UiT
THE ARCTIC UNIVERSITY OF NORWAY

Faculty of Science and Technology

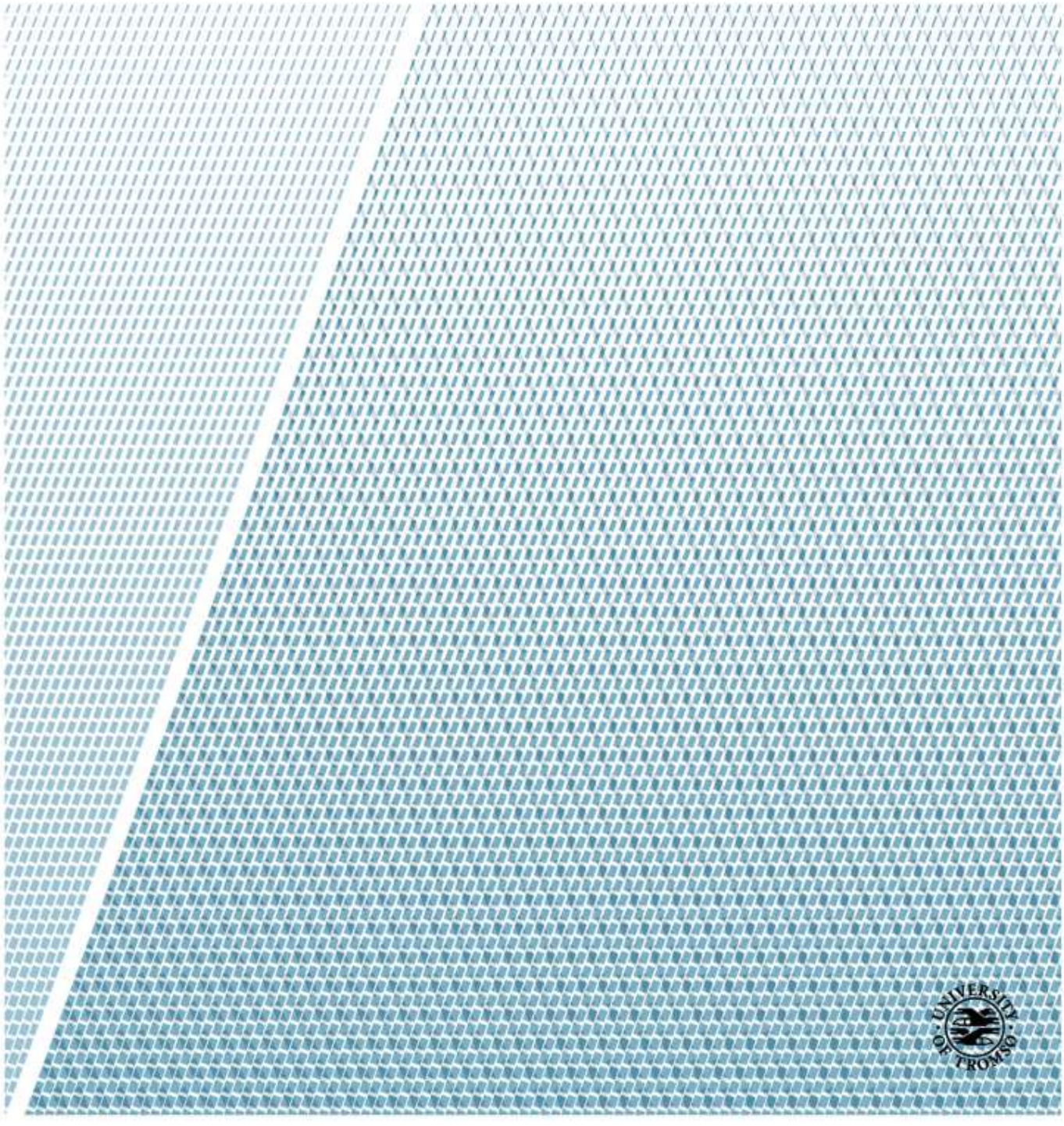
Department of Mathematics and Statistics

High frequency financial time series prediction: machine learning approach

—

Ekaterina Zankova

MAT-3900 Master Thesis in Mathematics - May 2016



Acknowledgement

First and foremost I would like to thank my supervisors: associate professors Martin Rypdal and Espen Sirnes for guidance and consultation throughout the whole work on this thesis. Special thanks to Espen Sirnes for providing the data for the research.

I would also like to express gratitude to my parents and to my husband for great support, inspiration and encouragement during all my Master program and work on this research.

Abstract

Purpose

The purpose of this research is to apply machine learning techniques for predicting high frequency financial time series. Experiments are conducted using several regressors which are evaluated with respect to prediction quality and computation cost. The obtained results are analysed in order to reveal parameter combination for particular regressor that yields the best results according to chosen performance criteria.

Motivation

Machine learning is a rapidly evolving subfield of computer science. It has enormous amount of applications. One of the application domains is financial data analysis. Machine learning was usually applied for analysis and forecasting of daily financial time series. Availability of high frequency financial data became another challenge with its own specifics and difficulties. Regressors, being a significant part of machine learning field, have been selected as study subjects for this project.

Methods

An extensive quantified literature search is conducted in order to gain insight on financial data analysis and prediction techniques with focus on machine learning approaches.

High frequency financial data from Oslo Stock Exchange is the main source of information to work with.

Open-source machine learning library Scikit-learn, designed especially for Python programming language, is used to perform all experiments on the given data set. A list of regressor implementations is adopted in order to perform regression analysis and make predictions for particular values of the data set.

Results

The possible solution for prediction of price fluctuations based on the sliding window approach is proposed in this paper.

The approach is tested in a series of experiments on four different regressors. The combination of parameters that yields the best results in terms of predefined performance criteria

are chosen as optimal for each regressor. A comparative analysis of the regressors' performance is conducted.

The test results suggest that short-term prediction (approximately 1 minute ahead) is more favourable for the given high frequency financial data.

Conclusion

All the tested regressors have demonstrated the best prediction quality on short periods of time.

The multilayer perceptron regressor has demonstrated the best results in terms of both error values and time expenses.

Possible improvements to prediction technique have been suggested.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

Financial time series are a very valuable source of information for stock market trading. Patterns hidden in the data may be used to predict the price fluctuations. That is why nature and behaviour of real-world financial time series are of particular interest and under continuous investigations for years.

Nowadays more and more data is available for analysis and inspection. Availability of high frequency data increased the opportunities for new discoveries in the scope of machine learning.

Statistical approaches and models traditionally applied for data analysis are no longer in trend. Rapid growth of computational power and development of new machine learning techniques make it possible to suggest new solutions for data analysis.

1.2 Goals

The main goal is to develop a solution that predicts price fluctuations using existing resgressor implementations.

The other goal is to evaluate the regressors' performance and reveal combinations of their parameters that provide the best results according to the predetermined performance criteria.

1.3 Report structure

The report structure is further described.

Chapter 2. Theoretical framework This chapter gives an overview about high frequency trading and financial data analysis. Machine learning algorithms applied within this research are described.

Chapter 3. Literature review This chapter provides scientific insight into development of financial data analysis and machine learning over time.

Chapter 4. Methods and materials This chapter provides specifications of tools that are used for implementation and testing. Data description is also presented in the chapter.

Chapter 5. Design of experiment This chapter provides description of the data preparation process and the testing procedure. Performance criteria for regressor evaluation are also discussed.

Chapter 6. Implementation This chapter describes the implementation procedure in detail.

Chapter 7. Experiment results This chapter describes the experiment results for each regressor that is tested. Comparative analysis of the examined reressors is conducted in this chapter and it is aimed to choose a regressor with a set of parameters that provides the best results in terms of performance criteria.

Chapter 8. Discussion This chapter discusses experiment findings and issues encountered while performing the experiments.

Chapter 9. Conclusions and future work This chapter summarizes the performed experiments and discusses the contribution of this research. Possibilities for potential improvements and future work are discussed in the end of the chapter.

Chapter 2

Theoretical framework

2.1 High frequency trading (HFT)

High-frequency trading is one of the most favoured forms of algorithmic trading (AT)[1] nowadays. Sufficient amount of literature is dedicated to high-frequency trading, though there is no uniform definition of HFT. However, it may be commonly described by characteristics [17, 34, 66] presented in the list below:

1. Extremely high execution speeds (order of milliseconds, seconds)
2. Generation of immense amount of orders and their subsequent rapid cancellation
3. Short-term holding periods – from seconds to couple of minutes
4. Overnight positions are rarely carried (trading day is preferred to be closed in neutral position)
5. Practice of co-location[2], individual data feeds in order to minimize the access time

High-frequency trading employs various strategies. The most practical of them are listed and described below [6, 28, 54]:

1. Market making – placing limit orders[3] in a specific way to earn money from bid-ask spread
2. Statistical arbitrage – getting profit from exploiting prices discrepancies and imbalance of their correlations

[1] Algorithmic trading (AT) - trading system that operates computer programs built on mathematical models and/or algorithms in order to carry out trading strategies. It is also known as algo-trading, automated or black-box trading. [52]

[2] Co-location (colocation) - locating trading appliances in physical vicinity of exchange centers. [66]

[3] Limit order – an order to buy/sell at a price that does not exceed/fall behind a particular/specialized/predetermined price

3. Spoofing – placing fake orders in order to influence prices and market

4. Quote stuffing – flooding the market via placing and removing great amount of orders in order to slow down competitors actions

The influence of high-frequency trading and its strategies on market quality is an ongoing research in financial industry. The following effects were discovered and investigated by various researchers: increased liquidity[4], narrowed bid-ask spreads[5], price discovery (formation) [28], improved market efficiency and diminished transaction expenses [6, 17, 66]. Some of the questions are still debatable such as whether HFT reduces [66] or increases [6] volatility[6] It is also claimed that bigger firms have more advantages as they may use co-location and other additional services.

