Machine Learning for Forecasting Mid Price Movement using Limit Order Book Data

Paraskevi Nousi[a,*], Avraam Tsantekidis[a,*], Nikolaos Passalis[a], Adamantios Ntakaris[b], Juho Kanniainen[c], Anastasios Tefas[a], Moncef Gabbouj[b], Alexandros Iosifidis[b,d]

[a]*Department of Informatics, Aristotle University of Thessaloniki, Greece*
[b]*Laboratory of Signal Processing, Tampere University of Technology, Tampere, Finland*
[c]*Laboratory of Industrial and Information Management, Tampere University of Technology, Tampere, Finland*
[d]*Department of Engineering, Electrical and Computer Engineering, Aarhus University, Denmark*

Abstract

Forecasting the movements of stock prices is one the most challenging problems in financial markets analysis. In this paper, we use Machine Learning (ML) algorithms for the prediction of future price movements using limit order book data. Two different sets of features are combined and evaluated: handcrafted features based on the raw order book data and features extracted by ML algorithms, resulting in feature vectors with highly variant dimensionalities. Three classifiers are evaluated using combinations of these sets of features on two different evaluation setups and three prediction scenarios. Even though the large scale and high frequency nature of the limit order book poses several challenges, the scope of the conducted experiments and the significance of the experimental results indicate that Machine Learning highly befits this task carving the path towards future research in this field.

Keywords: Machine Learning, limit order book, feature extraction, mid price forecasting

1. Introduction

Forecasting of financial time series is a very challenging problem and has attracted scientific interest in the past few decades. Due to the inherently noisy and non-stationary nature of financial time series, statistical models are unsuitable for the task of modeling and forecasting such data. Thus, the nature of financial data necessitates the utilization of more sophisticated methods, capable of modeling complex non-linear relationships between data, such as Machine Learning (ML) algorithms.

*Corresponding authors; Equal contribution

Email addresses: paranous@csd.auth.gr (Paraskevi Nousi), avraamt@csd.auth.gr (Avraam Tsantekidis), passalis@csd.auth.gr (Nikolaos Passalis), adamantios.ntakaris@tut.fi (Adamantios Ntakaris), juho.kanniainen@tut.fi (Juho Kanniainen), tefas@aiia.csd.auth.gr (Anastasios Tefas), moncef.gabbouj@tut.fi (Moncef Gabbouj), alexandros.iosifidis@eng.au.dk (Alexandros Iosifidis)

Early Machine Learning approaches to this problem included shallow Neural Networks (NNs) [1, 2], and Support Vector Machines (SVMs) [3, 4, 5]. However, the lack of appropriate training and regularization algorithms for Neural Networks at the time, such as the dropout technique [6], rendered them susceptible to over fitting the training data. Support Vector Machines were deemed as better candidates for this task, as their solution implicitly involves the generalization error.

The development of effective and efficient training algorithms for deeper architectures [7], in conjunction with the improved results such models presented, steered scientific interests towards Deep Learning techniques in many domains. Deep Learning methods are capable of modeling highly non-linear, very complex data, making them suitable for application to financial data [8], as well as time series forecasting [9].

Furthermore, ML techniques which perform feature extraction may uncover robust features, better-suited to the specific task at hand. Autoencoders [10], are Neural Networks which learn new features extracted from the original input space, which can be used to enhance the performance of various tasks, such as classification or regression. Bag-of-Features (BoF) models comprise another feature extraction method that can be used to extract representations of objects described by multiple feature vectors, such as time-series [11]. The process of feature extraction is of major importance as it can significantly affect the performance of the used machine learning algorithms.

In this paper we utilize various Machine Learning algorithms and data preprocessing techniques for the prediction of future price movements of stocks. The dataset we use comes from the temporal progression of the limit order book (LOB) which is the highest resolution possible one can observe the stock markets in. The characteristics of LOB data are discussed in detail in Section 4.1, along with a comprehensive description of expertly hand-crafted features that can be extracted from a LOB.

The main contribution of this paper is a very extensive study into the significance of the information provided by the limit order book for the task of predicting future mid price movements of stocks. Handcrafted features, founded on financial expertise, and features extracted by sophisticated Machine Learning techniques are combined and utilized towards this purpose. Three scenarios are assessed regarding the span of time for which predictions are made: a scenario where the movement of the mid price of the immediately succeeding sample in time is predicted, one where the average mid price movement of the next five samples is predicted and, last, one where the average movement of the mid price for the next ten samples is predicted. Multiple ML classifiers are evaluated on two experimental setups for all three scenarios and for various combinations of the extracted features, taking into consideration the class imbalance which accompanies the data as well as the real-time requirements of High Frequency Trading. The results achieved indicate that the LOB contains valuable information which, in conjunction with various Machine Learning algorithms, can give meaningful insight into the stock market trend and lead the models to make accurate predictions, without any external human intervention. Although Machine Learning techniques have been widely used to model other types of financial data [12, 13], only recently have they begun to be applied and evaluated on LOB data [14]. To the best of our knowledge, our work constitutes the first in depth review and evaluation of ML techniques

for high frequency Limit Order Book data.

The rest of the paper is organized as follows. Section 2 presents previous related work upon which we build. A description of the data contained in a limit order book and used for this work is presented in Section 3. The ML algorithms used to extract features from the original financial data are described in Section 4, followed by the classification algorithms used for the prediction of the mid price direction. In Section 5 the experimental setup and results are described and analyzed. Finally, Section 6 summarizes the conclusions drawn from studying the application of the described Machine Learning algorithms to financial data.

2. Related Work

The dynamics of the high frequency limit order book comprise a challenging field of study which has been investigated in past literature. An extensive survey on stochastic models and statistical techniques for modeling high frequency limit order book data can be found in [15], highlighting the inadequacies of statistical models as well as the need for more complex models, such as Machine Learning ones. Statistical models often make unrealistic assumptions about the distribution of the data, such as assuming a Poisson distribution of limit order events. Machine Learning techniques make no assumptions on the distribution of the data. Furthermore, the distribution of limit order events changes rapidly, not just from one day to the next but also within the same day. It is thus very challenging for statistical models which typically assume stationary signals to be used effectively used for modeling limit order book data. The drawbacks of statistical models and advantages of Machine Learning approaches have also been examined in [14]. An extensive analysis of high frequency financial data can be found at [16], and the dynamics of limit order books are explained in detail in [15, 17].

Machine learning has been used very extensively to analyze the financial market from many different aspects. A recent study [18], pitted traditional econometric methods several ML methods against for forecasting, and find that the best of latter outperform the best of the former. In particular, NNs and SVMs have been two of the most popular techniques for this task, as indicated by the large number of works, e.g., [1, 2, 3, 4, 5, 19, 20, 21], which use models based on these architectures. In [22] an SVM model is trained to predict the direction of the movement of the NIKKEI 225 index. An SVM and Multilayer Perceptron (MLP) comparison can be found in [23], where daily direction of the price of the Korea Composite stock index is predicted, using 12 different indexes as input features. Using two different windows, one long and one short term, in order to capture both the trend and higher frequency information of the time series of treasury bond returns, [24] utilizes an MLP model and attempts to predict the movement of the future bonds returns. In [4] the authors compare the performance of SVMs, MLPs and Radial Basis Function (RBF) NNs in predicting price changes of future asset contracts. In [25] different dataset sizes are used to train a neural network model and it is shown that using too many samples that span too far into the past can degrade the prediction quality. The evaluation comparison is based on the profit that each model produces. In Deep Portfolio Theory [26], the authors use

autoencoders to optimize the performance of a portfolio and beat the profit benchmarks such as the biotechnology IBB Index.

In a similar fashion to this work, [14] uses several handcrafted time sensitive and insensitive features, extracted from the limit order book. These features include bid-ask spreads and mid prices, price differences, mean prices and volumes, along with derivatives of the price and volume, average and relative intensity indicators, totaling to 144 different features. However, in [14] the proposed methods are evaluated on a very small dataset that contains about 400,000 order book rows. In contrast, in our case a large-scale dataset that contains information for 10 days and 5 stocks is used, with the raw data being more than 4 million samples.

To the best of our knowledge, this is the first work that utilizes such a vast volume of data produced by the high frequency limit order book while attempting to forecast the mid price of the stocks described by it, by using multiple Machine Learning techniques including feature extraction methods and various classifiers. The scope of the conducted experiments and the significance of the observed results indicate that Machine Learning highly befits this task. The presented baseline results carve the path towards future research in this field that can potentially achieve even more accurate and significant predictions.

3. High Frequency Limit Order Book

In financial equity markets a limit order is a type of order to buy or sell a specific number of shares within a set price. For example, a sell limit order (ask) of \$10 with volume of 100 indicates that the seller wishes to sell the 100 shares for no less that \$10 a piece. Respectively, a buy limit order (bid) of \$10 means that the buyer wishes to buy a specified amount of shares for no more than \$10 each. An in-depth survey of the properties of the LOB can be found in [15].

Consequently, the order book, which contains the above information, has two sides: the bid side, containing buy orders with prices $\mathbf{p}_b(t)$ and volumes $\mathbf{v}_b(t)$, and the ask side, containing sell orders with prices $\mathbf{p}_a(t)$ and volumes $\mathbf{v}_a(t)$. The orders are sorted on both sides based on the price. On the bid side $p_b^{(1)}(t)$ is the highest available buy price and on the ask side $p_a^{(1)}(t)$ is the lowest available sell price.

Whenever a bid order price exceeds an ask order price $p_b^{(i)}(t) > p_a^{(j)}(t)$, where $p_b^{(i)}(t)$ is the i-th element of the bid side at time step t and $p_a^{(j)}(t)$ is the j-th element of the ask side at the same time step, they "annihilate", executing the orders and exchanging the traded assets between the investors. Typically, an order that leads to an immediate execution is called a market order. In this case, an investor makes an order to buy or sell a specific number of shares immediately, at the best available current price. Since the orders do not usually have the same requested volume, the order with the greater size remains in the order book with the remaining unfulfilled volume.

