	10	Numerical
Volume per price level (bids)	10	Numerical
Event buy side	1	Boolean
Event sell side	1	Boolean
Event both sides	1	Boolean
Order type open	1	Boolean
Order type done-canceled	1	Boolean
Order type done-market filled	1	Boolean
Order type done-limit filled	1	Boolean
Order type change	1	Boolean
Elapsed time since last event	1	Numerical

Table 4.1: Features list per sample

4.2 Data preprocessing

After having reconstructed the LOB states, the data was still not suitable to be fed to a ML/DL model. Three different normalization/representations were applied to the dataset: z-score, stationary features, and a tokenized version of the data. Stationary and tokenized are proposed in this research as a novel method to represent the LOB sequential data from a different perspective.

For the z-score normalization, mean and standard deviation are computed per feature. Details on how the stationary and tokenized versions were created are

explained in Chapter 5: The proposed limit order book sequence representation.

At the end, three different preprocessed reconstructed datasets were available. No further preprocessing is made to these datasets besides the sequence segmentation that will be explained next.

4.3 Sequence segmentation

Having represented the reconstructed data with the three different methods mentioned above, the data was segmented in order to create the examples that were used for training. This allowed to phrased the LOB time series data as a supervised learning problem, particularly a classification problem.

The segmentation of the data was made by using a sliding window with a look back of 100 time steps (in this context, time steps are referred to LOB states or events), i.e., just the 100 more recent LOB states are used for predicting the mid-price movement at a future time. Same sequence length is used in [20, 21, 17, 18, 30, 50, 22, 23].

Comparisons against longer sequences are presented in [14], although slightly higher accuracy values are achieved when using a longer sequence, yet, there are some trade-offs to be made in terms of complexity and training time. For better comparison to existing literature results, the same sequence length was selected.

Stride was set to the length of the sequence, i.e., 100. There is no overlapping between sequences. However, later data augmentation was applied on these datasets by setting the stride to 20, i.e., having 5 times more sequence data than before. For this case, data augmentation was made just on the training set. Validation and test sets were generated with no overlapping.

A total of 6,429,227 sequences were created for each dataset, where each sequence is a vector of 40 dimensions, which originally (before preprocessing) cor-

responded to the volume and prices of the first 10 price levels in the LOB (bids and asks).

4.4 Task description and data labeling

This research focuses on predicting the price direction at a future time step (i.e. a horizon measured in number of events), given some past or historical data (sequence). Here the metric to predict is the direction of the mid-price $p(t)$, which is a virtual price calculated from the best bid $b(t)$ and best ask $a(t)$ at time t, and it is given by [90]:

$$p(t) = \frac{a(t) + b(t)}{2} \tag{4.1}$$

For the price direction forecast, some works have approached this problem by defining a two-class classification problem (Up or Down), however, this approach is better suited for longer horizons: days, months, hours [11]. On the other hand, for the mid-price movement prediction task a *de facto* standard is to define the task as a three-class classification problem, where any sample instance can take on one of three classes: Up, Down or Stationary. The stationary class can be seen as to better separate the Up class from the Down class in an effort to reduce signal noise from the noisy LOB sequences.

The labeling method proposed by Tsantekidis et. al. in [20] to classify the mid-price movements was adopted in this work. By following their method, very small and noisy short-term changes are smoothed out. The method uses the average of the previous k mid-prices $m_b(t)$, and the average of the next k mid-prices $m_a(t)$ as follows:

$$m_b(t) = \frac{1}{k} \sum_{i=0}^{k} p_{t-i} \tag{4.2}$$

$$m_a(t) = \frac{1}{k}\sum_{i=1}^{k} p_{t+i} \tag{4.3}$$

Where k is the prediction horizon measured in number of events. Then, for every sequence in the dataset, a label l_t was given by comparing the percentage change that resulted from $m_b(t)$ and $m_a(t)$. The cases below cover the three different class labels assigned.

$$l_t = \begin{cases} 1, & \text{if } m_a(t) > m_b(t) \text{ . (1 + } \alpha\text{)} \\ -1, & \text{if } m_a(t) < m_b(t) \text{ . (1 - } \alpha\text{)} \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

Where 1, -1 and 0 correspond to the Up, Down and Stationary classes respectively. And α is a threshold defined as to measure the minimum percentage change for a mid-price movement to be considered an upward or downward movement. As a result, whenever the magnitude of the mid-price movement is less than or equal to α, the sequence was classified as Stationary.

Considerations were made as to not select α to be smaller than the percentage change resulting from a change in price equals to the tick size (for the pair BTC/USD in Coinbase is defined to be $0.01).

In order to assess the performance of different horizons, and to follow commonly used horizons in previous works [22, 15, 26, 19, 30, 28, 18, 16, 21, 20], the following prediction horizons k were explored, $k = \{10, 20, 50, 100\}$. Then, a total of 4 labels per example (sequence) were generated.

4.5 Model architecture selection

Two different DL models were selected to be applied on this research: LSTM and Transformer. The selection was made by considering that both models are naturally sequence oriented, as well as both have proven to be not only suitable but efficient in solving complex problems.

The LSTM is the preferred model to be used on time series data [12]. Besides financial time series analysis, LSTMs can handle tasks like language modeling, language translation, speech recognition, sentiment analysis, etc.

On the other hand, the Transformer is relatively new, does not use any recurrence or convolution, however, its attention mechanism has been setting new state-of-the-art results in many applications previously dominated by LSTM models [33]. For these reasons, LSTM/Transformer-based architectures were proposed and their diagrams are shown in Figure 4.4. Details for each architecture are given next.

Figure 4.4: a) Transformer-based Architecture, b) LSTM-based Architecture.

4.5.1 LSTM-based architecture

The LSTM-based architecture is shown in Figure 4.4 (b). Two different LSTM-based configurations were used; one that had an embedding layer and one that did not. The embedding layer was only used whenever the input sequence was fed in the form of tokens, otherwise, the input sequence was directly fed to the LSTM layers.

The embedding layer acts as a lookup table that maps tokens (in a very sparse space) to a n-dimensional space (dense vectors). It compresses the input feature space into a smaller one. There are some other methods to generate embeddings [82, 95, 96], however by setting the first hidden layer to be an embedding layer, the embedding weights are being learned during training and are task specific.

In this work, tokens represented the clusters assigned after following the proposed method in Chapter 5. Several embedding dimensions were tested and it was determined that having an embedding size of 8, delivered the best results. Same embedding size was used along all different models/architectures.

Next, the output from the embedding layer (or the input from a no tokenized input sequence) is passed to the first LSTM layer. At every time step, a hidden state h_t is computed and fed back to the next time step. Different configurations for the number of stacked LSTM layers and hidden size for the hidden state were evaluated. Best results were achieved when using a stack of 2 LSTM layers and a hidden size of 64. Here, the last hidden state from the first LSTM layer is fed to the second LSTM layer as input.

Then, last hidden state from the second LSTM layer is passed to a dropout layer as an effort to reduce overfitting by randomly zero out some of the outputs from the LSTM with probability of $p = 0.5$, i.e., just 50% of the output is kept at each forward pass during training. At inference time, the dropout does not longer have this behavior and the whole output from the LSTM is used.

A fully connected layer is then used to linearly transform the input, from an

input size equals to the hidden size, i.e., 8 to 3, which corresponds to the number of classes each example can be classified. Then, this output is fed to a Log Softmax function in order to output the probabilities for each class (Up, Down or Stationary).