High-frequency trading became of great use staring in 2000 with approximately 10% of all transactions. Significant growth by 164% was reached within 2005-2009. In 2010 HFT was accused of involvement in Flash Crash[7]. Other theories were discovered, but the true reason of this dramatic for financial industry event still remains unclear. But that did not affect its popularity and trading volumes continued to rise reaching roughly 70% in 2012.

2.2 Financial data analysis

Financial market is known to be intricate and complicated. In order to obtain the benefits, market participants must be aware of all possible information relevant to the market and its current situation. Continuous records of financial time series may be analysed to study market quality and behaviour and thus be of great use to traders and investors. Potential patterns hidden in the data may reveal early warning signals for such crucial events like flash crashes, anticipate volatility intervals, price movements and so much more.

In the scope of prediction concerning market behaviour the efficient market hypothesis (EMH) should be mentioned. It states that "all the available information is instantly processed when it reaches the market and it is immediately reflected in a new value of the assets traded" [54]. That is, there is no such approach that could "beat the market" and provide any profit from existing data. The EMH has three forms that vary by amount and type of available information: weak, semi-strong and strong. In this paper the weak form of the efficient market hypothesis is tested, using past prices of particular time period.

The efficient market hypothesis is a controversial question as both its supporters and opponents have empirical evidence of their respective correctness.

In spite of the EMH existence, financial data is a subject of numerous researches all over the world.

[4]Liquidity - a property of an asset to be converted into money without dramatic changes in its price. [14]

[5]Bid-ask spread - the difference between best ask and best bid prices. It is also known as bid-offer or buy-sell spread.

[6]Volatility - a measure of alteration within trading time series over a period of time. It is usually derived from standard deviation of asset returns. [14]

[7]Flash Crash (May 6th, 2010) - U.S. stock market crash characterized by rapid decline and further recovery of security prices within small time interval

Fundamental and technical analysis are traditional approaches for financial data analysis. The first one is based on investigation of company's characteristics, financial statements (e.g. assets, earnings, liabilities, etc.), factors that may affect its activity, history of its business performance in order to improve decision making, estimate risks and company value, etc. Aim of technical analysis is to predict price movements through examination of historical data and advise future actions for stakeholders. It relies on statistical properties of the data, technical indicators, analysis of chart patterns, etc.

2.3 Machine learning and data analysis

Machine learning is a subfield of artificial intelligence that supplies computers (machines) with capacity to learn from training sets of information and then performing on new data sets. It has numerous applications within the scope of data analysis, pattern and image recognition and classification.

Historical reference

Technological progress and new techniques of artificial intelligence make learning process faster and more efficient. Machine learning is practiced in everyday objectives and goals.

It emerged in the middle of the 20th century but began to prosper only around 1990s. It is still evolving and provides more and more techniques that may be applied in various fields of life.

2.3.1 Methods and algorithms

Artificial Neural Networks (ANNs)

Artificial neural networks are digital imitations of biological nervous system. This concept resembles the way human brain treats and processes information.

The architecture of NNs includes input, output and one or more hidden layers. Every layer is represented by several neurons (or nodes) that are linked to the neurons from preceding layers. Each link has a particular weight associated with it. All neurons have activation (or transfer) functions that define the output of every node for every layer by mapping the weighted inputs. Activation functions presented below are commonly applied, especially in feed-forward neural networks:

1. Logistic
$$f(x) = \frac{1}{1 + \exp^{-ax}},$$
where a is a slope parameter

2. Hyperbolic tangent
$$f(x) = \tanh(x)$$

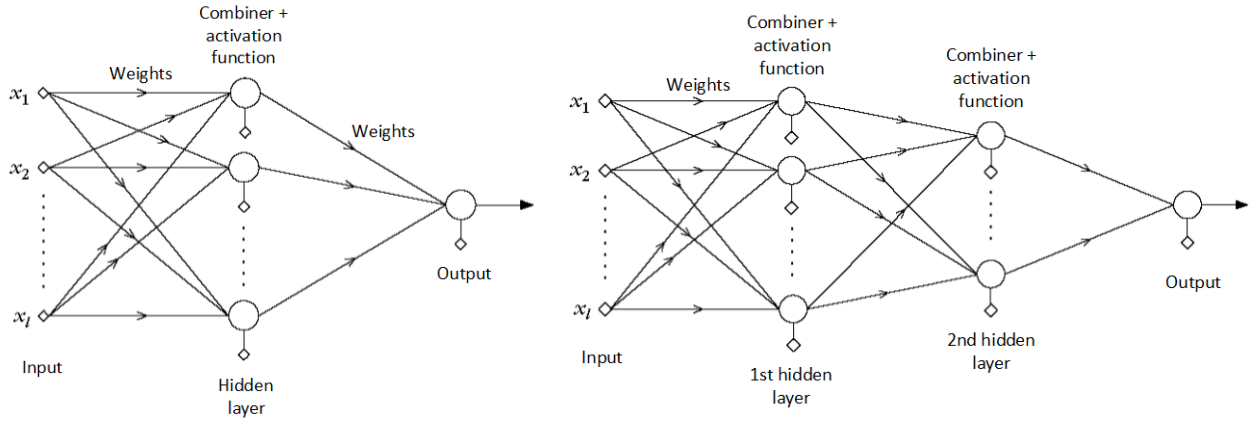
Due to the nature of digital representation of ANNs (usually represented as matrices of arbitrary sizes) it is possible to use parallel programming techniques to speed up the training and classification processes. This is commonly achieved by using multithreading techniques and technologies such as CUDA.

There is more information about artificial neural networks in chapter 3 Literature review.

Multilayer perceptron (MLP) is an example of a feed-forward neural network (figure[8] 2.1).

Its training procedure involves weights adjustment so the correct prediction can be made. Backpropagation is the most popular technique and is further described. Each processing element of the network is "punished" for mismatches which is implemented via backward propagation of the transfer function gradient through all the hidden layers to the very first.

All weights are updated to reduce cost function[9] value. Mean squared error between predicted and true outputs of the network is often chosen as a cost function.



(a) Two-layer perceptron (one hidden)

(b) Three-layer perceptron (two hidden)

Figure 2.1: Multilayer perceptron architecture

In backpropagation, the cost function is usually minimized using the gradient descent technique. It is based on following the direction along the negative gradient of the cost function at current point. However, there are some drawbacks:

1. Local minima of the cost function may be mistakenly taken for a desired global minimum and thus result in incorrect performance of the MLP
2. The computations involved require significant time expenses and may negatively affect convergence speed of the MLP

Performance quality of the MLP is also dependent on its internal architecture. Particular amounts of nodes and layers, type of activation functions and their parameters may significantly affect how well the MLP solves the given problem.

[8]Source: [80]

[9]Cost function - is a measure that specifies how distant current and optimal solutions are from each other.

On the other side, the multilayer perceptron is able to recognize nonlinear models and is capable of on-line (in real time) learning.

Support Vector Machines (SVM)

Support vector machines is an algorithm that searches for a hyperplane (or a set of those) that provides the largest margin[10] and separates the classes. A hyperplane with the largest margin is known as the optimal separating hyperplane (OSH) [29, 37].

In order to describe how the algorithm works, linearly separable two-class problem (figure[11] 2.2) is presented below. [80]

Let $\mathbf{x}_i$, $i = 1, 2, \ldots, N$ denote training feature vectors, which belong to either class w_1 or class w_2. The goal so far is to construct the following hyperplane

$$g(\mathbf{x}) = \mathbf{w}\mathbf{x} + b = 0, \tag{2.1}$$

where $\mathbf{x}$ is a column feature vector, $\mathbf{w}$ is a weight vector (is responsible for the hyperplane direction) and b is a threshold (specifies the exact position of the hyperplane in given space). Distance from the origin to the hyperplane (along the normal vector $\mathbf{w}$) is defined by $\frac{b}{||\mathbf{w}||}$.