Several tasks arise from this data, ranging from the prediction of the price trend and the regression of the future value of a metric, e.g., volatility, to the detection of anomalous events that cause price jumps, either upwards or downwards. These tasks can lead to interesting

applications, such as protecting the investments when market conditions are unreliable, or taking advantage of such conditions to create automated trading techniques for profit.

The data used in this work consists of 10 orders for each side of the LOB. Each order is described by 2 values, the price and the volume, yielding a total of 40 values for each time step The stock data, provided by Nasdaq Nordic, come from the Finnish companies Kesko Oyj, Outokumpu Oyj, Sampo, Rautaruukki and Wartsila Oyj. The time period used for collecting that data ranges from the 1st to the 14th June 2010 (only business days are included), and the data is provided by the Nasdaq Nordic data feeds [27, 28]. The dataset is made up of 10 days for 5 different stocks and the total number of messages is about 4.5 million with equally many separate depths.

4. Proposed Methodology

In this Section, we briefly review the data preprocessing and the feature extraction procedure. Then, we introduce two feature learning methods that are used to *learn* low-dimensional features using the extracted handcrafted features. Finally, we introduce the classifications methods that are used to predict the mid price movements.

4.1. Handcrafted Features

The raw order book data is first preprocessed by removing the unnecessary messages from the exchange, e.g., event messages, and then the features proposed in [14] are extracted. More specifically, first, a basic set of features which includes the prices and volumes for every level of the ask and bid side of the order book is extracted. This information yields 40 values at each time step. Then, time-insensitive features describing the spread, mid-price, price and accumulated price differences between the bid and ask orders of each depth level, and price and volume spreads are extracted. Finally, time-sensitive features are extracted corresponding to the average intensity for trades, orders, cancellations, deletion, execution of visible limit orders and execution of hidden limit orders. This set of features also includes the price and volume average values at each level of the LOB, the average intensity per trading type as well as comparisons between the intensities and limit activity acceleration (derivatives of average intensities). Because of the non-linear nature of time in the LOB data, we follow an event-based inflow. The interested reader is referred to [14, 29] for a more detailed description of the extracted features. Note that one feature vector is extracted for every 10 limit order events that change the LOB, effectively subsampling the data by a factor of 10. The total number of the collected limit order events is about 4.5 million, leading to a total of 453.975 extracted feature vectors.

Instead of using only the feature vector extracted from the current time step, as proposed in [14], we propose three additional ways to extract representations capable of capturing more temporal information. Therefore, the following four different feature vectors are produced and used as inputs to the evaluated models, using a time sliding window of length 5:

1. A single feature vector with 144 values as described above, corresponding to the last sample in the sliding window (abbreviated as *last*).

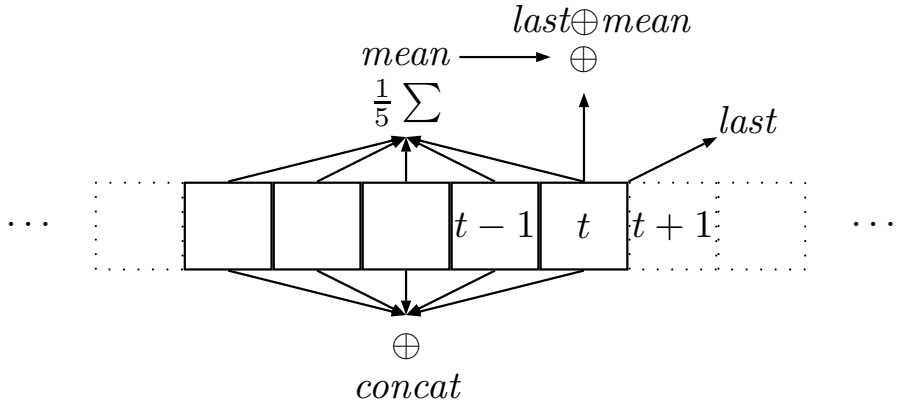

Figure 1: Representations extracted by using a sliding window of length 5 on the handcrafted feature vectors. The *last*⊕*mean* representation is the concatenation of the depicted *last* and *mean* representations.

2. The mean of the 5 samples currently in the window, which is also a 144-dimensional vector (abbreviated as *mean*).
3. The concatenation of the last sample and the mean of all 5 samples in the window, yielding a 288-dimensional feature vector (abbreviated as *last*⊕*mean* – the circled plus symbol is used to denote the concatenation operation).
4. The concatenation of all 5 samples, yielding a 720-dimensional feature vector (abbreviated as *concat*).

Figure 1 illustrates the process of obtaining the above representations.

Using this sliding window approach allows for temporal information to be incorporated into the representations, which are subsequently used to predict the movement of the stock's mid price. By averaging over the five entries contained in the window, some of the inherent noise of the features is alleviated. Concatenating different representations, e.g., the last feature vector and the average of the last 5 feature vectors, allows for more temporal information to be introduced in the final feature vector.

Since normalization is crucial for most ML techniques, we normalize the extracted features using z-score standardization:

$$\mathbf{x} = \frac{\mathbf{x}_{raw} - \bar{\mathbf{x}}_{raw}}{\sigma_{raw}} \tag{1}$$

where $\mathbf{x}_{raw}$ is the vector of values to be normalized, $\bar{\mathbf{x}}_{raw}$ is the mean vector of the data and σ_{raw} is the standard deviation vector of the data. Both the mean and the standard deviation are computed element-wise.

4.2. Prediction Labels

Of the values accompanying LOB data, the tick price, which is the price of the last executed trade, typically varies wildly between the two sides of the margins, introducing

great amounts of noise to the prediction labels. The so-called market micro-structure noise can be partially reduced by using mid-prices, i.e. the mean of the best ask and best bid prices, instead of transaction prices. Thus, in this paper, we considered mid-prices as the stock price observations. Another advantage of using mid-prices instead of transaction prices is that they are observable every time as long as there are orders on both bid and ask sides while transaction prices are updated only at transactions. The mid price $p(t)$ is defined as:

$$p(t) = \frac{p_a^{(1)}(t) + p_b^{(1)}(t)}{2} \tag{2}$$

where $p_a^{(1)}$ and $p_b^{(1)}$ are the best bid and best ask price. Note that the mid price of the stock is one of the features in the set of 144 features derived by following the feature crafting process described in [14].

An averaging filter is applied over the past N_b values (including the current time step t) of the mid price of the samples to reduce the impact of the noise in the signal:

$$m_b(t) = \frac{1}{N_b} \sum_{i=1}^{N_b} p(t - i + 1) \tag{3}$$

In our experiments we use $N_b = 9$. We have also evaluated the models for different values of N_b, i.e., $N_b = 5, 7$ and 11, obtaining similar results.

We compare this price at time t with the mean of the succeeding N_α smoothed mid prices (not including the current time step t):

$$m_a(t) = \frac{1}{N_\alpha} \sum_{i=1}^{N_\alpha} m_b(t + i) \tag{4}$$

For each sample, the movement of the mid price is defined by comparing the current smoothed mid price $m_b(t)$ to the mean of the next N_α smoothed mid prices $m_a(t)$. Small changes of the price should be considered insignificant. To this end, we introduce a parameter α to control the threshold a price movement must surpass in order to be considered as either upwards or downwards. Thus, the label $l(t)$ to be predicted for time step t is computed as:

$$l(t) = \begin{cases} 1 \ , & \text{if } m_b(t) > m_a(t) \cdot (1 + \alpha) \\ -1, & \text{if } m_b(t) < m_a(t) \cdot (1 - \alpha) \\ 0 \ , & \text{otherwise} \end{cases} \tag{5}$$

In other words, we treat the problem of forecasting the mid price movements of stocks as a classification problem with three possible outcomes: upwards movement, downwards movement and no change with labels $1, -1$, and 0 respectively as defined in Equation (5). Three sets of such labels are generated through the above process, for $N_\alpha = 1, 5$ and 10 and $\alpha = 0.0001, 0.0002$ and 0.0003 respectively. All the classifiers were extensively evaluated using these three different prediction horizons.

In practice, the mid price of a stock very rarely remains the same through consecutive time steps, thus for $\alpha = 0$ the majority of the mid price movements are categorized as either upwards of downwards, depleting the no-change class of samples. The value of the α parameter can be crucial as even small fluctuations significantly affect the balance between the three possible classes. As α increases, so does the number of samples belonging to the no-change class. Although raising this threshold would lead to a more balanced problem, i.e., all three classes would contain about the same number of samples, high values of α are undesirable as they allow upwards and downwards movements to be categorized as not having changed even though the movement could be significant. Choosing a meaningful value for α constitutes a trade-off between the balance among classes and the meaningfulness of the produced labels.

The values used for α are minuscule, which is meaningful since larger values would deprive the upwards and downwards classes of samples. On the other hand, smaller values would introduce a lot of noise in these classes by causing insignificant changes to be considered significant. Nonetheless, the selected values still produce an imbalanced dataset with most of the samples being classified as not having changed.

The resulting class imbalance constitutes a problem which must be taken into consideration by the ML algorithms used for the classification task. In this work, we deal with this problem by introducing class weights inversely proportional to the number of samples in each class. The interested reader is referred to [30], and [31], for a detailed review of techniques that can be used to deal with the problem of class imbalance.

4.3. Feature Learning

Apart from the four handcrafted feature vectors proposed in the previous subsection, two *feature learning* methods were deployed and evaluated in our experiments. The methods and the details of their application to the problem at hand are described below. The extracted representations are used as inputs to the evaluated classifiers, on their own or in combination with the described representations created from the handcrafted features.

4.3.1. Autoencoders

Autoencoders (AEs) are neural networks which map their input data to itself, through multiple levels of non-linear neurons [10, 32, 33]. Thus, the input and output layers consist of as many neurons as the dimension of the data. Such networks are comprised of an encoding part, which maps the input to an intermediate representation, and a decoding part, which maps the intermediate representation learned to the desired output and is symmetrical to the encoding part layer-wise.