4.5.2 Transformer-based architecture

The transformer-based architecture is shown in Figure 4.2 (a). For the embedding layer same rules explained on the LSTM-based architecture apply. Then, since the transformer model does not uses any recurrence or convolution [33], information about the position is injected via positional encodings. In order to create the positional encodings for the sequence, sine and cosine functions of different frequencies were used.

The dimension of the positional encoding is the same size as the embeddings (or the number of input features from a no tokenized input sequence), this allows that both can be summed.

The original transformer architecture uses an encoder and decoder for machine translation tasks, however, for solving the problem at hand (classification), no decoder is needed.

The encoder block shown in the transformer-based architecture is form of a stack of 2 identical encoder layers. Every encoder layer has 4 attention heads in the multi-head attention mechanism, and a fully connected position-wise feed-forward network of size 32. Best number of layers, heads and size of fully connected feed-forward layer were determined after a performing a grid search over these parameters.

Outputs from the multi-head attention sub-layer and the feed-forward network sub-layer are both followed by a layer normalization (just right after having added residual connections to these outputs).

The output from the encoder block is passed through a pooling layer as to sum-

marize the contextual vectors along the sequence dimension. Fully connected layer and Log Softmax function have the same effect as explained in the LSTM-based architecture.

4.6 Model evaluation

For evaluating the prediction performance, four metrics were used for each of the resulting models: macro precision, macro recall, macro F1-score and accuracy. Even though four metrics were computed for each model, just the F1-score was used for selecting the best models.

As will be shown later, the datasets generated had imbalanced classes, where for almost all prediction horizons ($k = \{10, 20, 50, 100\}$) the stationary class was over-represented. Details on class distribution for the different prediction horizons, and how the problem of class imbalance was approached is explained later on Chapter 6: Experimental setup.

Under this scenario, accuracy is not the best metric since it can not distinguishes the correct predictions for the different classes [97]. A biased classifier towards the stationary class would get most of the labels to be correct, however, it will have no predictive value for the rest of the classes.

F1-score, precision and recall macro versions were chosen instead of the weighted versions as to allow very low results for any of the classes to be represented on the summarized result metric. Performance metrics were reported after being evaluated on a hold-out dataset, i.e., not previously seen by the model on any stage.

Chapter 5

The proposed limit order book sequence representation

In this chapter, the core of the proposed methodology is presented. This proposed methodology is centered on the feature engineering made as to transform the sequence into a hopefully better representation of the data. Two different approaches were taken for the LOB sequence representation, the first one is referred as the stationary version, while the second is referred as the tokenized version. Details for both are given in the following sections.

5.1 Proposed limit order book: stationary version

The stationary version representation of the LOB sequence, is just a middle step towards the tokenized version. A general overview of the sub-processes applied to the reconstructed LOB is shown in Figure 5.1. Details for each sub-process will be given next.

Figure 5.1: Sequence representation processes pipeline

After having reconstructed the LOB, some preprocessing steps were applied for the different representations. Preparing the time series data to make the problem easier is always desirable, for instance, making the time series stationary [98]. This is one of the main concerns about applying ML/DL techniques on time series data. For this reason, the initial focus was to construct stationary features out of the raw reconstruted LOB features.

The order book shape is used as an indicative for future price movements and market liquidity [31]. For this reason, features regarding price and volume were used to characterize the LOB state at any point in time.

Analogous to what a unit vector is commonly used for (specify the direction of a vector), the order book shape for a particular point in time can be described with an scaled version relative to itself, i.e., here the shape of the LOB is more important than their magnitude as long as their relative sizes are maintained within the features.

For the price features, the relative prices of all price levels to the best bid $b(t)$, and best ask $a(t)$ were considered as it is presented in [90]. For a given price p the

bid-relative and ask-relative prices are as follows:

$$\delta^b(p) := b(t) - p \tag{5.1}$$

$$\delta^a(p) := p - a(t) \tag{5.2}$$

The above is called the relative depth profile, however this does not contain information about the bid-ask spread commonly used in quantitative analysis, and very often used as input for ML models. In order to address this problem, the following modifications to the above equations were proposed:

$$\delta^b(p) := \frac{a(t) - p}{a(t)} \tag{5.3}$$

$$\delta^a(p) := \frac{p - b(t)}{b(t)} \tag{5.4}$$

These features are still relative, but the main difference is that now, bids are relative to the best ask $a(t)$, and asks are relative to the best bid $b(t)$. This, would give as a result the bid-ask spread value on the first price levels, while maintaining the relative ratio between each price. Also, $\delta^b(p)$ and $\delta^a(p)$ results are divided by $a(t)$ and $b(t)$ respectively. The resulting features represent percentage changes and are stationary, i.e., features are scale (price) invariant.

It should be noted that by applying this preprocessing, information related to the raw price is lost during the process, and thus, any pattern related to the price magnitude can not be spotted by a learning algorithm. However, from a high-frequency perspective and its microstructure-based approach, the attention is centered around market dynamics rather than raw asset prices themselves.

Then, under this scenario, such market dynamics are assumed to be more or less

stable across a range of different price scales.

For the volume features, the relative volumes and symmetry between both sides of the book, i.e., sell and buy where used. Then for each LOB state, at every price level, the volume is defined as follows [31]:

$$s_k^b = \frac{v_k^b}{\bar{v}} \tag{5.5}$$

$$s_k^a = \frac{v_k^a}{\bar{v}} \tag{5.6}$$

For $k = 1, 2, ..., H$, where k is the number of price levels on the reconstructed LOB (in this case ten price levels, so H=10), v_k^b and v_k^a are the volumes at the kth level on the bids and asks sides respectively, and $\bar{v}$ is given by:

$$\bar{v} = \sum_{i=1}^{H} (v_k^a + v_k^b) \tag{5.7}$$

By doing this scaling on the volume features, relative volume at each price level as well as the imbalance (degree of symmetry) between both sides of the book is represented. The order book imbalance itself, is considered a key driver for future price dynamics [14].

After having computed the stationary features, it was noted that the individual distributions on the features were right-skewed. For this reason, a log-transformation for each feature was applied on the whole dataset, as a result, less skewed and more symmetrical features were obtained.

A feature with normal-like distribution can be more informative when applying the z-score normalization, besides, the k-means (which was applied later) algorithm makes the assumption of a Gaussian data distribution [99].

Before applying the z-score normalization, consecutive duplicate values were deleted. The rationale behind is to avoid having a high number of appearances of a particular LOB state when there was actually no change at all over consecutive

equal states for the first 10 levels of the LOB. From 808,913,682 time steps contained in the reconstructed LOB, just 391,028,298 where left after removing consecutive duplicate values.

Not removing these duplicates would lead to an overrepresentation of some LOB states. Hence, making it difficult to obtain a fair transition probability density distribution of LOB states, which is in some way used while computing the statistics for normalizing the data.

Finally, a feature wise z-score normalization was applied [63]. z-score was applied by computing the mean and standard deviation from the training set. Validation and test sets were assumed to be drawn from the same distribution.

5.2 Proposed limit order book: tokenized version

Up to this point, the reconstructed LOB was now in its stationary version. Three more sub-processes were performed in order to create the tokenized LOB version.

At this stage, the aim is to apply the k-means algorithm to find a clustering that is able to represent the LOB states in a simplified and compressed version [100]. It is well known that the state space of a LOB is huge [90], so finding a way of simplifying it, and at the same time keeping the important information is always desirable.

One of the disadvantages of the k-means algorithm is that the number of clusters k needs to be provided in advance as input [59]. For the clustering at hand, no optimal number of clusters was known *a priori*.

Several techniques can be applied to estimate the number of clusters [74], the elbow method and the silhouette score were tried. Nonetheless, results showed that none of these techniques was appropriate for this specific task. For the elbow method, the more the value of k was increased the more the inertia decreased, i.e.,

no elbow point was reached. Same occurred for the silhouette score, no maximum score value was obtained as it continuously behaved as a decreasing function.