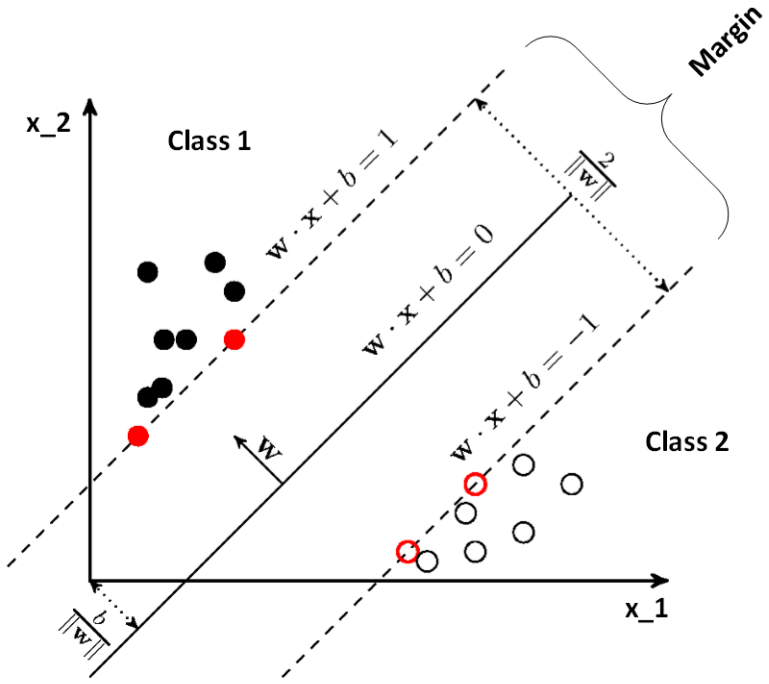


Figure 2.2: Support vector machines: linearly separable two-class problem

[10] Margin - a distance from a hyperplane to the closest training data point of a class. [29]

[11] Source: `http://mropengate.blogspot.no/2015/03/support-vector-machines-svm.html`

Vectors that directly lie on one of the hyperplanes

$$\mathbf{w}\mathbf{x} + b = \pm 1 \tag{2.2}$$

are the closest to the decision hyperplane and are called support vectors (displayed as red on the figure 2.2).

In order to guarantee the absence of the points inside the margin, that is $\frac{2}{||\mathbf{w}||}$ by definition, the following must be true

$$\begin{aligned} \mathbf{w}\mathbf{x} + b \geq 1, \quad & \forall \mathbf{x} \in w_1 \\ \mathbf{w}\mathbf{x} + b \leq -1, \quad & \forall \mathbf{x} \in w_2 \end{aligned} \tag{2.3}$$

The goal may now be posed as a nonlinear (quadratic) optimization problem

$$\min J(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2} \tag{2.4}$$

with linear inequality constraints derived from 2.3

$$y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \ldots, N, \tag{2.5}$$

where $y_i = 1$, $\forall \mathbf{x}_i \in w_1$ and $y_i = -1$, $\forall \mathbf{x}_i \in w_2$.

The solution of this problem is a weighted average of the training points [81]

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \tag{2.6}$$

subject to

$$\sum_{i=1}^{N} \lambda_i y_i = 0, \tag{2.7}$$

where λ_i are the Lagrange multipliers of optimization problem formulated above.

The nonlinear problem requires the following hyperplane [68]

$$g(\mathbf{x}) = \mathbf{w}\varphi(\mathbf{x}) + b = 0, \tag{2.8}$$

where $\varphi(\mathbf{x})$ is a kernel function that maps the original feature vector to a higher dimensional space, where the desired hyperplane may exist. This is also known as the "kernel trick". The most commonly used kernels are [1]:

1. Linear $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)$
2. Polynomial $K(\mathbf{x}_i, \mathbf{x}_j) = ((\mathbf{x}_i \cdot \mathbf{x}_j) + r)^d$
3. Radial basis function $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2)$
4. Sigmoid $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma(\mathbf{x}_i \cdot \mathbf{x}_j) + r)$,

where $(\mathbf{x}_i \cdot \mathbf{x}_j)$ denotes dot product of vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, $||\mathbf{x}_i - \mathbf{x}_j||^2$ is a squared Euclidean distance and parameter $\gamma > 0$.

Support vector machines have some advantageous properties [29, 80, 81] such as

1. Uniqueness of decision hyperplane, though same property for the Lagrange multipliers is not quaranteed

2. Resistance to the overfitting[12] problem

3. No local minima

4. Support of kernel trick - ensures cheaper computations of dot products in a feature space and improves SVM performance in spaces with higher dimensionality

On the other side, several drawbacks are present in the algorithm such as high computational and memory expenses, undefined selection procedure of kernel function and its parameters. Improper choice of kernel parameters may cause overfitting. [68, 80]

Support vector machines is also discussed in chapter 3 Literature review.

Support Vector Regression (SVR)

Support vector regression is a modification of the SVM suggested to deal with regression tasks. The difference between support vector classifier and regressor is in the decision function, which is binary in the first case and returns real values in the second, which allows to solve prediction problems.

The optimization problem for the nonlinear case of SVR is formulated as described below [73, 80, 85]:

$$\min J(\mathbf{w}, b, \varepsilon) = \frac{||\mathbf{w}||^2}{2} + C \sum_{i=1}^{N} \varepsilon_i \tag{2.9}$$

with the inequality constraints

$$\begin{aligned} y_i - (\mathbf{w} \cdot \mathbf{x}_i) - b \leq \varepsilon \\ (\mathbf{w} \cdot \mathbf{x}_i) + b - y_i \leq \varepsilon, \end{aligned} \tag{2.10}$$

where $\mathbf{x}_i$ is a training sample, y_i is a class indicator, N is amount of data samples, dot product $(\mathbf{w} \cdot \mathbf{x}_i) + b$ defines prediction for particular sample $\mathbf{x}_i$, tube parameter ε is a threshold whose range contains true predictions, $\varepsilon \geq 0$; C is a penalty parameter which is paid if data samples lie outside the tube, $C \geq 0$.

[12] Overfitting - adaptation of the algorithm to particularities of the training data; results in low generalization performance on a new data set. [80]

k Nearest Neighbours (*k*NN) Regression

The k nearest neighbours is a nonparametric method that has applications both in classification and regression problems.

Prediction is based on the true (target) outputs of k closest neighbours of particular training data point. Neighbourhood is determined by Euclidean distance between the given point and the rest of the points within the training set. The prediction is an average value of the k outputs with the smallest distances to the certain point and is shown below

$$\hat{y}_i = \frac{1}{k} \sum_{n=1}^{k} y_{n(i)}, \tag{2.11}$$

where $\hat{y}_i$ is the predicted output for the training data point x_i; $y_{n(i)}$ is the true output of nth nearest neighbour and k is the amount of neighbours.

Parameter k can be defined heuristically or via cross-validation technique (for more information see section 5.2 Performance criteria).

Generally, greater k implies smoother fit and smaller variance, but results in higher bias. The inverse logic is true for small k. [7, 74]

kNN is capable of solving multi-output tasks. [4]

Decision Tree (DT) Regression

Decision tree is another nonparametric method applied for regression tasks. It implements a hierarchical tree structure for constructing a model for prediction of a target variable "by learning simple decision rules inferred from the data features" [2].

Basic components are listed and described below:

1. Non-leaf nodes - depict tests on features
2. Branches – represent values for tested features
3. Leaf nodes (leaves) – hold predicted outcome value

The tree is constructed based on the training set. The topmost node is the root node. Variable that leads to the greatest drop of the MSE[13] is chosen with its split threshold. Splitting procedure is continuously repeated until the MSE reaches the specified threshold. [7]

Deeper trees have more sophisticated decision rules but provide better model fitness. [7]

Some of the advantages of decision trees are stated below [2]:

1. Only little data preprocessing is required
2. Logarithmic prediction cost (in the amount of used data points)
3. The ability to solve multi-output problems

[13]MSE – mean squared error (for more information see section 5.2 Performance criteria).

4. The ability to perform on both categorical and numerical data

On the other side, DT may produce biased trees, if there are dominating classes in the data set, and suffer from overfitting.

2.3.2 Summary

This chapter introduced the reader to the concept of high frequency trading, its strategies and potential effects on market. Importance of financial data analysis was discussed alongside with the methods commonly applied for this purpose. Special attention was paid to machine learning techniques such as artificial neural networks, support vector machines and regression, decision trees and k nearest neighbour regressions. The principles behind those techniques, their advantages and drawbacks were discussed in details.

Chapter 3

Literature review

3.1 Literature sources and search criteria

The following digital libraries have been used as sources of information: Google Scholar[1], ACM[2] (Association for Computing Machinery), IEEE[3] (Institute of Electrical and Electronics Engineers) Xplore, ScienceDirect[4], SSRN[5] (Social Science Research Network).

Several criteria were applied in order to find the most relevant articles and scientific papers. Only papers written in English were considered. Furthermore, only the articles published since 2000 were chosen.

3.2 Search methods and results

Primarily, all the sources were searched for articles that match the keywords and search criteria mentioned above. The number of "Found" papers stands for the quantity of articles that satisfied these conditions. Then, the papers were sorted depending on their relevance to the scope of this research. This number is denoted as "Relevant".