Typically, an AE is used for dimensionality reduction as well as feature extraction, which means that the intermediate representation learned is lower-dimensional than the input data. The layers of both parts of the network $l = 1, \ldots, l_{enc}, \ldots, L$, where l_{enc} is the encoding layer, are accompanied by weights $\mathbf{W}^{(l)}$ which multiply each layer's input to produce an output. A bias term $\mathbf{b}^{(l)}$ is also added to the output of the neuron, and a non-linearity $s(\cdot)$ called the activation function of the neuron is applied to this output to produce the neuron's activation

value. The output $\mathbf{x}_{out}^{(l)}$ of the l-th layer is given by:

$$\mathbf{x}_{out}^{(l)} = s(\mathbf{W}^{(l)}\mathbf{x}_{in}^{(l)} + \mathbf{b}^{(l)}) \tag{6}$$

where $\mathbf{x}_{in}^{(l)}$ is the input to the l-th layer, which is equal to the output of the previous layer, or:

$$\mathbf{x}_{in}^{(l)} = \begin{cases} \mathbf{x} & , \quad l = 0 \\ \mathbf{x}_{out}^{(l-1)}, & \quad l > 0 \end{cases} \tag{7}$$

where $\mathbf{x} \in \mathbb{R}^D$ denotes an input sample. The *last* representation is used as the input to the AE, i.e., the original input dimension is $D = 144$.

The network's parameters can be learned using the well-known backpropagation algorithm [34], combined with an optimization method, such as Stochastic Gradient Descent (SGD) [35], with the final objective being the optimization of the network's loss function. Specifically for autoencoders, their objective is to minimize the reconstruction error, i.e., the squared l_2-norm between the network's output $\mathbf{x}_{out}^{(L)}$ and the desired output $\mathbf{x}$, which is the same as the network's input:

$$\ell = \|\mathbf{x} - \mathbf{x}_{out}^{(L)}\|_2^2 \tag{8}$$

The objective of the network is to minimize the mean of errors over all data samples.

As the training process converges, the activations of the intermediate layers can be used as learned feature representations of the input data. Let $\mathbf{x}_{enc}$ denote the output of the l_{enc} layer:

$$\mathbf{x}_{enc} = \mathbf{x}_{out}^{(l_{enc})} \in \mathbb{R}^d, d < D. \tag{9}$$

Then $\mathbf{x}_{enc}$ can be used as the low-dimensional representation of the data in the subsequent classification task.

4.3.2. Bag-of-Features

Bag-of-Features (BoF) models [36, 37], allow for extracting constant-length representations of samples that consist of multiple feature vectors, e.g., feature vectors extracted from various locations of an image or from various time points of a time series, such as the 5 feature vectors contained in the current sliding window. BoF models, originate from and comprise an extension of the standard Bag-of-Words model [38], which uses word frequencies as features to describe each document. Similarly, the BoF model describes each sample as a histogram over a set of predefined codewords, which is also called *codebook* or *dictionary*.

To encode our data using the Bag-of-Features model, we must first learn the dictionary. To this end, we pick a random subsample of the data and apply the k-means clustering algorithm to find K centers that best partition the data into clusters [39]. The k-means algorithm firstly picks K random cluster centers and assigns each sample to the cluster whose center lies the closest to it. The cluster centers are then updated to be the mean of the samples belonging to each cluster and the process is repeated until the centers converge. The final cluster centers $\mathbf{v}_k, k = 1, \ldots, K$ form the dictionary of the BoF model.

The learned clusters act as histogram bins in which the feature vectors are quantized. For every sample to be encoded we compute its similarity to each of the codewords (cluster centers) as:

$$d_{i,k} = \exp(\frac{-\|\mathbf{v}_k - \mathbf{x}_i\|_2}{g}) \tag{10}$$

where $\mathbf{x}_i$ is the i-th ($i = 1, \ldots, 5$) feature vector of the current window, $d_{i,k}$ denotes the k-th element of the $\mathbf{d}_i$ vector and g is a scaling parameter which controls the participation score of the feature vectors to each bin. All five samples contained in the sliding window as described in Section 4.1 are used for this process. Adjusting the scaling parameters g alters the *fuzziness* of the quantization process, with larger values leading to more fuzzy assignment [40]. Equation (10) is then normalized to have unit l_1 norm as follows:

$$\mathbf{u}_i = \frac{\mathbf{d}_i}{\|\mathbf{d}_i\|_1} \tag{11}$$

The vector $\mathbf{u}_i$ expresses the membership of the i-th feature vector to each of the clusters.

To obtain the final histogram representation $\mathbf{h}$ of the sample we average over the membership vectors $\mathbf{u}_i$:

$$\mathbf{h} = \frac{1}{N_F} \sum_{i=1}^{N_F} \mathbf{u}_i \tag{12}$$

where N_F is the length of the current window, i.e., $N_F = 5$ in this work. This histogram vector $\mathbf{h}$ can be then used for the subsequent classifications tasks alone or in combination with the other extracted feature vectors.

4.4. Classifiers

Three different classifiers were evaluated, combined with eight sensible combinations of the handcrafted input data and the features learned from that data.

4.4.1. Support Vector Machines

Support Vector Machines are linear binary classifiers of the form $f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$, whose aim is to maximize the margin of the hyperplane separating the two classes. The vector $\mathbf{w}$ is orthogonal to the separating hyperplane, while the offset of the hyperplane is determined by the value of b. Multiple such binary classifiers can be used to solve multi-class classification problems. The optimization problem can be formulated as:

$$\begin{aligned} &\min_{\mathbf{w}\in\mathbb{R}^d, \xi_i \in \mathbb{R}^+} \|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n} \xi_i \\ &\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \forall i \end{aligned} \tag{13}$$

where $\xi_i, i = 1, \ldots, n$ are slack variables which allow the classifier to achieve better generalization, C is a regularization parameter which controls the width of the margin learned

by the SVM, and $y_i \in \{-1, 1\}$ are the binary classification targets. In our experiments, the parameter C was selected by performing 3-fold cross-validation on the training set (possible C values range from 0.00001 to 0.1).

For a multiclass classification problem with unbalanced label distributions over the data, the multiple SVM classifiers trained can have different C values. Choosing large values for the separation of less represented classes will lead to fewer misclassified samples. This is important to avoid deteriorated class-wise performance of the classifier for the less represented classes. To account for the class-imbalance in the training set, the regularizer is set to be inversely proportional to the frequency of each class in the training dataset, i.e., $C_i = \frac{1}{3}\frac{N}{N_{c_i}}C$, where C_i is the regularizer for the SVM responsible of recognizing samples of the i-th class, N is the total number of training samples, N_{c_i} is the number of training samples that belong to the i-th class and C the global regularization parameter selected using cross-validation.

The direct solution of the above optimization problem requires the storage in memory of an $N \times N$ matrix containing the inner products between all pairs of samples, where N is the number of samples in the training set. This can prohibit the computation of the direct solution when the dataset is large, as in our case. To combat this issue, gradient descent-based optimization techniques have been utilized, including SGD [41]. The learning process of following the gradient of the optimization objective might end near but not exactly at the global minimum. However, it allows for training the classifier by using minibatches of data [42], thus leading to better generalization error by allowing for more samples to contribute towards the classifier's learning process.

4.4.2. Single Hidden Layer Feedforward Neural Networks

Non-linear kernel methods, such as Kernel SVMs [43], and Kernel Ridge Regression [44], can significantly increase the classification accuracy over their linear counterparts. However, these kernel methods are even more computationally intensive than their linear variants, requiring the calculation of the kernel matrix between all the training samples. To alleviate this problem, approximate methods have been proposed, such as Prototype Vector Machines [45], and Approximate Kernel Extreme Learning Machines [46]. These methods employ Single Hidden Layer Feedforward Neural Networks (SLFNs) to approximate the kernel solution through a non-linear hidden layer.

In this work, a max-margin SLFN formulation is used, i.e., the output layer is trained using a max-margin objective, as in [47]. First, the hidden layer weights are learned by clustering the data into N_H clusters, where the centroid $\mathbf{w}_k$ of each cluster corresponds to a prototype vector. The activation of the k-th hidden neuron $x_{hid,k}$ is calculated by measuring the similarity between the input vector $\mathbf{x}$ to each prototype vector $\mathbf{w}_k$ using a Radial Basis Function (RBF):

$$x_{hid,k} = \exp\left(\frac{||\mathbf{x} - \mathbf{w}_k||_2^2}{2\sigma^2}\right) \tag{14}$$

where σ is scaling parameter that alters the spread of RBFs. Typically σ is set to the mean distance between the input samples. Then, the output of each binary classifier is calculated as:

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x}_{hid} + \mathbf{b} \tag{15}$$

where $\mathbf{W}$ and $\mathbf{b}$ are the weights and the biases of the output layer. The output weights are learned by using gradient descent and a max-margin objective (as described in Equation (13)). As before, the regularization parameter C is selected using 3-fold cross-validation and appropriately weighted considering the distribution of the class labels.

4.4.3. Multilayer Perceptrons

Multilayer Perceptrons (MLPs) [48], also known as Multilayer Feedforward Neural Networks, consist of several layers of weighted connections through which the input is processed. A non-linear activation function is applied after each layer. The calculation of each layer's output is similar to the autoencoder's:

$$\mathbf{x}_{out}^{(l)} = s(\mathbf{W}^{(l)}\mathbf{x}_{in}^{(l)} + \mathbf{b}^{(l)}) \tag{16}$$

where $\mathbf{x}_{in}^{(l)}$ is the input to a layer, obtained by Equation (7), $s(\cdot)$ is the activation function, $\mathbf{W}^{(l)}$ is the weight matrix accompanying the current layer l, and $\mathbf{b}^{(l)}$ is the current layer's bias vector. Equation (16) is applied for every layer in the neural network up until the last layer l_{out} where each neuron represents a different class, meaning that in the l_{out} there must be as many neurons as classes in our dataset, i.e., $N_C = 3$ neurons. The softmax function $\sigma(\cdot)$ is applied to the output of the final layer of the MLP, to produce a probability distribution over the existing classes:

$$\sigma(\mathbf{x}_{out})_j = \frac{e^{x_{out,j}}}{\sum_{k=1}^{N_C} e^{x_{out,k}}} \tag{17}$$

where $\sigma(\mathbf{x}_{out})_j$ is the predicted probability for class j and $x_{out,j}$ is the output of the j-th neuron of the network.