The above insights showed the continuous nature of the LOB state space, for which a better approach was to define the problem as vector quantization task, where a continuous space is mapped to a discrete space [59].

An analogous problem to this, is the image quantization problem, that consists of mapping the color space, in RGB 256^3 different colors, to a reduced representation, tipically within 8 to 256 colors [101]. This reduces the size of an image, and its contents or main objects are still discernable.

For images, the k value can be more or less estimated according to the desired quality of the image after clustering, this can be easily done by just looking at the final result. But, for time series data, this is something that can not be assessed straightforward. Then, in order to quantify the clustering quality in terms of information loss, a quality measure was proposed. The author referred to it as the Sequence Contrast Score (SCS) and it is defined next.

5.2.1 Proposed sequence contrast score

Given a sequence of LOB State vectors $L(t)$ (where $L(t)$ vectors are taken from the resulting vectors after the z-score normalization described at the end of section 5.1), with dimensionality d, and a sequence length T, the SCS can be computed as:

$$SCS = \frac{\sum_{t=1}^{T}(\sum_{i=1}^{d}(x_{i,t} - x_{i,t-1})^2)(I_A(w))}{\sum_{t=1}^{T}\sum_{i=1}^{d}(x_{i,t} - x_{i,t-1})^2} \tag{5.8}$$

Where $x_{i,t}$ represents the *ith*-feature in $L(t)$ for $t \in \{0, ..., T\}$, $w = \sum_{i=1}^{d}(c_{i,t} - c_{i,t-1})^2$, and $c_{i,t}$ represents the *ith*-dimension at time t of a centroid c to which a specific $L(t)$ is assigned after clustering. $I_A(w)$ is the indicator function [102] given by:

$$I_A(w) = \begin{cases} 1, & \text{if } w \in A. \\ 0, & \text{if } w \notin A. \end{cases} \tag{5.9}$$

Where $A = \{w | w \neq 0\}$. This score measures the clustering capacity of keeping the contrast from the original sequence on the clusterized version. In other words, if all of the pairs of contiguous LOB states that were different in the original sequence, are assigned to a different cluster c in the clusterized version, then SCS will be 1 (which is the maximum score). The trivial case would be to select the number of clusters k to be the same as the number of different examples N in the dataset, however, this is not practical. In practice, a $SCS = 1$ can be achieved for some $k \leq N$, i.e., the score might get saturated to 1 way before $k = N$.

The intuition behind this measure, is that sequential bigger changes in the LOB states are more penalized for not being assigned to different clusters as compared to those where sequential changes are minimal. This is analogous to focus more on the contour of a figure in an picture, than to its inner pixels, which might be not so different.

SCS is a function of k, and although in some cases the highest score can not be achieved because of practicality, in theory, by computing the SCS a better estimate of what k would need to be (as to not lose the entire signal during the process) is obtained. Different k values were tried along with their respective SCS. Results are shown in Figure 5.2.

Figure 5.2: SCS for some k values.

From the plots above, it can be seen that the score flattens towards 1 as the number of cluster increases, however, SCS is a bounded exponential function, in which the increment rate decreases drastically as k increases. For this reason, no attempt was made to get the maximum score ($SCS = 1$). That would imply having a very large value for k, and will not simplify the problem. Instead, three different k values were empirically selected, $k = \{3000, 30000, 300000\}$, with SCS scores of 0.6626, 0.8715 and 0.9072 respectively. These values were selected as to test the trade offs between having a higher SCS by increasing the number of clusters, and the model performance and overall complexity involved.

5.2.2 k-means clustering

The instances for the k-means algorithm are the LOB states (in their stationary version) that resulted from the preprocessing steps presented in 5.1.

Before feeding these stationary instances to the k-means algorithm, and in order to reduce the number of data points to be clustered, the data was sampled down. A total of 40,000,000 (≈10% of the data) points were randomly sampled without replacement. Similar clustering quality is achieved by using the whole dataset, however by using the subsampled version, its training time is reduced drastically since

the k-means algorithm has a quadratic time complexity $O(n^2)$ [103].

The k-means algorithm was trained on a GPU by using the *Faiss* library from Facebook, which is a library for efficient similarity search and clustering. Euclidean distance is used as similarity measure between centroids and data points. At each iteration, each centroid is updated by taking the mean of all data points assigned to the same cluster.

After having clustered the data, centroid vectors are used to assign the closer centroid to every example on the training, validation and test datasets. For assigment, a 1-Nearest Neighbor search (kNN with k = 1) between examples and centroids was applied.

LOB state vectors were replaced by the assigned cluster number. A partial sequence is shown Figure 5.3. From here on, this representation of the data is referred as the tokenized version.

Sequence = [21761, 15559, 15559, 21157,7500, 793, 793, 1304, 15559, 3424, 3424,...]

Figure 5.3: Example of a tokenized sequence.

Chapter 6

Experimental setup

In this chapter, a detailed description is given in regards of how the models were trained, the experiments performed, and the data used. As it was mentioned before, all proposed models and data representations were evaluated on the mid-price movement prediction task. For a detailed description of the task, the reader is referred to Chapter 4.

6.1 Input data

All available data comprises the period spanning from August 25th to October 25th, 2020 (62 days in total). The data was divided onto three splits: training, validation and test. The training set is used for fitting the model, validation set is used for hyperparameter optimization and model selection during training, and the test set is used for reporting an unbiased evaluation for the models trained [59].

Splits were made as to keep the time occurrence of the data, and to avoid the look-ahead bias [94], i.e., the use of future information not available at some point in time for fitting a model or statistical measure. Then, the data for the training set goes from October 25th to August 23rd, August 24th for validation set, and August

25th for the test set. The distribution of each class label for the different prediction horizons k is shown in Table 6.1.

Table 6.1: Class distribution of dataset splits. Values are given as a percentage (%) of the total number of examples.

Horizon (k)	Stationary	Up	Down
Training set			
10	57.06	22.25	20.69
20	48.26	26.71	25.03
50	37.76	31.76	30.48
100	29.81	35.53	34.66
Validation set			
10	68.07	17.08	14.85
20	60.39	21.07	18.54
50	50.08	26.31	23.61
100	41.51	30.97	27.52
Test set			
10	66.83	17.37	15.80
20	59.42	21.31	19.27
50	49.58	26.22	24.20
100	41.02	30.72	28.26

The number of training examples is: 6,217,096, validation examples: 99,852, and test examples: 112,279. The whole dataset was preprocessed as per the methods described in Chapter 4 and Chapter 5, leading to a total of five different dataset configurations: baseline (z-score normalization), stationary, and tokenized with its three representations: 3k, 30k and 300k tokens.

Each example is a sequence of 100 time steps; for the baseline and stationary configurations, each time step is a vector of 40 features, while for the tokenized representation, each time step corresponds to a single token.

6.1.1 Data feeding

Training data was fed to the models in batches of size 512, thus, optimization was done by using the mini-batch gradient descent technique, which allows to take advantage of the parallelization of matrix operations while making updates to the

model's weights without using the whole training set [55].

Different batch sizes were evaluated and it was determined that 512 allowed the batched data to fit on memory while reducing the training time as compared to smaller batch sizes. There was not a significant difference in the model's performance convergence when compared to smaller batch sizes.

Training set size is about 100GB, and did not fit on memory. Data loaders were used for loading training examples directly from disk. Data loading and feeding was improved by using a total of 32 workers for preparing the data for the next batch to be fed to the model. Each worker prepared a batch of 512 examples.