It should be mentioned that some articles can be found within several digital libraries at once.

Results of the search are presented in the Table 3.1 below.

Some articles are not dedicated to analysis of financial data but to analysis of other information, such as medical records [11, 21], information about water discharges [51], electricity prices and level of consumption/demand [25, 47, 60], amount of incoming requests to a particular server, sales analysis [8] and so on. Some of these articles were not removed from bibliography because they also describe relevant algorithms and methods for working with time-series.

[1] `https://scholar.google.com`
[2] `http://dl.acm.org`
[3] `http://ieeexplore.ieee.org`
[4] `http://www.sciencedirect.com`
[5] `http://papers.ssrn.com`

Keywords	Source	Found	Relevant
financial + data + analysis	Google Scholar	28	1
	ACM	37	4
	IEEE	6	0
high + frequency + + financial + data	Google Scholar	62	6
	ACM	12	1
	IEEE	2	0
high + frequency + trading	Google Scholar	38	16
	ACM	3	2
	IEEE	2	**1**
machine + learning + + financial + time + series	Google Scholar	186	24
	ACM	43	0
	IEEE	36	3
	ScienceDirect	6	2
	SSRN	1	1
machine + learning + + time + series + forecasting	Google Scholar	27	6
	ACM	35	0
	IEEE	19	2
	ScienceDirect	58	7
patterns + time + series	Google Scholar	14	2
	ACM	27	6
	IEEE	5	1
	ScienceDirect	18	4
time + series + prediction	Google Scholar	1	0
	ACM	36	12
	IEEE	4	1
Total		706	102

Table 3.1: Literature search results

3.3 Related work

The following subsections present the findings about the fields relevant to time series analysis with focus on financial data analysis and machine learning.

3.3.1 High-frequency trading (HFT)

Great interest to the high frequency trading appeared after the Flash Crash in May, 2010, when a sharp drop in security prices took place within very short interval of time. The mention of this significant for the financial world event is usually present in most of the papers that provide theoretical background of electronic and high frequency trading. Some researches are dedicated to examination of the Flash Crash itself [31,33], its prerequisite and

causes [34], and consequences for market [42]. These studies provoke relevant investigations on whether it is appropriate to use such trading strategies. This issue is discussed by Gomber et al. [34] who propose HFT supervision in order to reach trustful and healthy market environment.

A number of articles analyze the impact of high frequency trading on market quality which involves such concepts as liquidity, volatility and price development [13, 17, 18, 34, 66].

The topic remains extremely relevant up to the present day because the absolute majority of modern traders employ various electronic trading strategies including HFT.

3.3.2 Financial data analysis

Historical reference

The financial market is a complex system. When computers became capable of recording and storing tremendous amounts of data, the idea of its analysis appeared. Patterns and regularities that may possibly hide within data, may provide particular profit for market participants such as traders and investors, positively affect precision of investment decisions [44]. Nowadays a lot of effort is put into researches that are dedicated to prediction or estimation of price direction and turning points [29, 37, 39, 45, 89], motifs and patterns detection [59,62,63,75], volatility measuring, forecasting [15,55,56] and modeling [83], bankruptcy disclosure, presence of anomalies in market behavior, etc.

Financial data may have various formats and origins depending on the financial asset under consideration and the organization which assembles the data.

Thus, the methods of analysis may also vary depending on the subject data and the aims of research.

Methods of analysis

Financial market analysis splits into two totally opposite paradigms: technical and fundamental.

Fundamental analysis examines economic factors (e.g. inflation, fluctuations in currency exchange rate, economic growth/recession, etc. [29, 54]) that impact the market "in order to determine the intrinsic value of the market" [38].

It should be mentioned that no detailed information was detected concerning fundamental analysis or its techniques in the scientific paper that were denoted as relevant to the current research work.

The goal of technical analysis is to predict possible market behavior, such as price direction, in the future (short or long term) based on past-periods price values and trading volumes.

Technical analysis employs statistical and mathematical tools called technical indicators. The latter include: on-balance volume (OBV), average directional index (ADX), Aroon indicator, moving average convergence divergence (MACD), relative strength index (RSI),

stochastic oscillator and much more. These indices are sometimes used as inputs to Support Vector Machines [23, 43, 53].

Both technical and fundamental analysis conflict with Efficient Market Hypothesis (EMH) which states that market behavior is random and cannot be predicted. For more details about the EMH see chapter 2 Theoretical framework.

3.3.3 Machine learning

Machine learning offers a vast amount of approaches in solving data classification, pattern recognition and forecasting problems. The following algorithms have been tested on time series data during the past 15 years.

Artificial neural networks (ANNs or NNs) are usually referred to as the most applicable machine learning techniques for time series prediction.

Several research projects point to the fact that quality of ANNs performance depends on

1. Network structure [84, 90]
   (a) Number of hidden layers [27, 39, 70]
   (b) Connections between input and output layers (feedback, feedforward or direct) [27]
   (c) Quality, quantity and accuracy of parameters estimation (learning rate, weights, activation function, etc.) [39, 70, 89, 90]
2. Quality of input data [7, 84, 87, 90]
   (a) Data preprocessing (normalization, log transformation, removing outliers, trends and seasonality if any of these take place)
   (b) Size of data set [89]
3. Input variables [39, 44, 84, 90]

Each method has its strong sides and shortcomings which affect the researcher's decision whether or not to use the particular approach.

According to Tay and Cao [20,77–79] no prior assumptions about the data are required in order for the ANNs to be universal function approximators. Also the model misspecification problem affects ANNs to a smaller degree.

Neural networks are reported to be tolerant to data errors [90].

However, ANNs also possess a number of meaningful disadvantages such as

1. Existence of local minima [25,26,70,89]. There is a chance that network's training algorithm may reach local minima instead of global. There may also be various possibilities of local minima that will imply different results even for the same set of parameters.
2. Low learning rate precision [70, 90]. There is no deterministic technique to define parameters of the network that will provide the best performance.

3. Overfitting [39, 70, 78].

4. Computational cost [25, 26, 70]. Learning algorithms based on the gradient descent method (e.g. back propagation) may cause slow convergence during the network training.

5. Potentially poorer performance due to noise in the data [7, 20, 39, 40, 77–79], highly multidimensional data and data in large amounts [7, 39, 40].

Yu et al. [89] calls the neural networks "unstable learning methods" because even little variations in the training set and/or parameters may result in significant alterations in prediction accuracy.

Neural networks are currently represented in research papers in many various forms. In some cases they are still used in their pure original form proposed by Rosenblatt [67] and depending on the target problem yield good results. Modified and hybrid variations are used for more complicated tasks. The goal is generally to solve the existing classification problem and reach the best possible performance.

The short overview of artificial neural networks variations for time series forecasting that were recently tested, compared or discussed by researches is presented further (for more information on the methods see chapter 2 Theoretical framework) [87, 90]:

1. Multilayer perceptron (MLP) [7, 87] (for more information see chapter 2 Theoretical framework).

2. Recurrent Neural Network (RNN) overcomes performance of back propagation algorithms due to internal memory that identifies temporal dependencies in the data [45]. Unique solution is not guaranteed as initial weights are randomly set [82].

3. Radial basis function (RBF) networks [7, 20]

4. Bayesian NN [7, 87]

5. Neuron-fuzzy networks [48]

6. Generalized Regression Neural Network (GRNN) [7]

7. kNN (k Nearest Neighbor) regression [7, 74]

8. Regression tree [7]

9. Extreme Learning Machine (ELM) [70] surpasses issues induced by methods based on gradient descent [25, 70], produces high forecasting precision [70] and requires less time for training than networks with back propagation [25, 36].

Some of the methods listed above are also progenitors to further modified hybrid techniques.

Support Vector Machines (SVMs) are often compared with neural networks and usually surpass them in some ways. SVMs also display overfitting problem but due to Structural Risk Minimization[6] (SRM) principle [12, 23, 37, 39, 51, 79] show more resistance than ANNs who implement Empirical Risk Minimization[7] (ERM) [20, 39]. Another issue that SVMs share with NNs is estimation of parameters [20, 51, 68].

Unlike algorithms trained by back propagation, SVMs are more efficient in terms of speed [78]. However, large amounts of data may require additional time for training procedure [82].

SVMs possess the following positive properties:

1. Guaranty of unique solution [20]
2. Good generalization performance [12, 19, 20, 35, 37] even in complex cases [82]
3. Curse of dimensionality is excelled [35]

In order to improve prediction accuracy and eliminate some of the existing shortcomings SVMs undergo certain changes.

Support Vector Regression (SVR) was proposed to deal with regression problems [7, 53, 68, 88]. For more details about SVR see chapter 2 Theoretical framework.

For non-stationary time series Dynamic SVM [19] and C-ascending SVM [77] were proposed. Both employ less support vectors than the regular SVM and acquire increased generalization performance. Adaptive SVM was proven to possess the same properties [20] in order to handle alterations in structure of financial data.

Nonlinearity of financial data inspired researchers for the Wavelet Kernel SVM [35]. Experimental outcomes display improvements regarding forecasting precision and generalization performance.