To encode the prediction objective as a differentiable cost function, the categorical cross entropy function is used:

$$\ell = -\sum_j y_j \log \sigma(\mathbf{x}_{out})_j \tag{18}$$

where y_j is the desired output for the j-th output neuron, $\mathbf{x}_{out}$ denotes the output of the MLP, and $\sigma(\mathbf{x}_{out})_j$ is the predicted probability for the j-th class. As with AEs, the weights of each layer can be learned using SGD. However, instead of using plain SGD to optimize the parameters of the MLP, the ADAM algorithm [49], which is a more advanced optimization algorithm that allows for faster and more stable convergence, is used.

5. Experimental Evaluation

5.1. Evaluation Setup

An overview of our proposed system of analysis is shown in Figure 2. Events taking place in the market are described in the limit order book, from which we can then extract handcrafted features as described in Section 4.1 as well as target values corresponding to the movement of the mid-price. The data used consist of combinations of the handcrafted feature vectors and features learned from these as discussed in Section 4.3. The classifiers

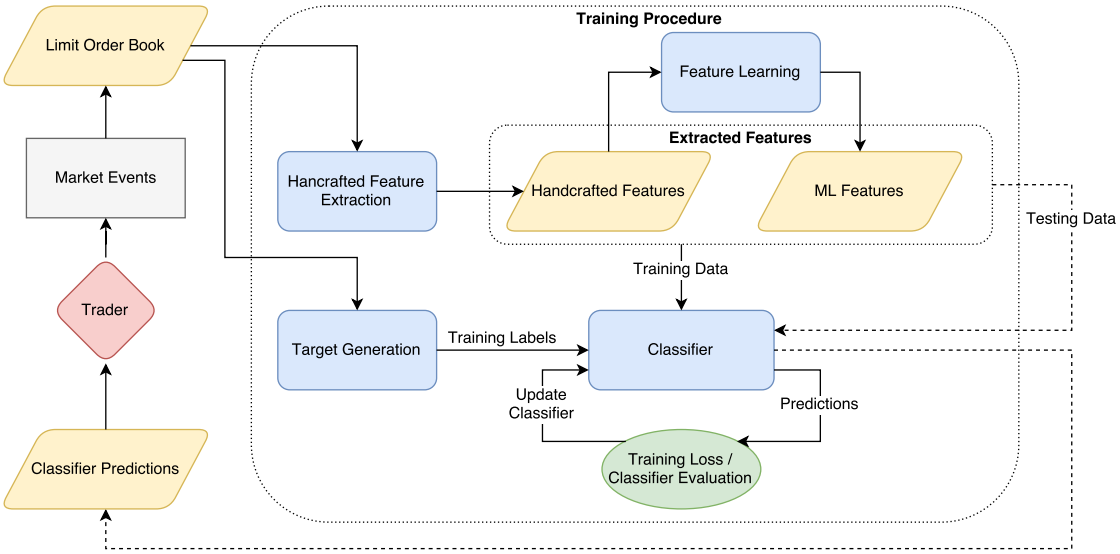

Figure 2: Pipeline of the stock mid price prediction system used. Limit order events generated by market events are used to extract feature vectors which describe the stocks present in the LOB, and a classifier is trained to recognize the movement of the mid-price of those stocks. Dashed lines indicate the deployment phase, where a trader receives classification predictions and potentially acts on them thus influencing the market.

are then trained incrementally using minibatches of data. Once training has converged, the classifier makes predictions about unseen data which may influence the behavior of a trader, who then takes place in creating market events. To evaluate the performance of our models two different experimental setups are used, which are described below.

5.1.1. Anchored walk forward

In this case, the first $d = 1, \ldots 9$ day samples of all the stocks are progressively used to train our models and the sample belonging to the $(d + 1)$-th day is used for testing. Since our data consists of 10 days for each stock, this means we have a total of 9 different folds to run our models. This evaluation method is known as *anchored walk forward analysis*, as the starting point in time remains fixed [50]. Figure 3 illustrates the first three progressions of this evaluation method. By using this evaluation method, we can determine whether the models are able to learn features which capture temporal information, so as to be able to predict future mid price movements.

5.1.2. Hold-out per stock

For this setup, the models are trained on data describing 4 stocks and evaluated on the last unknown stock. The experiments are executed 5 times so that every stock is used as the unseen stock the models are evaluated on. This evaluation method is used to determine whether the models can learn features from stocks that can be applied to an unknown stock.

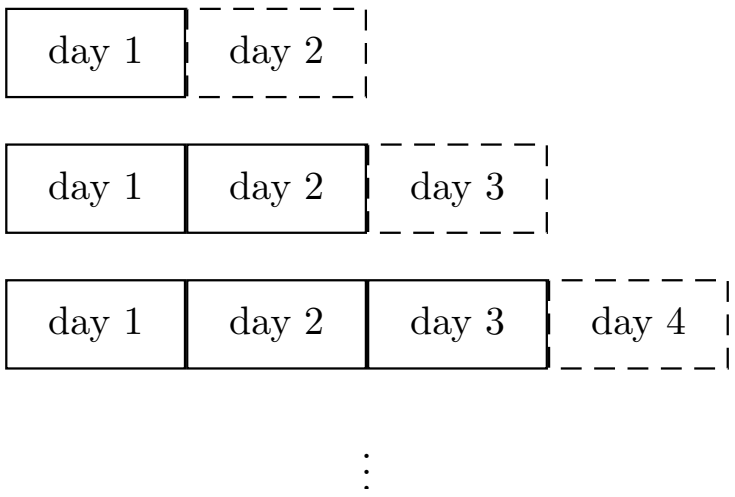

Figure 3: In the anchored walk forward evaluation method, the first $d = 1, \ldots 9$ days are used progressively as the training dataset and evaluation is performed on the $(d + 1)$-th day. The solid line is used to depict days used for the training dataset, whereas the dashed line indicates the day used for the evaluation. The first three progressions of this evaluation setup are shown.

5.2. Evaluation Results

Three metrics are used for the evaluation of the described models: *average precision per class* (macro precision), *average recall per class* (macro recall) and *average F-score* (macro F-score). The precision is defined as the ratio of the true positives over the sum of the true positives and the false positives, while the recall is the ratio of the true positives over the sum of the true positives and the false negatives. The F-score is defined as the harmonic mean of the precision and the recall. However, as the F-score is also macro-averaged over the three classes, its values might lie outside the range of the mean of the precision and recall.

Because of the class imbalance caused by considering only meaningful changes in the movement of the mid prices, a classifier biased towards the no-change class would achieve very high accuracy, as the majority of the predicted labels would match the ground truth. The accuracy metric becomes meaningless in such severely imbalanced problems and is omitted. For the macro-averaged metrics, the corresponding metric is first computed for each class and finally all three values — one for each class — are averaged. Thus, low results in the less represented classes will equally affect the final result.

For the AE representation, a 5-layer architecture of 144-72-24-72-144 neurons for each respective layer is used. The intermediate 24-dimensional representation $\mathbf{x}_{enc}$ is extracted and used as input to the evaluated classifiers. As for the BoF model, the fuzziness parameter g is set to 0.01 and K is equal to 128, thus producing 128-dimensional histograms to be used as representations of the input for the classification task.

The representations obtained by the sliding window over the handcrafted features in combination with the features extracted by the AE, and BoF models are used as the input to the described classifiers. The circled plus symbol is used to denote the concatenation

operation in Tables 1-6 which summarize the evaluation results. For example, *last*$\oplus$*BoF* denotes the concatenation of the *last* representation with the representation extracted by the BoF model. To evaluate the models under a wide range of conditions we have conducted extensive experiments using three different prediction horizons (N_a), i.e., for $N_\alpha = 1$, $N_\alpha = 5$ and $N_\alpha = 10$, and for both of the experimental setups discussed.

5.2.1. SVM Results

For the linear SVM classifier trained with SGD, the class imbalance accompanying the data is rectified by introducing weights associated with each class and adjusting them to be inversely proportional to the number of samples belonging to that class. This ensures that the less represented classes will be taken into account in the optimization process, making the classifier less biased towards the better represented class and thus more useful for practical applications.

Table 1 contains the results achieved for all three different sets of prediction targets for the anchored day evaluation setup, whereas Table 2 contains the results for stock hold-out setup. The N_α column contains the number of succeeding samples taken into consideration for the production of the prediction targets, as discussed in Section 4.2. The *Input* column contains the representation used as input to the classifier. The macro-averaged precision, recall and F-score are presented, as well as the standard deviation observed for these metrics over the progressive experiments for both evaluation setups.

For the anchored day setup, the results indicate that information from the past and the present can be used to generalize and make useful predictions about future movements of the stocks' mid prices. Taking into consideration all of the metrics, the results seem to improve when the prediction target is derived as the mean of next 5 and 10 samples, and slightly deteriorate when the prediction target is the movement of the next 1 sample. This means that the classifier is able to better capture the average movement of the mid price over a few succeeding time steps, which is expected as the movement of the mid price in the directly succeeding time step can be more noisy.

Moreover, for all prediction targets, as past values are taken into consideration, i.e., for all representations excluding *last* and *AE* but including their concatenations with the rest, the classifier's performance improves. Thus, information from not only the current time step but also from a few steps back seems to be important in the generalization of the classifier and the correctness of the predicted movements.

Reducing the dimensionality of the data, i.e., using the *AE*, or the *BoF* representation, improves the speed of the classification process but at the same time harms the prediction metrics. However, combining the learned features with the handcrafted features seems to improve the classification metrics. For example, combining the *last* or the *AE* features, that only contain information of the current time sample, with the *BoF* representation, that captures the temporal dynamics of the stock as expressed in the last 5 time steps, improves the performance over simply using the *last* features. This can provide useful insight for further developing deep learning techniques that will capture both the current tendency as well as the temporal progression of the time series data.

Furthermore, note that the performance of the SVM is further improved upon when the

dimensionality of the data increases. This is reflected by the deteriorated performance of the AE representation, where the input is only 24-dimensional, as well as by the improved performance of the concatenated representations, e.g., *last*$\oplus$*mean*, *concat*, $AE \oplus BoF$, where the dimensionality of the data is augmented by the concatenation operation. This behavior is to be expected when using a linear classifier, as data lying in high-dimensional spaces are more easily separated by linear hyperplanes.