Training data was shuffled at the end of each epoch to reduce variance and avoid overfitting [60].

6.2 Experimental design

For the experimental design, the dependent variable was identified as the model's performance as measured by the mid-price movement prediction task. Here the evaluation metric is the F1-score.

Experiments were designed as to assess the effect of varying different independent variables on the final outcome for the dependent variable, i.e., the F1-score. Then, the experiments have several explanatory variables and are listed below:

- 1) The input data.
- 2) Number of tokens (on the tokenized representation).
- 3) The model's architecture.
- 4) The prediction horizon.

The motivation behind each one of the independent variables selected was justified to directly answer some of the research questions stated at the beginning of

the research. Some other variables like the model's architecture or the prediction horizon do not answer directly any research question, however, visualizing the effect caused is of high importance for any trading practitioner as well as for future research directions.

Three input data representations are used: baseline, stationary and tokenized, for the tokenized representation, three different number of tokens are used (3k, 30k, and 300k). Two architectures are evaluated: LSTM and Transformer. And finally, four prediction horizons are investigated (10, 20, 50, and 100 events). Hence, the total number of experiments performed were 40.

Two additional experiments were performed by selecting the best model (with the highest F1-score) out the 40 experiments. On the first additional experiment, the best model was trained with the addition of order flow and time elapsed features between events. And for the second additional experiment, an augmented dataset (five times larger) was fed to the model. Under this new setup, order flow, order arrival time and size of training dataset effects were then explored as well.

6.3 Software and hardware

Tasks such as code development, data preprocessing, training and visualization, were performed by using the Python programming language [104], and making use of out-of-the box machine/deep learning libraries/frameworks and scientific packages such as Pytorch [105], Faiss [106], scikit-learn [107], Matplotlib [108], seaborn [109], NumPy [110] and pandas [111]. Some other support libraries were used but not mentioned here.

Training was performed on a Tesla v100-sxm2-32gb GPU. GPU resources were used whenever it was beneficial to use them, even for some not training related tasks (e.g., performing mathematical operations with tensors).

6.4 Training setup

6.4.1 Objective function

Models were trained as to reduce the categorical cross-entropy loss (see section 3.4.2 for more details) for the three possible labels in the data: Stationary, Up and Down. As it was shown in Table 6.1, all dataset splits suffer from class imbalance.

In order to mitigate the overrepresentation and to avoid the models to learn a mapping biased towards the majority class, the inverse class frequency of each class were passed as weights to the loss function. The loss function was multiplied by its respective weight value in an effort to balance the training attention [112, 113].

6.4.2 Optimizer

The Adam optimizer [114] was used for updating the network weights. Default parameters as suggested in [114] were used; learning rate $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. During training it was observed that keeping a fixed learning rate was not suitable along all the training epochs, for this reason a learning rate scheduler was adopted instead.

The scheduler *ReduceLROnPlateau* available in [105] was used to reduce the learning rate by a factor of 0.5 every time the validation loss at the end of the epoch did not improved for a *patience* number of epochs. *Patience* was set to 1, and a *cooldown* value of 1 was given as well. *Cooldown* parameter controls the number of epochs to wait just after the learning rate has been reduced.

A gradient norm clipping strategy [115] was also applied for mitigating the exploding gradients problem commonly found when training RNNs, *max norm* was set to a value of 0.5.

6.4.3 Hyperparameter tuning

Best hyperparameters were selected after performing a grid search over a discrete set of values. The hyperparameter selection was evaluated on the validation set, and the best hyperparameters were those were the F1-score (see section 3.9.4 for more details) in the validation set was the highest among all possible values. A summary of the selected hyperparameters for the LSTM-based architecture and the transformer-based architecture is shown in Table 6.2.

Table 6.2: Models' parameters and hyperparameters after a grid search.

Architecture	Parameters & Hyperparmeters
LSTM	$input_size = 40$ $hidden_size = 64$ $num_layers = 2$ $num_classes = 3$ $dropout_p = 0.5$ $embedding_dim = 8$
Transformer	$d_model = 8$ or 40 $dim_feedforward = 32$ $num_layers = 2$ $nhead = 4$ $dropout = 0.1$ $num_classes = 3$

6.4.4 Model selection

Once best hyperparameters were chosen, the model on each experiment was trained for an indefinite number of epochs. At the end of each epoch, the model was evaluated on the validation set, if the resulting F1-Score was higher than any past F1-Score obtained, the model was saved as the new best model. A value of 0.0001 was set as the minimum change for the F1-Score to be considered the new best model.

Training was stopped once the model did not improved for 20 consecutive epochs ($patience = 20$) as per F1-score metric in the validation set. Best models were saved for final evaluation on the test set.

Chapter 7

Results

On this chapter, the research results are presented. Results are given for each one of the experiments performed. Some extra results are also shown for the best model found. Contents on this chapter would then serve as a support when answering the research questions made at the beginning. Discussion and an in-depth analysis of the results are presented in next chapter.

7.1 Experimental results

After having performed the experiments discussed in the Experimental setup chapter (a total of 40 + 2 additional experiments), the evaluation metrics are reported for the different prediction horizons in Tables 7.1, 7.2, 7.3, 7.4 and 7.5. For each one of the experiments, metrics are reported by evaluating on the test set the best model found during training.

On all horizons, highest performances were achieved by experiments using a LSTM-based architecture along with a stationary representation of the data. The least favored experiments were those were a transformer-based architecture along with a tokenized representation were combined. It was also noted that increasing

Table 7.1: Experimental results for a prediction horizon $k = 10$

Input Data	Model	Accuracy	Precision	Recall	F1-score
Baseline	LSTM	79.41	69.23	73.65	71.01
	Transformer	73.32	61.38	64.32	62.55
Stationary	LSTM	**84.03**	**75.58**	**79.03**	**77.12**
	Transformer	81.98	72.91	72.83	72.86
Tokenized 3k	LSTM	75.01	63.00	66.47	64.33
	Transformer	74.68	61.42	62.33	61.85
Tokenized 30k	LSTM	74.15	62.02	65.70	63.33
	Transformer	73.58	60.77	63.21	61.73
Tokenized 300k	LSTM	72.42	59.65	62.87	60.75
	Transformer	70.16	57.58	60.97	58.50

Table 7.2: Experimental results for a prediction horizon $k = 20$

Input Data	Model	Accuracy	Precision	Recall	F1-score
Baseline	LSTM	80.10	73.60	75.08	74.28
	Transformer	74.26	66.43	66.95	66.63
Stationary	LSTM	**84.07**	**78.93**	**79.49**	**79.20**
	Transformer	81.03	75.61	73.57	74.41
Tokenized 3k	LSTM	75.13	66.99	68.45	67.59
	Transformer	73.85	64.71	64.16	64.40
Tokenized 30k	LSTM	74.62	66.22	67.61	66.80
	Transformer	73.53	64.39	64.26	64.32
Tokenized 300k	LSTM	72.56	63.40	64.58	63.88
	Transformer	73.53	64.39	64.26	61.48

Table 7.3: Experimental results for a prediction horizon $k = 50$

Input Data	Model	Accuracy	Precision	Recall	F1-score
Baseline	LSTM	77.77	74.43	74.33	74.38
	Transformer	72.42	68.79	67.47	67.77
Stationary	LSTM	**81.35**	**78.80**	**77.97**	**78.34**
	Transformer	78.77	76.00	74.28	74.97
Tokenized 3k	LSTM	73.79	69.34	69.35	69.33
	Transformer	71.99	67.07	65.92	66.33
Tokenized 30k	LSTM	72.83	68.10	68.03	68.06
	Transformer	71.27	66.23	65.68	65.77
Tokenized 300k	LSTM	70.62	65.56	65.67	65.59
	Transformer	68.78	63.26	63.31	63.27

Table 7.4: Experimental results for a prediction horizon $k = 100$

Input Data	Model	Accuracy	Precision	Recall	F1-score
Baseline	LSTM	75.13	73.71	73.40	73.47
	Transformer	69.72	68.15	67.59	67.55
Stationary	LSTM	**78.40**	**77.41**	**76.66**	**76.87**
	Transformer	74.97	74.26	72.51	72.82
Tokenized 3k	LSTM	70.94	68.79	68.75	68.70
	Transformer	69.58	64.76	64.63	64.51
Tokenized 30k	LSTM	70.14	67.82	67.75	67.70
	Transformer	68.70	66.20	65.91	65.81
Tokenized 300k	LSTM	67.79	65.14	65.16	65.05
	Transformer	67.38	64.76	64.63	64.51

the number of tokens has an inverse effect on all reported metrics.