3.3.4 Other techniques for time series analysis

Artificial intelligence approach offers other techniques for data analysis. The following methods are also gaining popularity in analysis of financial time series

1. Genetic Algorithm (GA) [29, 40, 45]
2. Self-organizing Maps (SOM) [79]
3. Fuzzy Inference System (FIS) [47]
4. ANFIS (Adaptive Network-based FIS) [47, 48]

Before machine learning became popular, the following statistical methods for time series modelling and analysis were in use (the list also includes combined forms):

1. Auto Regressive Moving Average (ARMA) [47, 49, 90]

[6] Principle that focuses on minimizing the generalization error upper bound

[7] Principle that attempts to minimize the amount of missclassification-related errors

2. Auto Regressive Integrated Moving Average (ARIMA or Box-Jenkins ARMA) [27, 32, 70, 82, 89–91]

3. SARIMA (Seasonal ARIMA) [70]

4. Generalized Auto Regressive Conditional Heteroskedastic (GARCH) [47]

5. Random Walk (RW) [37]

6. Lest Squares Estimation (LSE or LS) [47]

Several researchers indicate that combination of methods from different research fields may improve total performance and precision and may work better for particular problems [10, 27, 76, 79, 87, 89, 91]. The following hybrid approaches were proposed and tested

1. SARIMA + SVR [70]

2. ICA (Independent Component Analysis) + SVR [53]

3. SOM + SVM [79]

4. ARIMA + ANN [91]

5. SARIMA + ANN [70]

6. FIS + LSE [47]

7. SOM + SVM [79]

8. LS + SVM [12, 74, 83, 86]

9. ARMA + GRNN (Generalized RNN) [49]

For more information about methods and techniques mentioned in this subsection see chapter 2 Theoretical framework.

3.4 State of the Art

This section aims to represent the most recent (within past 5 years) news about researches and achievements in the field of machine learning techniques for financial data analysis and prediction.

Over the years Internet and mobile technologies were gaining popularity and are now a significant part of modern people's lives. Taking that into consideration, researchers proposed forecasting market behavior based on queries in search engines (e.g. Google) [30, 58, 64] and digital libraries (e.g. Wikipedia) [58]. Other studies are dedicated to sentiment analysis that involves investigation of news [50, 72], posts in social networks and blogs (e.g. Facebook,

Twitter) [9, 16, 24, 65, 69, 71, 92]. The majority of researchers agree that the connection between Internet activity and stock market is present.

The development of computing capabilities further enables studies of applying parallel programming on Graphics Processing Units (GPU) for financial market analysis that results in better performance with decreased computation cost and are energy efficient [22, 46, 61].

3.5 Discussion

Despite the fact that there is a significant amount of guidelines for working with different data formats in context of machine learning, designing a successful analysis/prediction technique appears to be heuristic to a large degree and involves a lot of experimentation. Optimal parameters for given machine learning approach are often obtained through a series of tests.

It might be possible to draw inspiration from other fields where machine learning has been applied with a great degree of success while designing new hybrid approaches for financial data analysis (e.g. image processing, voice recognition).

It should be mentioned that some factors (political, social, environmental, etc.) that affect market prices are not taken into consideration in many of the reviewed papers.

3.6 Conclusion

Conducting a quantified comprehensive comparison and distinguishing the best approach for financial time series analysis is a rather complicated task due to a vast variety of input data, different error estimation methods and problems being solved across different research projects. However, artificial neural networks proved to be the most prevailing technique in the prediction of the market behaviour.

Many factors may influence the results of prognosis. Some of them are linked to quality and structure of input data. In case of financial market even small imprecision in forecast is crucial and may result in significant financial losses. Thus, the phase of data preprocessing should not be ignored.

Numerous tests display that modified methods are sometimes more efficient and precise. Specific problems may require more complex methods such as designing hybrid algorithms.

Computing capabilities are evolving with great speed inspiring to produce more accurate and robust techniques for data preprocessing and its further analysis.

It should be noted that traditional methods of time series analysis are still in use. In time they were modified and coupled with machine learning techniques which resulted in improvement of prediction quality.

Artificial intelligence techniques were also evolving and developing new features within the last 15 years.

3.7 Summary

This chapter presented an overview of accomplishments in the scope of machine learning and artificial intelligence approaches applied for data analysis, in particular for financial time series. A quantified literature search has been conducted in order to gain knowledge on relevant topics.

Existing academic knowledge of high-frequency trading was briefly described. Further, the concept of financial market analysis was depicted, followed by the description of its main paradigms. Significant portion of the chapter was dedicated to the overview of the methods for financial time series prediction that were applied and implemented within the last 15 years. Drawbacks and advantages of the algorithms, results of experiments were discussed afterwards.

Chapter 4

Methods and materials

4.1 Tools

Programming language Python[1] (v.3.4.4) is chosen as a programming language due to its simplicity, multifunctionality and applicability for artificial intelligence related task.

Integrated development environment (IDE) The choice of IDE[2] falls on Eclipse[3] (v.23.0.2). It is extended with PyDev[4] (v.4.4.0) and Anaconda[5] (v.2.3.0, 64-bit) modules designed especially for Python language.

Machine learning and supporting libraries Python is extended with the list of scientific libraries (all of them as a part of the Anaconda scientific package) described below:

1. NumPy[6] (v.1.10.4) - scientific multidimensional (e.g. arrays, matrices) computations and high-level mathematical functions
2. SciPy[7] (v.0.17.0) - technical and scientific computations
3. Scikit-learn[8] (v.0.17.1) - machine learning algorithms implementations, support of data analysis
4. Matplotlib[9] (v.1.5.1) - 2D plotting

[1] `https://www.python.org/`

[2] Integrated development environment (IDE) – is a software application supplied with a set of instruments (such as source code editor, graphical debugger, build automation tools) for software development. Main purpose is to increase programmer's productivity.

[3] `https://eclipse.org/`

[4] `http://www.pydev.org/`

[5] `https://www.continuum.io/`

[6] `http://www.numpy.org/`

[7] `https://www.scipy.org/`

[8] `http://scikit-learn.org/stable/`

[9] `http://matplotlib.org/`

5. Plotly[10] (v.1.9.9) - on-line data visualization and analytics

4.2 Materials

4.2.1 Data description

High frequency financial time series were provided by associate professor Espen Sirnes, University of Arctic, Tromsø, Norway.

The whole data set covers three-year interval (2006-2008) of trading activity on the Oslo Stock Exchange (OSE) of three different companies: Birdstep Technology (BIRD), REC Silicon (REC) and Statoil (STL).

Each year is presented in separate `.csv` file that contains the following information for each company and year:

Column 1 Date/time (according to Microsoft Office Excel time convention, zero is 01.01.1900)

Column 2-6 The best ask prices[11] (column 2 is the best, column 3 the second best, etc.)

Column 7-11 The best bid prices[12] (column 7 is the best, column 8 the second best, etc.)

Column 12-16 Volumes[13] corresponding to the best ask prices (column 12 is the best, column 13 the second best etc.)

Column 17-21 Volumes corresponding to the best ask prices (column 17 is the best, column 18 the second best etc.)

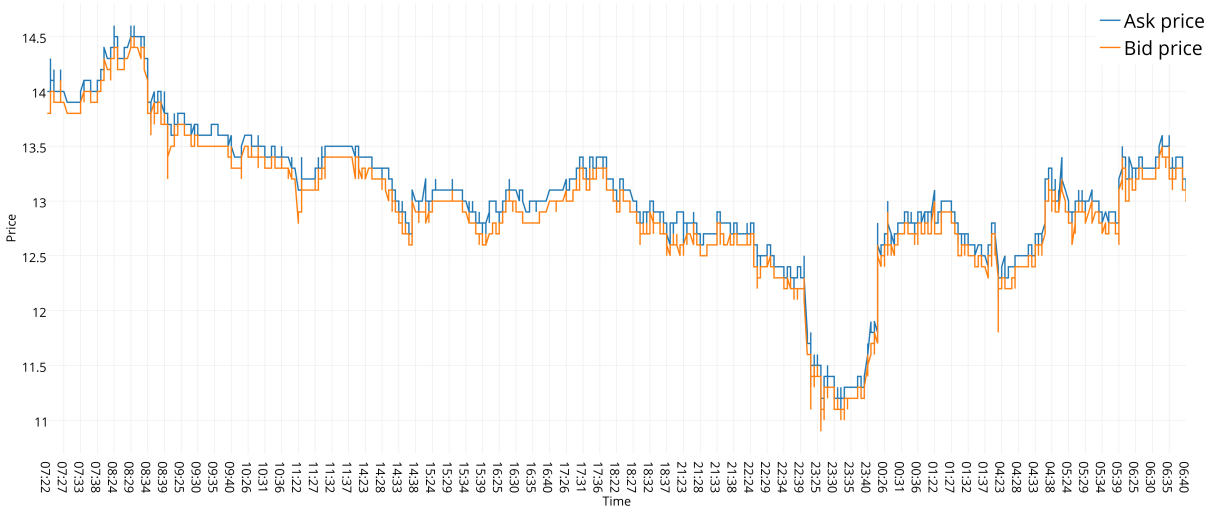
Data within one day interval is presented in figure 4.1