As for the hold-out evaluation setup, the results demonstrate that information from other stocks can be utilized to make predictions for an unknown stock. Thus, the classifier captures the general trend of the stock-market by extracting information from a few stocks and is able to generalize to another stock, which hasn't contributed to its training process. As with the previous evaluation setup, the results seem to slightly improve when N_α is equal to 5 or 10, instead of 1, reaffirming the stipulation that the average movement of the mid price over the next few samples is easier to predict than its direct movement in the next 1 sample. However, the results for $N_a = 1$ are still significant and outperform a biased classifier. Note that a perfectly biased random classifier cannot consistently achieve a (macro-averaged) F-score higher than 33.33% on the considered 3-class problems.

Once again, due to the linearity of the classifier, the lower-dimensional representations

Table 1: Anchored walk forward evaluation: SVM

$\mathbf{N_\alpha}$	**Input**	**Precision**	**Recall**	**F-score**
1	last	45.72 ± 5.83	37.96 ± 2.99	37.51 ± 2.45
1	mean	48.14 ± 5.08	40.81 ± 2.85	41.53 ± 1.78
1	last⊕mean	$\mathbf{51.97 \pm 5.02}$	42.77 ± 3.39	43.95 ± 1.68
1	concat	49.31 ± 4.13	$\mathbf{44.65 \pm 2.32}$	$\mathbf{45.44 \pm 1.87}$
1	AE	40.17 ± 7.21	35.88 ± 2.35	34.32 ± 2.06
1	BoF	44.50 ± 3.45	37.10 ± 1.07	37.73 ± 1.21
1	AE⊕BoF	$\mathbf{45.98 \pm 3.97}$	39.57 ± 3.16	38.16 ± 2.99
1	last⊕BoF	45.32 ± 4.17	$\mathbf{40.32 \pm 3.63}$	$\mathbf{39.80 \pm 2.68}$
5	last	40.47 ± 1.88	41.00 ± 1.92	38.56 ± 1.72
5	mean	50.60 ± 3.29	46.22 ± 1.32	46.91 ± 1.72
5	last⊕mean	$\mathbf{52.14 \pm 2.56}$	50.30 ± 1.54	$\mathbf{50.68 \pm 1.39}$
5	concat	50.95 ± 3.83	$\mathbf{50.64 \pm 1.90}$	49.81 ± 2.51
5	AE	42.66 ± 4.04	38.49 ± 3.07	36.75 ± 2.58
5	BoF	44.85 ± 2.93	41.73 ± 1.14	42.26 ± 1.34
5	AE⊕BoF	$\mathbf{47.64 \pm 1.96}$	42.57 ± 1.57	$\mathbf{43.27 \pm 1.53}$
5	last⊕BoF	45.13 ± 3.03	$\mathbf{44.16 \pm 2.35}$	42.43 ± 1.78
10	last	42.10 ± 2.88	40.86 ± 1.80	40.48 ± 1.94
10	mean	48.03 ± 2.96	47.28 ± 2.48	46.62 ± 3.31
10	last⊕mean	$\mathbf{50.07 \pm 1.89}$	49.51 ± 1.60	$\mathbf{49.25 \pm 1.58}$
10	concat	49.74 ± 3.15	$\mathbf{49.86 \pm 1.47}$	49.20 ± 2.27
10	AE	39.83 ± 2.72	38.49 ± 2.31	36.01 ± 1.84
10	BoF	42.89 ± 1.43	42.01 ± 1.82	41.57 ± 1.31
10	AE⊕BoF	$\mathbf{46.18 \pm 2.05}$	42.36 ± 1.98	41.70 ± 2.45
10	last⊕BoF	44.35 ± 2.62	$\mathbf{44.27 \pm 1.91}$	$\mathbf{42.70 \pm 1.39}$

Table 2: Hold-out per stock evaluation: SVM

$\mathbf{N_\alpha}$	**Input**	**Precision**	**Recall**	**F-score**
1	last	41.41 ± 5.48	38.28 ± 3.12	31.03 ± 9.08
1	mean	46.24 ± 3.14	40.06 ± 3.36	40.02 ± 2.63
1	last⊕mean	48.04 ± 3.67	42.74 ± 4.53	$\mathbf{41.49 \pm 2.69}$
1	concat	$\mathbf{48.98 \pm 6.28}$	$\mathbf{44.19 \pm 7.81}$	38.85 ± 6.36
1	AE	37.05 ± 6.75	34.78 ± 1.70	27.98 ± 13.10
1	BoF	40.78 ± 3.94	$\mathbf{37.50 \pm 1.73}$	31.97 ± 10.18
1	AE⊕BoF	41.63 ± 2.37	36.64 ± 2.86	34.85 ± 2.74
1	last⊕BoF	$\mathbf{42.97 \pm 3.41}$	37.26 ± 2.18	$\mathbf{36.48 \pm 2.93}$
5	last	40.58 ± 2.65	39.45 ± 2.38	36.08 ± 5.50
5	mean	47.37 ± 2.61	46.11 ± 2.58	45.00 ± 2.12
5	last⊕mean	$\mathbf{49.19 \pm 3.48}$	49.18 ± 2.39	47.59 ± 1.72
5	concat	48.95 ± 2.64	$\mathbf{50.17 \pm 2.50}$	$\mathbf{48.58 \pm 1.57}$
5	AE	38.93 ± 1.43	36.22 ± 2.52	27.78 ± 11.62
5	BoF	44.33 ± 4.28	40.61 ± 3.89	37.93 ± 5.22
5	AE⊕BoF	43.02 ± 2.01	38.14 ± 2.21	37.77 ± 3.00
5	last⊕BoF	$\mathbf{44.76 \pm 2.86}$	$\mathbf{40.85 \pm 2.62}$	$\mathbf{40.20 \pm 2.80}$
10	last	43.32 ± 4.13	38.66 ± 1.96	37.66 ± 2.27
10	mean	46.01 ± 1.59	46.69 ± 3.84	45.70 ± 2.61
10	last⊕mean	$\mathbf{48.89 \pm 2.40}$	$\mathbf{47.91 \pm 2.40}$	$\mathbf{47.74 \pm 1.90}$
10	concat	47.71 ± 0.52	47.88 ± 4.46	45.14 ± 4.58
10	AE	42.78 ± 6.13	35.38 ± 1.17	30.91 ± 2.99
10	BoF	40.48 ± 1.44	38.51 ± 3.20	33.34 ± 9.69
10	AE⊕BoF	44.15 ± 2.57	37.38 ± 3.71	33.22 ± 3.82
10	last⊕BoF	$\mathbf{44.99 \pm 3.40}$	$\mathbf{40.27 \pm 3.11}$	$\mathbf{40.05 \pm 2.81}$

perform slightly more poorly than the higher-dimensional ones. The performance of the classifier is better when predicting the average movement of the mid price for the next 10 samples and when using higher-dimensional representations, such as the *concat* representation. The predictions made by the classifier reflect the general trend of the market rather than the individual tendencies of each stock. This is further corroborated by the high standard deviations in the F-score especially in the case where $N_\alpha = 1$, which indicate that information only from other stocks is insufficient when making predictions about an unseen stock.

5.2.2. SLFN Results

For the SLFN, the size of the hidden layer is set to 1000 and σ is set to the mean pairwise distance between the training feature vectors. The k-means clustering algorithm is used for the computation of the weights of the hidden layer, whose activations are RBFs which measure the similarity between the input and each of the 1000 prototype vectors learned. Similarly to the linear SVM, class weights are used to manage the imbalance between classes. The performance of this classifier is summarized in Table 3 for the anchored day setup and in Table 4 for the hold-out setup.

For the anchored day setup, the performance of the classifier is more or less consistent for all three prediction targets. This means that the classifier is able to make generalizations about the movement of the mid price in the immediately succeeding time step, just as well as for its average movement in the next 5 and 10 time steps. In comparison to the linear SVM classifier, the SLFN achieves somewhat better precision at the cost of slightly worse recall and F-score values. In other words, the predictions made by this model are more precise, but at the same time the model falsely classifies many positive samples as negative.

Although the model takes into consideration the class imbalance in the final classification task, it relies heavily on its first, unsupervised part which performs clustering on the input data without taking into account the distribution of the classes. Thus, the prototype vectors might be unfairly distributed over the classes, favoring one over the others, depending on the geometric distribution of the data in the input space. If an input sample is highly similar to one or more prototype vectors which the model has been trained to map to one of the classes, the model confidently classifies this sample as positive, i.e., as belonging to that class. Thus, such samples are more likely to be correctly classified, leading to more precise predictions. However, due to the unsupervised nature of the clustering step, an input sample is more likely to resemble prototype vectors mapped to multiple classes. Such samples lead the model to false negative predictions, accounting for the low recall scores. This phenomenon is even more severe when distribution shift and concept drift issues exist, as in the case of the hold-out evaluation setup (Table 4).

For the hold-out evaluation setup, the performance of the classifier is consistently inferior in comparison to the performance achieved by the SVM. This can be attributed to the fact that these representations are derived in an unsupervised fashion and used as the input to the also unsupervised clustering algorithm, which greatly affects the overall performance of the classifier. In combination with the fact that the hidden layer of the SLFN is not trainable, this constitutes the most major drawback of this classifier. Moreover, the F-score exhibits very high variance in all three prediction targets. This is indicative of the fact that the classifier fails to generalize and make correct predictions about an unseen stock, by using data only from other stocks. Thus, the SLFN fails to capture the general trend of the stock market and apply its knowledge to unknown data.

5.2.3. MLP Results

The architecture of the MLP includes an input layer with as many neurons as the dimension of the input representation, two successive hidden layers each with 512 neurons, and the output layer consisting of three neurons, corresponding to the three possible classes. The results for the anchored day and the hold-out evaluation setups are shown in Table 5 and Table 6 respectively. The results are notably better than the ones achieved by the previous classifiers. This can be attributed to the fact that MLPs are capable of capturing more complex non-linear relations between the data, are inherently more robust to noisy inputs and can better handle distribution shift phenomena.