For the F1-Score metric, the best predictive model was obtained when forecasting the price movement direction at a horizon of $k = 20$. However, for the vast majority of models tested, a horizon of $k = 50$ was optimal. This can also be noted by visually inspecting Figure 7.1.

Figure 7.1: F1-Score values for different horizons. Dotted line shows the mean F1-Score of the models per horizon.

Table 7.5 shows the metrics for the two additional experiments. A comparison against the best model obtained from the initial set of experiments shows that very small improvements were attained by adding the order flow or applying data augmentation on the training set. However, its statistical significance will be discussed

next.

Table 7.5: Additional experimental results for a prediction horizon $k = 20$

Input Data	Model	Accuracy	Precision	Recall	F1-score
Stationary	LSTM	84.07	78.93	79.49	79.20
Stationary + Order Flow	LSTM	84.14	79.05	79.49	79.26
Stationary + Data Augmentation	LSTM	**84.21**	**79.13**	**79.68**	**79.39**

7.1.1 Statistical test

The McNemar's test was used to compare the results of the experiment with the best performance against the rest of the classifiers. This test allows to quantify if the differences are statistically significant or they were just obtained by mere luck. Some other statistical tests (e.g., K-Fold Cross-Validated Paired t Test, 5×2 cv Paired t Test, 5×2 cv Paired F Test, among others) can be performed in order to compare two classification algorithms [59]. However, McNemar's test is better suited for deep learning models when the computational cost of running multiple copies of a classifier is very expensive [116]. Such is the case of the models trained in this research, where for each model the training lasted several days.

Even though the focus was on comparing the best model against the rest of the classifiers, a one-to-one test was also performed for all classifiers and horizons k. Results are shown in Figure 7.2. Test results showed that for the classifier with the highest performance, the differences with the rest of the classifiers in the F1-Score metric are all statistically significant at a significance level $\alpha = 0.01$. The vast majority of results for the different classifiers are also statistically significant.

A final statistical test was also applied for the results obtained on the additional experiments: 1) adding the order flow and 2) a data augmentation on the training set. Results are shown in Figure 7.3. Just the data augmentation experiment resulted to be statistically significant when compared with the best classifier (with the original smaller training set).

Figure 7.2: McNemar's Test results for all classifiers at a significance level $\alpha = 0.01$. Orange squares indicate an acceptance of the null hypothesis H_0, blue squares indicate a rejection of H_0.

Figure 7.3: McNemar's Test results for additional experiments at a significance level $\alpha = 0.01$. Orange squares indicate an acceptance of the null hypothesis H_0, blue squares indicate a rejection of H_0.

7.2 Training monitoring results

Training time varied from experiment to experiment, even though they were all trained on the same amount of training examples. Table 7.6 shows at what epoch the

best model was found for each experiment. An early overfitting effect was observed on those experiments using a tokenized representation of the data, and it was even more severe on experiments using a transformer-based architecture as well. The higher the number of tokens, the earlier the model started to overfit the training dataset.

Table 7.6: Epoch number where the best model as measured by the F1-Score was found.

		$k = 10$	$k = 20$	$k = 50$	$k = 100$
Input Data	**Model**	**Number of Epochs**			
Baseline	LSTM	42	37	30	33
	Transformer	10	21	13	5
Stationary	LSTM	24	40	36	32
	Transformer	10	14	19	5
Tokenized 3k	LSTM	19	28	28	23
	Transformer	14	8	12	13
Tokenized 30k	LSTM	10	15	15	12
	Transformer	11	6	3	7
Tokenized 300k	LSTM	7	5	6	7
	Transformer	2	2	1	2

7.3 Results of experiment with the highest performance

As mentioned above, the best experiment results measured by the F1-Score corresponds to a LSTM-based architecture and a stationary representation of the data. Detailed classification results are shown in Table 7.7 and in the normalized confusion matrix shown in Figure 7.4. Diagonal values in Figure 7.4 represent per class accuracy.

Table 7.7: Classification report for best model.

Class	Precision	Recall	F1-Score
Stationary	92.43	91.19	91.80
Up	73.71	74.03	73.87
Down	70.66	73.24	71.93
Macro avg	78.93	79.49	79.20
Accuracy			84.07

Figure 7.4: Confusion Matrix of best model.

Predictions from the best model are shown in Figure 7.5 for a small portion of the test set. As expected, and judging by the price fluctuations shown in the chart, periods of higher volatility seem to be harder to be predicted correctly.

Figure 7.5: Movement direction predictions on a 137.07s time series from the test set. Green vertical lines indicates a correct prediction, red vertical lines indicates an incorrect prediction. A total of 250 predictions are made.

Best model converged at very early steps during the training phase as shown in Figure 7.6. Differences on the loss or F1-Score metric between the validation and the training set are mainly due because of the different class distribution of each set.

Performance remained on test set as well, showing a great generalization capacity.

Figure 7.6: Learning curves for best model. Loss and F1-Score are shown per batch, 400 Simple Moving Averages (SMA) for the loss and F1-Score are also shown. Validation Loss and F1-Score were computed at the end of each epoch.

Chapter 8

Discussion, conclusions and future work

In this chapter, the author takes a deeper look at the results in order to give the reader a clearer depiction of what the obtained results represent for the task at hand. The research questions are also brought back to discussion in order to objectively analyze whether or not such questions have been addressed during this research. At the end, final conclusions and possible future paths of research are discussed as well.

8.1 Discussion

After having developed and executed the different experiments for the mid-price movement prediction task, some key insights can be drawn from their results. As stated before, the experiments had several independent variables for which the effects could be studied. Such effects will be analyzed next.

As per recent research works [117, 118], one could expect an increase in terms of performance by using transformers as compared to LSTM or RNNs. However, it

was not the case for this specific task. It was noted that the LSTM-based architecture outperforms the transformer-based architecture for the task at hand.

The difference in performance mentioned above might be because the contextualized vectors from the transformer model are computed from sequences of tokens that not always have the same ordering, even though their meaning could be the same. This is because the LOB data at this scale presents chaotic behavior in the way that new orders arrive at the market, making it difficult to extract representative contextualized embeddings. On the other hand, the inability of the LSTM of handling long sequences (because of the vanishing gradient problem) could have played in favor of the LSTM architecture, allowing it to forget relationships far away from the end of the sequence, and putting more weight on those that were closer to the end, which under this context, could be beneficial.

Another key insight observed was the comparison in performance for the type of input data regardless of the architecture used. Three input types were tested, the baseline (just a z-score normalization) and the proposed representations: stationary and tokenized. As per the results obtained, these could be ranked as follows: 1) stationary, 2) baseline, and last, 3) tokenized.