4.2.2 Regressor implementations

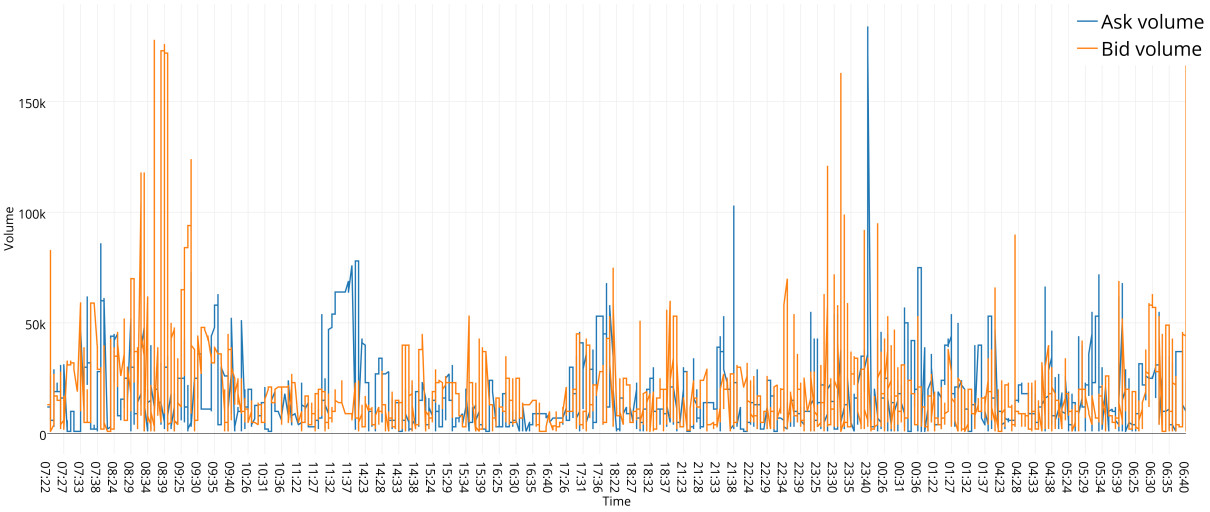
The following regressor implementations from Scikit-learn have been used:

1. Decision tree regressor
2. k Nearest neighbours regressor
3. Multilayer perceptron regressor
4. Support vector regressor

[10] `https://plot.ly/`
[11] Best ask price - the lowest price that a seller will accept for a security. [14]
[12] Best bid price - the highest price that a buyer will pay for a security. [14]
[13] Volume - the number of traded stocks. [14]



(a) Ask and bid prices



(b) Ask and bid volumes

Figure 4.1: One day of high frequency data from Bird 2006

4.3 Summary

This chapter presented the description of tools and materials that were used during this research. The origin of the data was described. One day of high frequency data was depicted.

Chapter 5

Design of experiment

5.1 Data preparation

All the experiments are held on one-year data for Birdstep Technology (file `Bird_2006.csv`). Some experiments are conducted on `Bird_2007.csv` and `Bird_2008.csv`. For detailed data description please see chapter 4 Methods and materials.

5.1.1 Data preprocessing

None of the data preprocessing techniques is applied. This decision is based on the idea of on-line application of regressors. If a regressor is applied in real time, it is not preferable and not always possible to preprocess the data due to the time cost which can make great difference in the scope of high-frequency trading.

Moreover, some preprocessing techniques (e.g. scaling, normalization) require certain knowledge (e.g. minimum and maximum values) about samples within data set in order to perform the procedure. Such techniques may not be applied within on-line learning procedures.

5.1.2 Data extraction

Only the information that will be used in testing is extracted from the original data file: date, first ask price, first bid price and corresponding volumes for both prices.

The whole data set is split into training and testing subsets in various proportions. The following proportions are chosen: 0.5/0.5, 0.6/0.4, 0.7/0.3, 0.8/0.2 and 0.9/0.1, where the left hand digit represents the training set and the other one stands for the testing set.

Since some regressors (MLP and SVR) have more parameters to alter than the other (kNN and DT) and require much longer time to test all the possible combinations of parameters. Thus, not all the data proportions are tested for certain regressors.

5.1.3 Sliding window

The sliding window approach is used in order to evaluate performance of different regressors with respect to various parameter combinations.

The training set construction is based on in-windows w_{in} and out-windows w_{out} that can be described by the following formulas:

$$\begin{aligned} w_{\text{in}}(i) &= [s_i, s_{i+1}, ..., s_{i+(k-1)}] \\ w_{\text{out}}(i) &= [s_{i+k}, s_{i+k+1}, ..., s_{i+k+(m-1)}] \end{aligned} \tag{5.1}$$

where s_i is the i-th sample in the set, $i \in [1, n-k-m]$; k is width of in-window w_{in}; m is width of out-window w_{out}; n denotes total amount of samples in the data set.

Windows move iteratively with a single unit step until sample s_{n-k-m} is reached (figure 5.1).

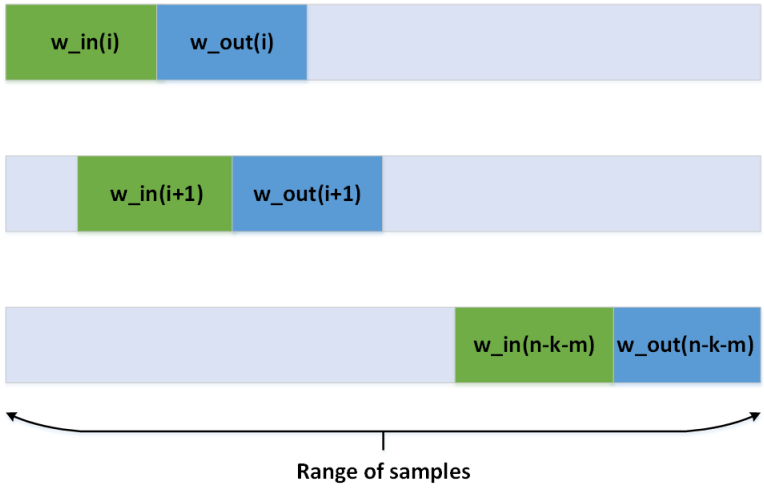


Figure 5.1: Sliding window approach

The initial window pair $(200, 60)$ (corresponds to $\approx$ 25 min and $\approx$ 8-10 min in real time respectively) is chosen arbitrarily. After a series of heuristic experiments, other window pair sizes are chosen according to yielded results.

The following window size combinations are chosen for further testing: $(20, 10)$, $(40, 20)$, $(60, 60)$, $(60, 200)$, $(200, 60)$, $(200, 200)$, where each pair is denoted as (w_{in}, w_{out}).

Depending on the results, one may conclude if the given data and regressor combination is more suitable for short- or long-term prediction.

5.1.4 Time performance

A timer is set for each of the regressor to measure its time expenses. Error computations are not included in the resulting time interval (except for the decision tree case).

5.2 Performance criteria

Error metrics Performance of the regression models is commonly characterized by metrics listed below [3]:

1. Mean absolute error
$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1} |y_i - \hat{y}_i| \tag{5.2}$$
2. Mean squared error
$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2 \tag{5.3}$$

where $\hat{y}_i$ is the predicted value for particular data sample x_i; y_i is the true value for x_i; n is the number of samples.

The metrics listed above usually belong to the interval $[0, 1]$. E.g. in our case, value 0.6 would mean that predicted value deviates from truth by 60% on average.

The smaller the error – the smaller the difference between the predicted and true output values and the better the prediction is.

Cross-validation Another way to estimate generalization performance is to apply a cross-validation technique. One of its variations is the k-fold cross validation.

The original data set must be divided into k equal subsets. One subset is used for testing and the remaining $k - 1$ are used for training. The procedure is repeated until each of the subsets is used for testing only once, that is k times in total. Averaged errors (MAE or MSE) are used for the regressor's evaluation.

The common choices are $k = 5$ and $k = 10$, but generally k is an arbitrary parameter. [57]

Computation time Time expenses are computed and recorded for every experiment. Time cost is a valuable performance criteria. However in this research it is not of first importance as opposed to prediction accuracy.