For the anchored day setup, the best performance is achieved when the prediction target is derived as the average movement of the mid price of the next five samples, that is $N_\alpha = 5$ and when higher-dimensional representations are used as input, i.e., the combinations of

Table 3: Anchored walk forward evaluation: SLFN

$\mathbf{N}_\alpha$	**Input**	**Precision**	**Recall**	**F-score**
1	last	46.57 ± 2.47	41.19 ± 4.00	35.60 ± 6.13
1	mean	48.74 ± 3.74	42.12 ± 4.34	36.67 ± 11.06
1	last⊕mean	$\mathbf{50.46 \pm 3.59}$	$\mathbf{37.52 \pm 1.35}$	$\mathbf{38.78 \pm 2.04}$
1	concat	47.37 ± 4.51	35.44 ± 1.18	35.42 ± 1.83
1	AE	37.08 ± 8.70	$\mathbf{36.22 \pm 3.49}$	31.57 ± 4.16
1	BoF	34.20 ± 9.12	34.72 ± 3.00	33.02 ± 2.93
1	AE⊕BoF	36.91 ± 8.04	35.33 ± 3.72	$\mathbf{33.03 \pm 2.24}$
1	last⊕BoF	$\mathbf{41.54 \pm 7.87}$	35.71 ± 2.60	32.60 ± 7.97
5	last	48.80 ± 2.76	41.36 ± 3.80	34.74 ± 6.61
5	mean	51.14 ± 3.40	$\mathbf{46.57 \pm 3.61}$	$\mathbf{43.43 \pm 3.26}$
5	last⊕mean	50.71 ± 2.09	43.95 ± 4.45	40.97 ± 3.30
5	concat	$\mathbf{51.29 \pm 4.27}$	36.49 ± 2.03	34.88 ± 2.17
5	AE	38.26 ± 4.18	37.08 ± 3.60	31.80 ± 9.24
5	BoF	45.77 ± 4.35	41.39 ± 4.89	35.18 ± 10.51
5	AE⊕BoF	47.35 ± 4.84	37.45 ± 4.28	31.79 ± 3.96
5	last⊕BoF	$\mathbf{49.49 \pm 2.15}$	$\mathbf{41.66 \pm 5.69}$	$\mathbf{35.29 \pm 7.72}$
10	last	48.01 ± 3.81	41.24 ± 3.79	35.34 ± 7.08
10	mean	51.10 ± 2.81	$\mathbf{44.66 \pm 5.87}$	$\mathbf{38.56 \pm 11.27}$
10	last⊕mean	51.59 ± 2.69	40.88 ± 3.43	37.97 ± 9.87
10	concat	$\mathbf{51.77 \pm 3.71}$	39.78 ± 5.16	37.75 ± 6.19
10	AE	39.61 ± 12.32	37.74 ± 3.55	31.44 ± 5.08
10	BoF	47.41 ± 3.03	40.53 ± 3.89	35.05 ± 7.32
10	AE⊕BoF	47.26 ± 2.76	$\mathbf{41.75 \pm 3.53}$	$\mathbf{36.73 \pm 4.96}$
10	last⊕BoF	$\mathbf{48.46 \pm 3.70}$	41.25 ± 5.34	36.03 ± 7.47

the handcrafted features. The dimensionality of the input data seems to slightly affect the performance of the classifier, although even the low-dimensional representations derived by the AE achieve competitive results. The *concat* and *last⊕mean* representations, which exhibit the highest dimension amongst the representations, yield the best results. Moreover, the classifier achieves great performance even when trying to predict the mid price movement of the immediately succeeding 1 sample.

For the hold-out setup, the classifier seems capable of making correct predictions for all three sets of labels evaluated, i.e., for $N_\alpha = 1, 5$ and 10. In fact, the MLP achieves the best performance of all three classifiers in this setup, meaning that it is capable of making better generalizations about unknown stock data, by learning from other stocks. Also, as dimensionality increases so does the performance of the classifier.

Furthermore, to provide better insight on the performance of the best model (MLP) we report the evaluated metrics separately for each class using the best available representation (*concat*) for the $N_a = 10$ scenario, which a trader might be most interested in. The results are summarized in Table 7. The results, in particular the precision scores, indicate that the classifier is capable of making a correct decision for the up and down classes at a rate higher than random guessing. However note that the recall and, by extension, the F-scores

Table 4: Hold-out per stock evaluation: SLFN

N_α	Input	Precision	Recall	F-score
1	last	42.76 ± 3.99	38.25 ± 1.80	28.09 ± 13.06
1	mean	$\mathbf{46.11 \pm 6.52}$	$\mathbf{41.72 \pm 6.28}$	$\mathbf{31.52 \pm 13.97}$
1	last⊕mean	37.06 ± 16.73	40.53 ± 5.95	31.36 ± 13.04
1	concat	45.58 ± 6.84	39.10 ± 8.21	30.93 ± 6.34
1	AE	29.21 ± 14.48	34.58 ± 1.58	19.50 ± 12.03
1	BoF	$\mathbf{38.13 \pm 7.08}$	35.14 ± 1.72	28.10 ± 13.60
1	AE⊕BoF	36.31 ± 5.49	$\mathbf{36.76 \pm 4.36}$	$\mathbf{28.34 \pm 10.30}$
1	last⊕BoF	38.07 ± 5.62	36.40 ± 4.01	27.21 ± 11.19
5	last	44.54 ± 4.12	39.96 ± 3.77	31.62 ± 8.66
5	mean	$\mathbf{49.84 \pm 0.91}$	$\mathbf{42.63 \pm 2.58}$	$\mathbf{42.46 \pm 2.42}$
5	last⊕mean	47.01 ± 2.49	39.31 ± 5.28	34.48 ± 3.03
5	concat	44.52 ± 9.78	37.58 ± 5.03	28.38 ± 7.92
5	AE	34.82 ± 8.76	34.03 ± 1.88	18.91 ± 8.42
5	BoF	42.82 ± 3.34	$\mathbf{36.11 \pm 2.91}$	30.28 ± 4.13
5	AE⊕BoF	$\mathbf{43.26 \pm 3.64}$	35.26 ± 1.83	27.83 ± 10.01
5	last⊕BoF	42.15 ± 6.46	35.74 ± 1.63	$\mathbf{33.40 \pm 4.21}$
10	last	43.99 ± 4.85	38.51 ± 4.16	22.80 ± 11.72
10	mean	45.63 ± 3.29	39.48 ± 4.49	24.90 ± 13.95
10	last⊕mean	$\mathbf{46.08 \pm 2.73}$	$\mathbf{44.39 \pm 7.24}$	$\mathbf{32.98 \pm 11.56}$
10	concat	35.25 ± 15.57	40.71 ± 4.82	29.95 ± 10.68
10	AE	31.52 ± 9.10	34.60 ± 1.14	22.65 ± 7.20
10	BoF	42.10 ± 2.28	35.31 ± 2.24	25.05 ± 7.95
10	AE⊕BoF	43.34 ± 2.21	39.15 ± 1.62	33.14 ± 6.50
10	last⊕BoF	$\mathbf{44.83 \pm 3.29}$	$\mathbf{41.81 \pm 4.41}$	$\mathbf{34.12 \pm 3.72}$

are highly affected by the very large number of samples in our dataset and don't necessarily reflect the requirements a trader might expect from a classifier, as traders act on positive signals. The lower performance of the classifier for short-term forecasts can be also attributed to the noisy nature of the mid-price at very short prediction horizons — the model performs significantly better for predicting the longer term behavior of the stocks.

Finally, to examine the effect of the used MLP architecture on the quality of the learned model, we evaluated the MLP using a number of different architectures by varying the number of hidden layers and neurons per layer. The results are reported in Table 8. Only the evaluation results for the anchored walk forward setup using the *concat* representation are reported due to lack of space. However, similar results were obtained for the rest of the representations and evaluation setups. Even though the used "512-512-3" architecture leads to the best F-score, using a different architecture does not severely impact the quality of the learned model.

The activations of the hidden layers of the MLP can be thought of as features learned from the input representation, which are biased towards the task of classification. Thus, the MLP performs a kind of supervised feature extraction, where the extracted features are learned via optimizing the classification error. On the contrary, the AE and BoF mod-

Table 5: Anchored walk forward evaluation: MLP

N_α	Input	Precision	Recall	F-score
1	last	57.26 ± 7.44	37.40 ± 1.40	38.65 ± 1.50
1	mean	57.61 ± 6.67	40.94 ± 1.79	43.63 ± 2.25
1	last⊕mean	62.03 ± 8.27	43.50 ± 2.85	46.62 ± 2.89
1	concat	$\mathbf{62.30 \pm 6.90}$	$\mathbf{45.40 \pm 3.14}$	$\mathbf{48.75 \pm 3.22}$
1	AE	45.86 ± 6.37	33.47 ± 0.10	31.79 ± 0.46
1	BoF	51.53 ± 7.49	34.43 ± 0.54	33.77 ± 0.89
1	AE⊕BoF	54.91 ± 7.41	35.01 ± 0.74	34.81 ± 1.17
1	last⊕BoF	$\mathbf{56.26 \pm 7.50}$	$\mathbf{38.73 \pm 1.81}$	$\mathbf{40.40 \pm 1.47}$
5	last	55.24 ± 5.69	40.82 ± 2.42	41.98 ± 2.58
5	mean	58.51 ± 6.33	47.79 ± 3.14	49.98 ± 3.23
5	last⊕mean	62.67 ± 6.60	50.26 ± 3.26	53.06 ± 3.30
5	concat	$\mathbf{62.71 \pm 5.80}$	$\mathbf{53.72 \pm 3.63}$	$\mathbf{56.06 \pm 2.82}$
5	AE	49.41 ± 7.63	33.69 ± 0.15	30.11 ± 1.08
5	BoF	52.79 ± 5.86	36.47 ± 0.71	35.78 ± 1.29
5	AE⊕BoF	55.03 ± 5.36	38.05 ± 1.40	38.32 ± 2.22
5	last⊕BoF	$\mathbf{57.18 \pm 6.76}$	$\mathbf{43.79 \pm 2.64}$	$\mathbf{45.38 \pm 1.99}$
10	last	52.17 ± 4.57	41.91 ± 2.07	42.87 ± 2.24
10	mean	56.32 ± 5.89	45.06 ± 2.33	47.08 ± 2.52
10	last⊕mean	60.00 ± 5.96	48.19 ± 2.17	50.74 ± 2.43
10	concat	$\mathbf{60.61 \pm 6.20}$	$\mathbf{49.70 \pm 3.17}$	$\mathbf{52.03 \pm 3.12}$
10	AE	49.51 ± 11.82	33.69 ± 0.08	29.22 ± 1.23
10	BoF	51.41 ± 5.18	37.05 ± 1.09	36.17 ± 1.53
10	AE⊕BoF	53.63 ± 5.23	38.09 ± 1.36	37.90 ± 2.05
10	last⊕BoF	$\mathbf{54.22 \pm 5.89}$	$\mathbf{43.93 \pm 1.35}$	$\mathbf{45.54 \pm 1.51}$

els perform unsupervised feature extraction, which has no guarantee of being suitable for the task at hand. The classification-biased feature learning process that occurs in parallel with the classifier's training, has the potential to produce very robust features and lead to improved predictions. This is reflected by the slightly deteriorated performance of the classifier, when the input representation is derived by unsupervised feature extraction techniques. In combination with the supervised feature learning performed by training the MLP, this further reaffirms the assumption that the activations of the MLP's hidden layers learn representations of the data which highly befit the task of classification.