The scale invariant state-based nature of the stationary representation was reflected on the performance, suggesting that a better representation of the data could be achieved by considering the LOB data as a transition of states without taking into account the relative magnitude of the vector, but its direction.

On the contrary, the nature of the baseline input type relies on the relative magnitude of the features as compared to the overall distribution of observed values. During the learning phase, the attention is centered around deviations from the computed statistics, without considering that some shifts on the statistical properties of the series could change over time [119].

Finally, results were less favorable for the tokenized representation. Under this

input data type, the aim was to find a simplified representation of the multi-variate time series. However, such simplification lead to underperforming those models with a full representation of the state space. This suggests that some information with predictive power was lost while creating the tokenized version of the data. Such vital information could be lost when applying the k-means algorithm since the clustering was performed in a task-agnostic fashion, i.e., without optimizing it for the mid-price movement prediction task.

It was also observed in the tokenized representation that having a very large number of tokens does not get positively reflected in performance. For the three possible number of tokens proposed: 3k, 30k and 300k, the best performance was achieved with 3k number of clusters. Larger number of clusters showed stronger overfitting effects while training. In order to see how much the number of tokens could be decreased and still get an increase in performance, two more experiments (although not included on the reports) were performed with 300 and 30 tokens. Results suggested that an optimal value is to be between 300 and 30k number of tokens.

Another insight worth discussing is the prediction horizon. Four different prediction horizons k were tested, and as per the results chapter, in general, for all experiments the best performances were achieved for a horizon of 50 events on the future. By observing Figure 7.1, the same pattern is observed through all prediction horizons, meaning that the relative performances between the experiments are in general stable along that dimension. However, for the tokenized-transformer combination, increasing the prediction horizon might seem to have direct proportionality on the performance.

The above results might be conterintuitive since one might expect to get better results at shorter horizons, and a expected decrease for longer ones. However judging by the results there is a trade-off between that parameter. Having a very

small prediction horizon could be as hard as predicting the next event price-flip as presented in [49] where just the stationary class was predicted with high F1-Score value. At smaller horizons, Up or Down movements seem to be more unpredictable than when a stronger trend at a longer horizon is established. This might indicate the underlying reaction speed of the market to current market information, which for the cryptocurrency market might be reflected in average to more or less 50 events in the future. At longer horizons the prediction capacity diminishes. Although longer sequences and higher label classification thresholds could permit an effective prediction for more distant horizons.

As for the additional experiments, no improvements were observed by including order flow data to the LOB data. This might indicate that the former lacks of informational content at this prediction horizon scale, or it might be that it is not an orthogonal source of information that could lead to a boost in performance. However, when increasing the data size of the training set, a statistically significant improvement was achieved (see Figure 7.3 in Chapter 7). This suggests that the models could still benefit from a larger and varied training dataset.

Another comparison was made to measure the degree of association between the different classifier predictions, the Cramér's V measure of association was utilized. As per Figure 8.1, it can be observed that for all horizons k, the highest correlations were found between the configurations: "stationary-LSTM and stationary-transformer" and " stationary-LSTM and baseline-LSTM".

For each pair the common elements are "stationary" and "LSTM" respectively. Hence, one can conclude that this high correlation can be attributed to the use of these elements. Looking back at the best performance obtained from the experiments, "stationary-LSTM" combination was the one with the highest performance, each of these two elements contributed for such results.

For the set of experiments on this research, it was observed that in general the

Figure 8.1: Cramér's V results. Values range from 0 to +1 (inclusive). Higher values indicates and stronger association between the variables.

less the degree of correlation between two classifiers, the greater the differences in performance. Even with this in mind, combining a number of classifiers sufficiently diverse would in general get better performance than the best individual predictor [120]. However, after performing 120 different combinations for ensembles (with hard voting and soft voting) of three different classifiers, none of those performed better that the best individual classifier.

This would in turn mean that different models learned more or less the same patterns on the data with a different degree of effectiveness. Some other orthogonal source of information might be needed in order to obtain more varied classifiers.

A hypothetical perfect ensemble can be estimated via the Maximum Possible Accuracy (MPA) [121]. This measure is computed by counting the number of correctly classified examples by at least one classifier, and dividing this quantity by the total number of examples. For the prediction horizon where the model with the highest performance was found, i.e., $k = 20$, the MPA is 94.47%, while the accuracy for the best model is 84.07%. This means that there is approximately a 10% improvement to be made if a perfect ensemble could be achieved. Perhaps such ensemble might not be possible even if an ensemble learning algorithm is used [55].

Finally, in order to assess the stability of the results during a full trading day, the F1-Score was computed in moving windows of 100 predictions for October 25th, 2020. The distribution of this metric is shown in Figure 8.2. Distribution shows low variance around the mean. If applied to a trading strategy this is a desirable property since it is directly related (among other factors) to very important risk statistics such as the maximum drawdown and longest drawdown period [86, 122].

Figure 8.2: Probability distribution of the F1-Score metric for a window size of 100 continuous predictions during October 25th, 2020.

On a final note, results derived from this research can not be fairly compared to previous works mentioned in the literature review chapter. To begin with, all of

them use a different dataset, sampling method, prediction horizon, length of look-back window, thresholds for classifying the examples, class balances, amount of data, financial market, etc.

However, leaving all these differences aside, if all the reported macro F1-Scores obtained in the previous works were to put together and compared with best F1-Score obtained in this research, i.e., 79.21%, the rank of the present best result would be positioned on the top-quartile of predictors, i.e., among the 25% best classifiers.

8.2 Conclusions and future work

Up to this point, research questions have been implicitly answered. Here, a final and explicit answer is given for each one of them.

Does a scale invariant state-based representation of the high-frequency data provides improved results on the mid-price movement prediction task?

Results demonstrated to have an improvement in performance from a practical point of view. Statistical tests also resulted to be significant against no scale invariant state-based representations of the high-frequency data.

Do the intention and perception of market participants can be captured and synthesized, through a modeling of the high frequency data using a WE representation and perform competitively on prediction tasks?

The answer for this research question is that no competitive models were generated by using a WE representation of the data (for the tokenized version). This might indicate that further research is to be made on this area, or, no semantic similarities exists on the data to effectively construct a WE representation.

Is the cryptocurrency market predictable at very short-time horizons?

In order to evaluate the market predictability one could use the performance of the models as a proxy. Thus, by comparing the performance obtained by the

classifiers (best was F1-Score of 79.20%) against simpler classifiers, take for instance the performance of the ZeroR classifier (which always predict the majority class) which gets a F1-Score of 24.84%, or a classifier that always predicts the direction of the last realized movement (e.g., if the market moved Up, then the market would continue to move Up for the next prediction) which gets a F1-Score of 51.42%. In conclusion, the cryptocurrency market is predictable at very short-time horizons.

In this thesis new data representations of high frequency data were proposed and evaluated. Overall results confirmed the market predictability in cryptocurrency markets as well as the DL model's capacity for solving real complex problems in financial markets as it is the mid-price movement prediction task.

In the context of a trading strategy, the scope of this work extents up to the signal generation. Nonetheless, further research could be made as to assess the possible applicability of the models on a trading environment, where risk factors as well as trading fees are brought onto the big picture.

Some other future work directions can be taken. For instance, for the tokenized representation, different clustering approaches or distance metrics (other than Euclidean) could be evaluated. Or a task-specific clustering could be learned while training as well. Also, the methods here proposed could be extended to other financial markets (Stock Market, Forex Market, etc) and evaluate their performances. In a more ambitious setting, transfer learning from the cryptocurrency market can be also explored.