5.3 Details of testing procedure

5.3.1 Parameters to alter

The following parameters are altered across a series of experiments:

1. Data set proportion (defined in subsection 5.1.2)
2. Window size (defined in subsection 5.1.3)
3. Regressor parameters

(a) DT
    i. Tree depth
(b) kNN
    i. Number of neighbours
    ii. Weight function applied in prediction
        A. Uniform - all points are considered equally weighted
        B. Distance - point is weighted by inverse of its distance (points that are closer to the query one, have greater impact)
(c) MLP
    i. Number of hidden layers
    ii. Number of nodes on each of the hidden layers
    iii. Activation function for hidden layers
        A. Logistic: $f(x) = \frac{1}{1+\exp^{-ax}}$
        B. Hyperbolic tangent (tanh): $f(x) = \tanh(x)$
        C. Rectified linear unit function (relu): $f(x) = \max(0, x)$
    iv. Weight optimization algorithm
        A. Stochastic gradient descent[1] (sgd)
        B. Adaptive moment estimation [2] (adam)
    v. Learning rate for weights updating
(d) SVR
    i. Kernel
        A. Linear
        B. Polynomial (poly)
        C. Radial basis function (rbf)
        D. Sigmoid
    ii. Tube parameter ε (epsilon)
    iii. Penalty parameter C

5.3.2 Testing procedure overview

Testing procedure may be described by the following stages:

1. Initial heuristic testing of different combinations of window size pairs and parameter values on a particular data set proportion in order to narrow down the interval of parameter values for further testing

[1] Stochastic gradient descent is a stochastic optimization of gradient descent method, see subsection 3.3.3 for more information

[2] Adam - a stochastic gradient-based optimization algorithm, "based on adaptive estimates of lower-order moments" [41]

2. Structured testing is conducted using the narrowed selection of parameter values for all selected data proportions and window pair sizes

3. Find combination of data set proportion and regressor parameter values that yield the smallest errors for each window (see 5.2 for more information about error metrics)

4. Apply cross-validation for those combinations to evaluate performance of the regressor

5.3.3 Testing procedure specifications

The following list describes specifics of the testing procedure:

1. Only the parameters listed in subsection 5.3.1 are varied during all experiments.

2. Window size pairs described in section 5.1 are varied in different combinations with other parameters specified in subsection 5.3.1.

3. Error values are computed for each experiment and stored in the table.

4. Computation time is recorded and also considered for every experiment. However, time is not an exclusion criteria for working with a certain data set or regressor. Time cost for each experiment is stored in the same table as errors.

5. Combination of data set proportion and parameter value which yields the least errors within each window pair is chosen for further evaluation. More than one option may be chosen if the difference between their error values is considerably small.

6. Final estimation of the regressor is performed by cross-validation. This technique does not require data proportion information and is applied to the whole data set for validation process. Only the combination of window size and parameter is assessed within cross-validation. Depending on averaged error values the decision about quality of regressor's performance is made. (For more information see the corresponding part of section 5.2.)

5.3.4 Summary

This chapter described operations that were performed on given data in order to use it for further experimentation with regressors. The sliding window approach was explained and depicted. Performance criteria that were further used to evaluate performance quality of the regressor were introduced. Possibilities of regressors' parameter variations were reported in details. Stages of testing procedure and its specifications were listed.

Chapter 6

Implementation

The original data in `Bird_2006.csv` (see Appendix A) is stored in one cell per one row. For the sake of convenience, information within each cell is split into separate columns (e.g. bid price, ask price, bid volume, ask volume, etc.) so data would form a matrix.

The data set with split values is written into a new file with the same name `Bird_2006.csv` (see Appendix A) and is used for further manipulations.

6.1 Implementation details

The whole implementation procedure is described below:

1. Selection of the values to work with: date, first ask price, first bid price and corresponding volumes for both prices (implemented in `select_column.py`, Appendix B).

   The values listed above are stored in columns in separate file which is used for further manipulations (Appendix A).

2. Dividing the original data set into windows w_{in} and w_{out}. The window size pairs and the splitting procedure were described earlier in subsection 5.1.3 Sliding window. Window generation is implemented in `in_means_and_perc_changes.py`, Appendix B.

3. Computations of mean values of in-window and out-window are conducted according to the equations specified in 6.1. Implemented in `in_means_and_perc_changes.py` and `out_means.py` respectively, Appendix B.

$$\begin{aligned} \bar{w}_{\text{in}}(i) &= \frac{1}{k}\sum_{q=1}^{k} s_q(i) \\ \bar{w}_{\text{out}}(i) &= \frac{1}{m}\sum_{q=k}^{k+m} s_q(i), \end{aligned} \tag{6.1}$$

where s_i is the i-th sample in the set; k - width of in-window w_{in}; m - width of out-window w_{out}.

The procedure is held for each window within each column of the data set for each in- (w_{in}) and out- (w_{out}) window.

The computed values for particular pairs $(w_{\text{in}}, w_{\text{out}})$ are written and stored as a separate `.csv` file in the corresponding folder.

The mean values are important as they are used to calculate patterns in the data. It does not seem possible for the regressor to perform well on raw price values.

The price fluctuation interval is not known beforehand so it is more preferable to work with relative values such as deviation from mean within the window.

Percent changes between price values are always able to show the exact amount of changes within prices. It is decided to use percentage approach to denote the patterns.

4. Computation of percent changes between mean value of the in-window $\bar{w}_{\text{in}}(i)$ and raw values s_j within that window is performed according to equations specified in 6.2:

$$X = \frac{s_j - \bar{w}_{\text{in}(i)}}{\bar{w}_{\text{in}(i)}} * 100\% \tag{6.2}$$

where s_j is the j-th sample in the in-window, $j \in [1, k]$, $\bar{w}_{\text{in}}(i)$ - mean value of the i-th window from data set, $i \in [1, n - k - m]$.

The computed values represent patterns mentioned in the item above and further denoted as X.

The procedure is held for each in-window w_{in} for each column of data set.

5. The target value y is predicted by the regressor. y is a percent change between mean values of in-window $\bar{w}_{\text{in}}$ and mean of out-window $\bar{w}_{\text{out}}$ (equation 6.3).

$$y = \frac{\bar{w}_{\text{out}} - \bar{w}_{\text{in}}}{\bar{w}_{\text{in}}} * 100\% \tag{6.3}$$

6. A set of pairs of percent changes and targets is split into training and testing sets. Splitting of the data is implemented in file `read_split_data.py` (Appendix B).

   The training/testing set size proportions were described earlier in subsection 5.1.2 Data extraction.

7. Feeding training and testing data sets into regressors with different combinations of parameters, as a result the predicted values are yielded (figure 6.1).

   Implementation is represented in `implementation.py` (Appendix B).

   The following Scikit-learn implementations are used:

   (a) Regressors (see section 2.3)

i. `sklearn.neural_network.MLPRegressor`

ii. `sklearn.svm.SVR`

iii. `sklearn.neighbors.KNeighborsRegressor`

iv. `sklearn.tree.DecisionTreeRegressor`

(b) Error metrics (see section 5.2 Performance criteria)

i. `sklearn.metrics.mean_squared_error`

ii. `sklearn.metrics.mean_absolute_error`

Alterations of regressors' parameters are conducted according to subsection 5.3.1 Parameters to alter.

8. Error calculation for revealing the best parameters combinations (`implementation.py`, Appendix B)

9. Applying cross-validation technique for the sets of parameters that provided best results (validation on one-year data: `cross_validation.py`, two-year data: `cross_validation_2_years.py`, three-year data: `cross_validation_3_years.py`, Appendix B).

   The only best option is chosen according to the results of cross-validation.

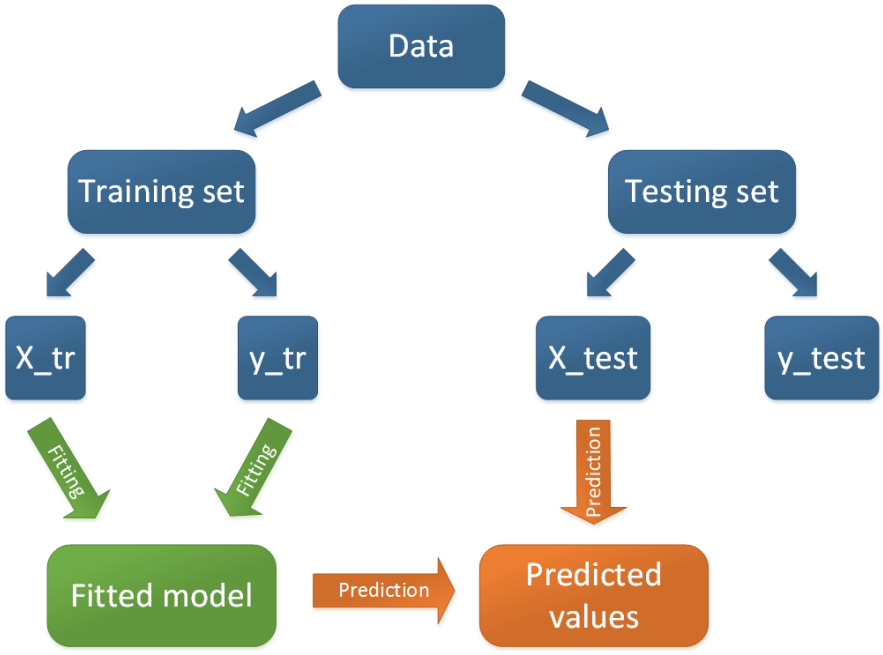


Figure 6.1: Feeding regressor with the data

6.1.1 Summary

The details of the test implementation were explained and supported with mathematical formulas and other relevant details. All steps of the implementation procedure were described and followed by references to the source code files.