5.3. Computational Complexity Analysis

Financial exchanges generate a vast amount of data that must be processed in real-time in order to quickly respond to the volatile conditions of the markets. Therefore, the speed of the deployed models is equally important with the forecasting accuracy in real-word applications where large amounts of data must be processed under strict time constrains. In this Subsection we provide both an asymptotic forecasting time analysis and an empirical study of the run time of the used models. Our analysis is focused on the prediction complexity, since the real-time constrains can be relaxed during training by using a slightly outdated

Table 6: Hold-out per stock evaluation: MLP

$\mathbf{N}_\alpha$	Input	Precision	Recall	F-score
1	last	57.43 ± 2.22	35.61 ± 0.53	35.30 ± 1.98
1	mean	58.28 ± 5.42	39.32 ± 4.03	40.05 ± 4.27
1	last⊕mean	59.15 ± 5.84	42.38 ± 5.52	43.08 ± 5.07
1	concat	$\mathbf{60.30 \pm 4.96}$	$\mathbf{43.11 \pm 2.01}$	$\mathbf{46.05 \pm 2.55}$
1	AE	40.93 ± 5.73	33.50 ± 0.16	31.26 ± 1.54
1	BoF	45.52 ± 3.77	34.07 ± 0.58	32.31 ± 2.62
1	AE⊕BoF	48.93 ± 1.96	34.20 ± 0.52	32.62 ± 2.02
1	last⊕BoF	$\mathbf{53.19 \pm 7.01}$	$\mathbf{37.79 \pm 2.48}$	$\mathbf{38.37 \pm 3.90}$
5	last	54.88 ± 5.36	37.86 ± 2.28	36.96 ± 3.11
5	mean	60.08 ± 1.36	44.96 ± 2.21	47.03 ± 3.39
5	last⊕mean	$\mathbf{62.53 \pm 2.22}$	49.62 ± 2.93	$\mathbf{52.12 \pm 3.20}$
5	concat	61.29 ± 6.83	$\mathbf{52.48 \pm 5.88}$	51.90 ± 5.83
5	AE	40.32 ± 7.26	33.48 ± 0.11	28.82 ± 1.86
5	BoF	47.91 ± 3.43	35.63 ± 1.12	33.31 ± 3.42
5	AE⊕BoF	52.85 ± 1.41	36.25 ± 1.89	34.41 ± 4.00
5	last⊕BoF	$\mathbf{58.07 \pm 7.30}$	$\mathbf{39.31 \pm 4.36}$	$\mathbf{38.23 \pm 6.54}$
10	last	52.80 ± 4.79	39.34 ± 3.38	37.61 ± 5.99
10	mean	58.30 ± 2.23	45.31 ± 2.73	47.25 ± 2.84
10	last⊕mean	$\mathbf{62.44 \pm 2.59}$	44.11 ± 2.82	45.44 ± 2.71
10	concat	61.19 ± 2.71	$\mathbf{48.68 \pm 4.23}$	$\mathbf{50.36 \pm 5.47}$
10	AE	41.96 ± 6.15	33.45 ± 0.06	27.66 ± 2.31
10	BoF	48.57 ± 1.51	35.86 ± 1.89	32.87 ± 4.59
10	AE⊕BoF	51.22 ± 2.37	36.60 ± 2.27	34.10 ± 4.81
10	last⊕BoF	$\mathbf{53.61 \pm 5.59}$	$\mathbf{41.05 \pm 3.42}$	$\mathbf{40.60 \pm 4.85}$

Table 7: Per class scores achieved by the MLP classifier using the *concat* representation.

Class	Precision	Recall	F-score
↑	49.60 ± 9.68	32.40 ± 7.44	37.96 ± 4.52
—	79.35 ± 5.25	89.47 ± 6.91	93.98 ± 5.04
↓	52.90 ± 11.33	27.23 ± 8.68	34.14 ± 5.78

Table 8: Comparing different MLP architectures using the *concat* representation (anchored walk forward setup, $N_\alpha = 10$).

Hidden Layers	Precision	Recall	F-score
512	61.24 ± 6.29	43.69 ± 2.85	47.03 ± 3.25
1024	61.09 ± 6.18	43.51 ± 2.07	46.97 ± 1.96
512-512	$\mathbf{62.30 \pm 7.18}$	$\mathbf{45.40 \pm 3.27}$	$\mathbf{48.75 \pm 3.36}$
1024-1024-1024	61.99 ± 7.98	44.65 ± 4.92	46.98 ± 3.57

model. Furthermore, all the used models can be incrementally trained and, as a result, adapt to the available computational resources during the training.

Let N_r be the size of the used representation, N_C be the number of possible mid-price movements ($N_C = 3$) and N_h be the size of the hidden representation (used only for the SLFN and the MLP models). Table 9 summarizes the time complexity of the models. The SVM has significantly lower time requirements than the other two models. The complexity of the SLFN and MLP models can also be adjusted by altering the size of the hidden layers ($N_h = 1000$ for the SLFN and $N_h = 512$ for the MLP). Note that the used SVM operates in the primal space and therefore the time complexity does not depend on the number of selected support vectors.

The results of the empirical run time analysis, reported in Table 10, also confirm the previous findings. All the classifiers were implemented using GPU-accelerated libraries [45] and a mid-range GPU with 6GB of RAM was used for the conducted experiments. To ensure a fair comparison of the compared classifiers we report the time needed after the feature extraction step. Note that even the MLP classifier, which is the most computationally-intensive model, is able to process almost 7,000 transactions per second using a mid-range GPU. On the other hand, the SVM achieves the best trade-off between forecasting accuracy and run time, being capable of processing more than 14,000 transactions per second.

Table 9: Computational complexity analysis of the used classifiers

Model	**Time Complexity**
SVM	$N_r N_C$
SLFN	$N_r N_h + N_h N_C$
MLP	$N_r N_h + N_h^2 + N_h N_C$

Table 10: Run time analysis of the used classifiers. The mean time observed over 1000 runs measured in milliseconds is reported.

Representation	**SVM**	**SLFN**	**MLP**
AE	0.062 ms	0.090 ms	0.124 ms
BoF	0.063 ms	0.093 ms	0.127 ms
last	0.065 ms	0.093 ms	0.133 ms
mean	0.065 ms	0.093 ms	0.133 ms
AE⊕BoF	0.065 ms	0.093 ms	0.130 ms
last⊕BoF	0.065 ms	0.093 ms	0.138 ms
last⊕mean	0.065 ms	0.098 ms	0.136 ms
concat	0.068 ms	0.112 ms	0.144 ms

5.4. Statistical Analysis

To validate the significance of the obtained results we performed a series of statistical tests. First, the three different classifiers used for forecasting the mid price movements were

compared using the Friedman's test [51]. The null hypothesis was defined as: "*There is no statistical significant difference ($a = 0.1$) between the SVM, SLFN and MLP classifiers.*" To compare the classifiers the F-score for the different representations and predictions horizons was used. The null hypothesis was rejected ($p = 3.9 \times 10^{-26}$), meaning that at least one classifier is significantly better than the others. The Nemenyi post-hoc test [51] was then used to evaluate the differences between the classifiers, as shown in Figure 4. The SVM and MLP classifiers are significantly better than the SLFN classifier. Note that even though the MLP classifier performs better than the SVM, the differences between the MLP and the SVM are not statistically significant ($a = 0.1$).

We also compared the eight different representations used for the conducted experiments. The null hypothesis was defined as: *"There is no statistical significant difference ($a = 0.1$) between the eight used representations."* Again, the null hypothesis was strongly rejected using the Friedman's test ($p = 5.5 \times 10^{-35}$). To further compare the representations the Nemenyi post-hoc test was used as before. The results are shown in Figure 5. The *concat*, *last+mean* and *mean* representations are significantly better than most of the other used representations. Also, note that the dimensionality of a representation seems to be correlated with its predictive power. However, even though the *last* and *mean* representations have the same dimensionality, the *mean* representation leads to significantly better results. This is mainly due to the de-noising effect of the averaging process used for extracting the *mean* representation, effectively suppressing possible outliers.