From the MPA metric it was observed that there is some room for improvement when creating an ensemble. For this reason it would be worth trying different ensemble learning algorithms in order to try to close this gap in performance. From a trading perspective, it could beneficial trying to also predict the magnitude of the movements and not just the direction. This could be used to determine the optimal position size when trading.

Finance applications and DL are meant to be together for the long run. Both complement each other's needs in many ways that no many existing areas of application share. With the continuous advancement of DL and AI in general, financial markets are set to be reshaped in the years to come.

References

[1] R. W. Melicher and E. A. Norton, *Introduction to finance: Markets, investments, and financial management*. John Wiley & Sons, 2013.

[2] P. P. Drake and F. J. Fabozzi, *The basics of finance: An introduction to financial markets, business finance, and portfolio management*. John Wiley & Sons, 2010, vol. 192.

[3] A. ElBahrawy, L. Alessandretti, A. Kandler, R. Pastor-Satorras, and A. Baronchelli, "Evolutionary dynamics of the cryptocurrency market," *Royal Society open science*, vol. 4, no. 11, p. 170623, 2017.

[4] M. Levinson, "Guide to financial markets: Why they exist and how they work, 6," *Aufl, New York*, 2014.

[5] G. C. Sanger and J. J. McConnell, "Stock exchange listings, firm value, and security market efficiency: The impact of nasdaq," *Journal of Financial and Quantitative Analysis*, vol. 21, no. 1, pp. 1–25, 1986.

[6] G. N. Gregoriou, *Handbook of high frequency trading*. Academic Press, 2015.

[7] J. Han, J. Hong, N. Sutardja, and S. F. Wong, "Machine learning techniques for price change forecast using the limit order book data," Working Paper, Tech. Rep., 2015.

[8] A. N. Kercheval and Y. Zhang, "Modelling high-frequency limit order book dynamics with support vector machines," *Quantitative Finance*, vol. 15, no. 8, pp. 1315–1329, 2015.

[9] D. Easley, M. M. L. de Prado, and M. O'Hara, *High-frequency trading: New realities for traders, markets and regulators*. Incisive Media, 2013.

[10] I. Aldridge, *High-frequency trading: a practical guide to algorithmic strategies and trading systems*. John Wiley & Sons, 2013, vol. 604.

[11] O. Bustos and A. Pomares-Quimbaya, "Stock market movement forecast: A systematic review," *Expert Systems with Applications*, vol. 156, p. 113464, 2020.

[12] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, "Financial time series forecasting with deep learning: A systematic literature review: 2005–2019," *Applied Soft Computing*, vol. 90, p. 106181, 2020.

[13] J. Huang, J. Chai, and S. Cho, "Deep learning in finance and banking: A literature review and classification," *Frontiers of Business Research in China*, vol. 14, pp. 1–24, 2020.

[14] J. A. Sirignano, "Deep learning for limit order books," *Quantitative Finance*, vol. 19, no. 4, pp. 549–570, 2019.

[15] Z. Zhang, B. Lim, and S. Zohren, "Deep learning for market by order data," *arXiv preprint arXiv:2102.08811*, 2021.

[16] A. Ntakaris, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods," *Journal of Forecasting*, vol. 37, no. 8, pp. 852–866, 2018.

[17] D. T. Tran, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Tensor representation in high-frequency financial data for price change prediction," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2017, pp. 1–7.

[18] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj, "Temporal attention-augmented bilinear network for financial time-series data analysis," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 5, pp. 1407–1418, 2018.

[19] P. Nousi, A. Tsantekidis, N. Passalis, A. Ntakaris, J. Kanniainen, A. Tefas, M. Gabbouj, and A. Iosifidis, "Machine learning for forecasting mid-price movements using limit order book data," *Ieee Access*, vol. 7, pp. 64 722–64 736, 2019.

[20] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Forecasting stock prices from the limit order book using convolutional

neural networks," in *2017 IEEE 19th Conference on Business Informatics (CBI)*, vol. 1. IEEE, 2017, pp. 7–12.

[21] ——, "Using deep learning to detect price change indications in financial markets," in *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE, 2017, pp. 2511–2515.

[22] J. Wallbridge, "Transformers for limit order books," *Econometric Modeling: Capital Markets - Asset Pricing eJournal*, 2020.

[23] R. Jha, M. De Paepe, S. Holt, J. West, and S. Ng, "Deep learning for digital asset limit order books," *Available at SSRN*, 2020.

[24] T. Fletcher and J. Shawe-Taylor, "Multiple kernel learning with fisher kernels for high frequency currency prediction," *Computational Economics*, vol. 42, no. 2, pp. 217–240, 2013.

[25] J. Han, J. Hong, N. Sutardja, and S. F. Wong, "Machine learning techniques for price change forecast using the limit order book data," 2015.

[26] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Using deep learning for price prediction by exploiting stationary limit order book features," *Applied Soft Computing*, vol. 93, p. 106401, 2020.

[27] J. Doering, M. Fairbank, and S. Markose, "Convolutional neural networks applied to high-frequency market microstructure forecasting," in *2017 9th computer science and electronic engineering (ceec)*. IEEE, 2017, pp. 31–36.

[28] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Temporal bag-of-features learning for predicting mid price movements using high frequency limit order book data," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 6, pp. 774–785, 2018.

[29] ——, "Temporal logistic neural bag-of-features for financial time series forecasting leveraging limit order book data," *Pattern Recognition Letters*, vol. 136, pp. 183–189, 2020.

[30] Z. Zhang, S. Zohren, and S. Roberts, "Deeplob: Deep convolutional neural networks for limit order books," *IEEE Transactions on Signal Processing*, vol. 67, no. 11, pp. 3001–3012, 2019.

[31] A. Brabazon, P. Lipinski, and P. Hamill, "Characterising order book evolution using self-organising maps," *Evolutionary Intelligence*, vol. 9, no. 4, pp. 167–179, 2016.

[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[34] B. G MALKIEL, "A random walk down wall street the time-tested strategy for successful investing," 2021.

[35] E. F. Fama, "Random walks in stock market prices," *Financial analysts journal*, vol. 51, no. 1, pp. 75–80, 1995.

[36] ——, "Efficient capital markets: A review of theory and empirical work," *The journal of Finance*, vol. 25, no. 2, pp. 383–417, 1970.

[37] E. Chong, C. Han, and F. C. Park, "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies," *Expert Systems with Applications*, vol. 83, pp. 187–205, 2017.

[38] J. Y. Campbell and M. Yogo, "Efficient tests of stock return predictability," *Journal of financial economics*, vol. 81, no. 1, pp. 27–60, 2006.

[39] R. H. Shumway and D. S. Stoffer, *Time series analysis and its applications: with R examples*. Springer, 2017.

[40] J. J. Murphy, *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin, 1999.

[41] B. Graham and D. L. Dodd, *Security analysis*. McGraw-Hill Companies, Inc, 2009.

[42] J. C. B. Gamboa, "Deep learning for time-series analysis," *arXiv preprint arXiv:1701.01887*, 2017.

[43] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang, "Deep learning with long short-term memory for time series prediction," *IEEE Communications Magazine*, vol. 57, no. 6, pp. 114–119, 2019.

[44] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Deep learning for event-driven stock prediction," in *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[45] M. Hiransha, E. A. Gopalakrishnan, V. K. Menon, and K. Soman, "Nse stock market prediction using deep-learning models," *Procedia computer science*, vol. 132, pp. 1351–1362, 2018.

[46] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, "Predicting stock market index using fusion of machine learning techniques," *Expert Systems with Applications*, vol. 42, no. 4, pp. 2162–2172, 2015.

[47] O. Hegazy, O. S. Soliman, and M. A. Salam, "A machine learning model for stock market prediction," *arXiv preprint arXiv:1402.7351*, 2014.

[48] A. Sharma, D. Bhuriya, and U. Singh, "Survey of stock market prediction using machine learning approach," in *2017 international conference of electronics, communication and aerospace technology (ICECA)*, vol. 2. IEEE, 2017, pp. 506–509.

[49] M. Dixon, "Sequence classification of the limit order book using recurrent neural networks," *Journal of computational science*, vol. 24, pp. 277–286, 2018.

[50] J. Sirignano and R. Cont, "Universal features of price formation in financial markets: perspectives from deep learning," *Quantitative Finance*, vol. 19, no. 9, pp. 1449–1459, 2019.

[51] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Adaptive normalization for forecasting limit order book data using convolutional neural networks," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 1713–1717.

[52] J. McCarthy, M. Minsky, and N. Rochester, "A proposal for the dartmouth summer research project on artificial intelligence," 1955.

[53] F. Nelli, *Python data analytics: with pandas, numpy, and matplotlib*. Apress, 2018.

[54] T. M. Mitchell *et al.*, "Machine learning," 1997.

[55] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[56] J. D. Kelleher, *Deep learning*. Mit Press, 2019.

[57] S. Russell and P. Norvig, "Artificial intelligence: a modern approach," 2020.

[58] A. C. Müller, S. Guido *et al.*, *Introduction to machine learning with Python: a guide for data scientists*. O'Reilly Media, Inc., 2016.

[59] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[60] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[61] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[62] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for machine learning*. Cambridge University Press, 2020.

[63] C. Albon, *Machine learning with python cookbook: Practical solutions from preprocessing to deep learning*. " O'Reilly Media, Inc.", 2018.

[64] I. Zafar, G. Tzanidou, R. Burton, N. Patel, and L. Araujo, *Hands-on convolutional neural networks with TensorFlow: Solve computer vision problems with modeling in TensorFlow and Python*. Packt Publishing Ltd, 2018.

[65] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[66] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[67] D. Rao and B. McMahan, *Natural language processing with PyTorch: build intelligent language applications using deep learning*. " O'Reilly Media, Inc.", 2019.

[68] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, 2014.

[69] S. Dutta, *Reinforcement Learning with TensorFlow: A beginner's guide to designing self-learning systems with TensorFlow and OpenAI Gym*. Packt Publishing Ltd, 2018.

[70] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[71] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[72] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[73] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[74] S. Nanjundan, S. Sankaran, C. Arjun, and G. P. Anand, "Identifying the number of clusters for k-means: A hypersphere density based approach," *arXiv preprint arXiv:1912.00643*, 2019.

[75] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[76] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[77] D. Rothman, "Transformers for natural language processing build innovative deep neural network architectures for nlp with python, pytorch, tensorflow, bert, roberta, and more," 2021.

[78] S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. O'Reilly Media, 2020.

[79] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[80] F. Chollet *et al.*, *Deep learning with Python*. Manning New York, 2018, vol. 361.

[81] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[82] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *arXiv preprint arXiv:1310.4546*, 2013.

[83] H. Lane, C. Howard, and H. Hapke, *Natural Language Processing in action*. Manning New York, 2019.

[84] J. Frost, *Hypothesis testing: An intuitive guide for making data driven decisions*. Statistics by Jim Publishing, 2020.

[85] A. L. Edwards, "Note on the "correction for continuity" in testing the significance of the difference between correlated proportions," *Psychometrika*, vol. 13, no. 3, pp. 185–187, 1948.

[86] S. Jansen, *Machine Learning for Algorithmic Trading: Predictive models to extract signals from market and alternative data for systematic trading strategies with Python, 2nd Edition*. Packt Publishing, 2020.

[87] U.S. Securities and Exchange Commission, "Equity market structure literature review part ii: High frequency trading," *Staff of the Division of Trading and Markets*, 2014.

[88] M. O'hara, *Market microstructure theory*. Wiley, 1997.

[89] A. Aziz, *Advanced Techniques in Day Trading: A Practical Guide to High Probability*, 2018.

[90] M. D. Gould, M. A. Porter, S. Williams, M. McDonald, D. J. Fenn, and S. D. Howison, "Limit order books," *Quantitative Finance*, vol. 13, no. 11, pp. 1709–1742, 2013.

[91] A. Hyttinen, S. Plis, M. Järvisalo, F. Eberhardt, and D. Danks, "Causal discovery from subsampled time series data by constraint optimization," in *Conference on Probabilistic Graphical Models*. PMLR, 2016, pp. 216–227.

[92] R. Huang and T. Polak, "Lobster: Limit order book reconstruction system," *Available at SSRN 1977207*, 2011.

[93] J. Wallbridge, "Transformers for limit order books," *arXiv preprint arXiv:2003.00130*, 2020.

[94] M. L. De Prado, *Advances in financial machine learning*. John Wiley & Sons, 2018.

[95] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[96] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[97] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 463–484, 2012.

[98] J. Brownlee, *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018.

[99] P. O. Olukanmi and B. Twala, "K-means-sharp: modified centroid update for outlier-robust k-means clustering," in *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*. IEEE, 2017, pp. 14–19.

[100] A. Gersho and R. M. Gray, *Vector quantization and signal compression*. Springer Science & Business Media, 2012, vol. 159.

[101] J.-P. Braquelaire and L. Brun, "Comparison and optimization of methods of color image quantization," *IEEE Transactions on image processing*, vol. 6, no. 7, pp. 1048–1052, 1997.

[102] P. J. Dhrymes, *Mathematics for Econometrics*. Springer Science & Business Media, 2013.

[103] M. K. Pakhira, "A linear time-complexity k-means algorithm using cluster shifting," in *2014 International Conference on Computational Intelligence and Communication Networks*, 2014, pp. 1047–1051.

[104] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[105] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[106] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *arXiv preprint arXiv:1702.08734*, 2017.

[107] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[108] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[109] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://doi.org/10.21105/joss.03021

[110] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[111] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[112] C. Elkan, "The foundations of cost-sensitive learning," in *International joint*

conference on artificial intelligence, vol. 17, no. 1. Lawrence Erlbaum Associates Ltd, 2001, pp. 973–978.

[113] Z.-H. Zhou and X.-Y. Liu, "On multi-class cost-sensitive learning," *Computational Intelligence*, vol. 26, no. 3, pp. 232–257, 2010.

[114] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[115] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*. PMLR, 2013, pp. 1310–1318.

[116] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural computation*, vol. 10, no. 7, pp. 1895–1923, 1998.

[117] A. Zeyer, P. Bahar, K. Irie, R. Schlüter, and H. Ney, "A comparison of transformer and lstm encoder decoder models for asr," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 8–15.

[118] S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. E. Y. Soplin, R. Yamamoto, X. Wang *et al.*, "A comparative study on transformer vs rnn in speech applications," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 449–456.

[119] A. Nielsen, *Practical time series analysis: Prediction with statistics and machine learning*. O'Reilly Media, 2019.

[120] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.

[121] M. E. Aragón, H. J. Jarquín-Vásquez, M. Montes-Y-Gómez, H. J. Escalante, L. V. Pineda, H. Gómez-Adorno, J. P. Posadas-Durán, and G. Bel-Enguix, "Overview of mex-a3t at iberlef 2020: Fake news and aggressiveness analysis in mexican spanish." in *IberLEF@ SEPLN*, 2020, pp. 222–235.

[122] E. P. Chan, *Quantitative trading: how to build your own algorithmic trading business*. John Wiley & Sons, 2021.