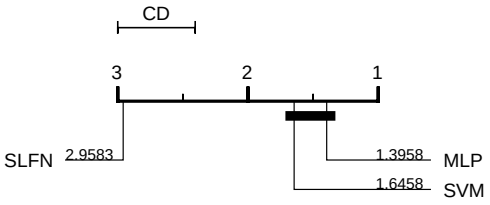

Figure 4: Nemenyi post-hoc test: Comparing different classifiers

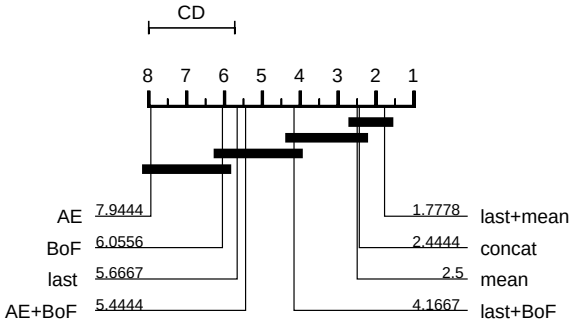

Figure 5: Nemenyi post-hoc test: Comparing different representations

5.5. Evaluation Summary

As corroborated by the statistical tests performed, the handcrafted representations and especially those that incorporate temporal information yield the most significant results. The *BoF* and *last*$\oplus$*BoF* representations follow, as they also take into consideration past values during the feature extraction process, but as the input dimension decreases so does the discriminative ability of the classifiers. The MLP classifier performs better when the input representation is derived by the handcrafted features, as its hidden layers extract a classification-based representation of its input. The low dimensionality of the *AE*, and *BoF* representations in combination with their unsupervised nature lead the MLP to make fewer correct predictions.

The low dimensionality of these learned representations allows for faster computations, albeit at the cost of achieving slightly deteriorated performances. The SVM classifier's performance increases with the dimensionality of the input representation, but falls behind the MLP's performance, as it is less resistant to the inherent noise of the data. Finally, the SLFN's performance is plagued by the high number of false negative predictions. The incorporation of past information does somewhat alleviate this drawback, but its overall performance falls back in comparison to the other classifiers.

6. Conclusions and Discussion

In this work, an extensive study into the information provided by the high frequency limit order book with respect to the forecasting of future mid price movements was presented. Several representations derived by handcrafted features as well as features learned by Machine Learning algorithms, ranging from 24 to 720-dimensional feature vectors, were considered and used as input to various classifiers for the forecasting task. Three scenarios were assessed regarding the span of time for which predictions are made. Finally, two evaluation analysis methods were examined for each classifier, scenario and input representation. Through the anchored walk forward setup, the ability of the evaluated classifiers to learn from past stock data and apply this knowledge to future, unknown data is determined. The hold-out setup serves to examine whether the classifiers are able to capture the general trends and movements of the stock market by learning from some stocks and applying this knowledge to unseen stocks.

The results achieved are remarkable in all cases, indicating that Machine Learning techniques are capable of correctly predicting mid price movements. Different classifiers seem to perform better in different aspects, such as the precision or recall of the predicted movements. We have provided the first, to the best of our knowledge, in depth review and evaluation that addresses the challenging characteristics of LOB data, such as the high velocity, variance, volume and strict real-time constraints, and uses ML techniques to predict the mid price movement, providing insight into the information contained in a LOB. The prediction results are improved when combining the extracted feature representations with the handcrafted ones, indicating that the feature extraction models are able to uncover latent, auxiliary knowledge. Finally, the learned representations also yield significant results

when used alone and systematically improve the time-wise performance of all classifiers by reducing the dimensionality of the input data.

References

[1] I. Kaastra, M. S. Boyd, Forecasting futures trading volume using neural networks, Journal of Futures Markets 15 (8) (1995) 953–970.

[2] I. Kaastra, M. Boyd, Designing a neural network for forecasting financial and economic time series, Neurocomputing 10 (3) (1996) 215–236.

[3] F. E. Tay, L. Cao, Application of support vector machines in financial time series forecasting, Omega 29 (4) (2001) 309–317.

[4] L.-J. Cao, F. E. H. Tay, Support vector machine with adaptive parameters in financial time series forecasting, IEEE Transactions on Neural Networks 14 (6) (2003) 1506–1518.

[5] C.-J. Lu, T.-S. Lee, C.-C. Chiu, Financial time series forecasting using independent component analysis and support vector regression, Decision Support Systems 47 (2) (2009) 115–125.

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

[7] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks., in: Aistats, Vol. 9, 2010, pp. 249–256.

[8] M. Längkvist, L. Karlsson, A. Loutfi, A review of unsupervised feature learning and deep learning for time-series modeling, Pattern Recognition Letters 42 (2014) 11–24.

[9] M. M. Rahman, M. M. Islam, K. Murase, X. Yao, Layered ensemble architecture for time series forecasting, IEEE transactions on cybernetics 46 (1) (2016) 270–283.

[10] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the International Conference on Machine Learning, 2008, pp. 1096–1103.

[11] M. G. Baydogan, G. Runger, E. Tuv, A bag-of-features framework to classify time series, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (11) (2013) 2796–2802.

[12] B. Park, J. K. Bae, Using machine learning algorithms for housing price prediction: The case of fairfax county, virginia housing data, Expert Systems with Applications 42 (6) (2015) 2928–2934.

[13] L. Yu, S. Wang, K. K. Lai, Forecasting crude oil price with an emd-based neural network ensemble learning paradigm, Energy Economics 30 (5) (2008) 2623–2635.

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

[15] R. Cont, Statistical modeling of high-frequency financial data, IEEE Signal Processing Magazine 28 (5) (2011) 16–25.

[16] R. F. Engle, J. R. Russell, Analysis of high frequency financial data, Handbook of Financial Econometrics, Y Ait-Sahalia (ed), forthcoming.

[17] J.-P. Bouchaud, M. Mézard, M. Potters, Statistical properties of stock order books: empirical results and models, Quantitative Finance 2 (4) (2002) 251–256.

[18] M.-W. Hsu, S. Lessmann, M.-C. Sung, T. Ma, J. E. Johnson, Bridging the divide in financial market forecasting: machine learners vs. financial economists, Expert Systems with Applications 61 (2016) 215–234.

[19] R. Khemchandani, A. Karpatne, S. Chandra, Generalized eigenvalue proximal support vector regressor, Expert Systems with Applications 38 (10) (2011) 13136–13142.

[20] R. Khemchandani, S. Chandra, et al., Regularized least squares fuzzy support vector regression for financial time series forecasting, Expert Systems with Applications 36 (1) (2009) 132–138.

[21] H. Mo, J. Wang, H. Niu, Exponent back propagation neural network forecasting for financial cross-correlation relationship, Expert Systems with applications 53 (2016) 106–116.

[22] W. Huang, Y. Nakamori, S.-Y. Wang, Forecasting stock market movement direction with support vector machine, Computers & Operations Research 32 (10) (2005) 2513–2522.
[23] K.-j. Kim, Financial time series forecasting using support vector machines, Neurocomputing 55 (1) (2003) 307–319.
[24] B.-L. Zhang, R. Coggins, M. A. Jabri, D. Dersch, B. Flower, Multiresolution forecasting for futures trading using wavelet decompositions, IEEE Transactions on Neural Networks 12 (4) (2001) 765–775.
[25] S. Walczak, An empirical analysis of data requirements for financial forecasting with neural networks, Journal of Management Information Systems 17 (4) (2001) 203–222.
[26] J. Heaton, N. Polson, J. Witte, Deep portfolio theory, arXiv preprint arXiv:1605.07230.
[27] M. Siikanen, J. Kanniainen, J. Valli, Limit order books and liquidity around scheduled and non-scheduled announcements: Empirical evidence from nasdaq nordic, Finance Research Letters to appear.
[28] M. Siikanen, J. Kanniainen, A. Luoma, What drives the sensitivity of limit order books to company announcement arrivals?, Economics Letters 159 (2017) 65–68.
[29] A. Ntakaris, M. Magris, J. Kanniainen, M. Gabbouj, A. Iosifidis, Benchmark dataset for mid-price prediction of limit order book data, arXiv preprint arXiv:1705.03233.
[30] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, 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) 42 (4) (2012) 463–484.
[31] R. Longadge, S. Dongre, Class imbalance problem in data mining review, arXiv preprint arXiv:1305.1707.
[32] G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[33] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, A. Y. Ng, Multimodal deep learning, in: Proceedings of the International Conference on Machine Learning, 2011, pp. 689–696.
[34] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, Cognitive modeling 5 (3) (1988) 1.
[35] T. Zhang, Solving large scale linear prediction problems using stochastic gradient descent algorithms, in: Proceedings of the International Conference on Machine Learning, 2004, p. 116.
[36] J. Sivic, A. Zisserman, et al., Video google: A text retrieval approach to object matching in videos., in: Proceedings of the International Conference on Computer Vision, Vol. 2, 2003, pp. 1470–1477.
[37] N. Passalis, A. Tefas, Information clustering using manifold-based optimization of the bag-of-features representation, IEEE Transactions on Cybernetics.
[38] G. Salton, J. Michael, Mcgill, Introduction to modern information retrieval (1983) 24–51.
[39] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A. Y. Wu, An efficient k-means clustering algorithm: Analysis and implementation, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (7) (2002) 881–892.
[40] N. Passalis, A. Tefas, Neural bag-of-features learning, Pattern Recognition 64 (2017) 277–294.
[41] O. Bousquet, L. Bottou, The tradeoffs of large scale learning, in: Proceedings of the Advances in Neural Information Processing Systems, 2008, pp. 161–168.
[42] S. C. Hoi, R. Jin, J. Zhu, M. R. Lyu, Semisupervised svm batch mode active learning with applications to image retrieval, ACM Transactions on Information Systems 27 (3) (2009) 16.
[43] T. Hofmann, B. Schölkopf, A. J. Smola, Kernel methods in machine learning, The annals of statistics (2008) 1171–1220.
[44] C. Robert, Machine learning, a probabilistic perspective (2014).
[45] K. Zhang, J. T. Kwok, B. Parvin, Prototype vector machine for large scale semi-supervised learning, in: Proceedings of the International Conference on Machine Learning, 2009, pp. 1233–1240.
[46] A. Iosifidis, A. Tefas, I. Pitas, Approximate kernel extreme learning machine for large scale data classification, Neurocomputing 219 (2017) 210 – 220.
[47] A. Iosifidis, A. Tefas, I. Pitas, Sparse extreme learning machine classifier exploiting intrinsic graphs, Pattern Recognition Letters 65 (2015) 192–196.
[48] S. S. Haykin, Neural networks and learning machines, Vol. 3, Pearson Upper Saddle River, USA, 2009.

[49] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.
[50] E. Tomasini, U. Jaekle, Trading Systems, Harriman House Limited, 2011.
[51] M. Hollander, D. A. Wolfe, E. Chicken, Nonparametric statistical methods, John Wiley & Sons, 2013.