Chapter 7

Experiment results

7.1 Decision tree regressor

The reference depth value interval is discovered in a series of heuristic experiments on data proportion 0.7/0.3 with windows (40,20) and (200,60) for a set of tree depth values $d = 3, 5, 10, 100$. The results are presented in the table 7.1.

Tree depth d	MAE	MSE	Time expenses t, sec
3	0.674	1.475	0.181
5	0.618	1.221	0.317
10	0.652	1.354	0.528
100	0.764	1.572	1.516

(a) Results for window pair $(40, 20)$

Tree depth d	MAE	MSE	Time expenses t, sec
3	1.335	4.496	1.579
5	1.233	4.161	2.554
10	1.341	4.71	4.646
100	1.439	4.892	8.784

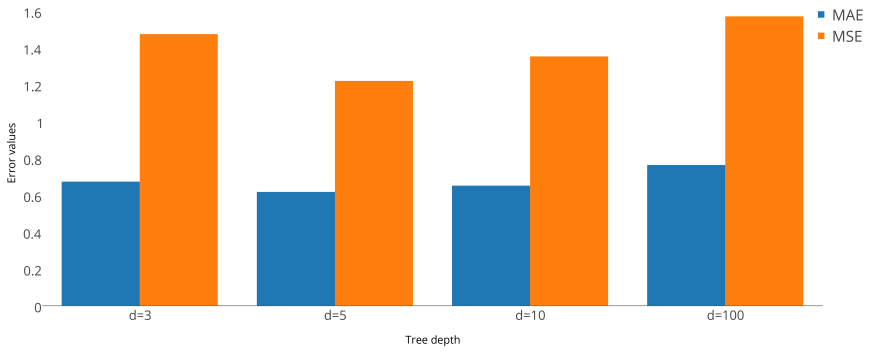
(b) Results for window pair $(200, 60)$

Table 7.1: Decision tree regressor. Results of heuristic experiments on data proportion 0.7/0.3

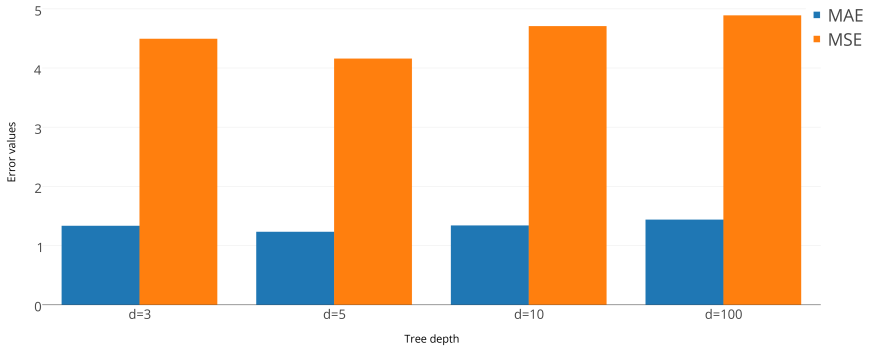
For convenience of comparison, the results are presented on the bar charts (figure 7.1).

According to the test results (table 7.1), deeper trees require more time to operate. Visual presentation is available for window pairs $(40, 20)$ and $(200, 60)$ (figure 7.2), but the statement appears to be true for all window pairs.

The tree depth between 3 and 10 yields acceptable error values (table 7.1). Thus, further tests are held for all windows and data proportions for tree depths within the interval $d = [3, 10]$.

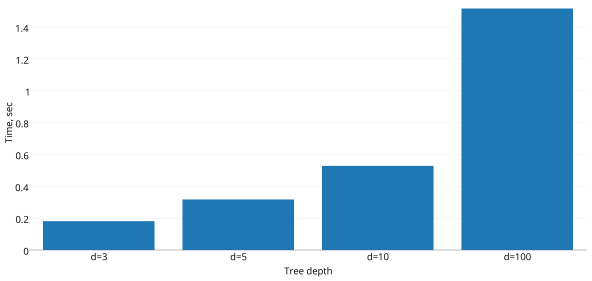


(a) Results for window pair (40,20)

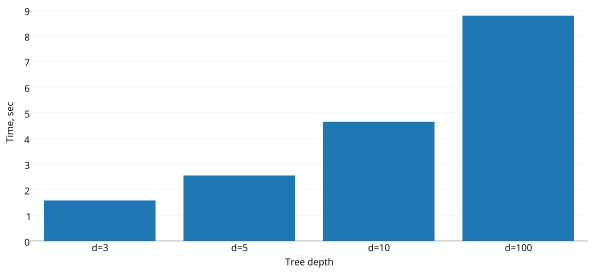


(b) Results for window pair (200,60)

Figure 7.1: Decision tree regressor. Results of heuristic experiments on data proportion 0.7/0.3



(a) Results for window pair (40,20)



(b) Results for window pair (200,60)

Figure 7.2: Decision tree regressor. Time expenses of heuristic experiments on data proportion 0.7/0.3

In order to reach better precision of the tree depth estimation, it is decided to test the values of the depths parameter within interval that provided good results: $d = 3, 4, 5, 6, 8$ and 10.

The best results within each window pair $(w_{\text{in}}, w_{\text{out}})$ are depicted in the table 7.2. Error values for each window pair are visualized in the figure 7.3.

Data proportion	Window pair	Tree depth	MAE	MSE	Time, sec
0.5/0.5	(20,10)	4	0.342	0.45	0.062
0.9/0.1	(60,60)	4	0.888	1.749	0.597
0.9/0.1	(60,200)	4	1.479	4.142	0.66
0.9/0.1	(200,60)	4	0.906	1.777	2.875
0.9/0.1	(200,200)	4	1.466	4.243	3.256
0.5/0.5	(40,20)	6	0.503	0.904	0.244

Table 7.2: Decision tree regressor. The best results of experiments for each window pair

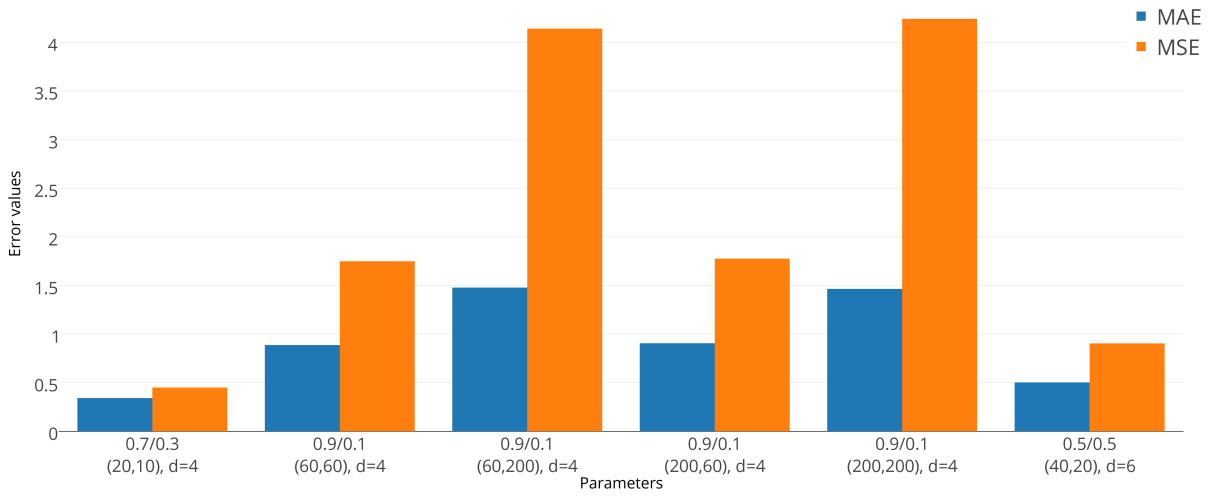


Figure 7.3: Decision tree regressor. The least error values achieved during experiments for each window pair

Table 7.2 and figure 7.3 demonstrate that window pairs, where w_{in} is insignificantly greater than w_{out}, provide better results.

See full experiment results in the file `DT.xlsx`, Appendix C.

Experiments are also performed on other one-year data sets (`Bird_2007` and `Bird_2008`). Results are mostly identical to those of data set `Bird_2006`. However, time expenses vary more often than error values due to different sizes of data sets (`Bird_2006` - 98 056 samples, `Bird_2007` - 84 963 samples, `Bird_2008` - 39 548 samples).

Only combinations that yields appropriate error values (that is, within interval [0,1]) are further estimated by k-fold cross-validation. Combinations and corresponding results are presented in the table 7.3.

Window pair $(w_{\text{in}}, w_{\text{out}})$	Tree depth d	MAE_{aver}	MSE_{aver}
(20,10)	4	0.294	0.317
(40,20)	6	0.425	0.595

(a) Results for $k = 5$

Window pair $(w_{\text{in}}, w_{\text{out}})$	Tree depth d	MAE_{aver}	MSE_{aver}
(20,10)	4	0.292	0.316
(40,20)	6	0.422	0.588

(b) Results for $k = 10$

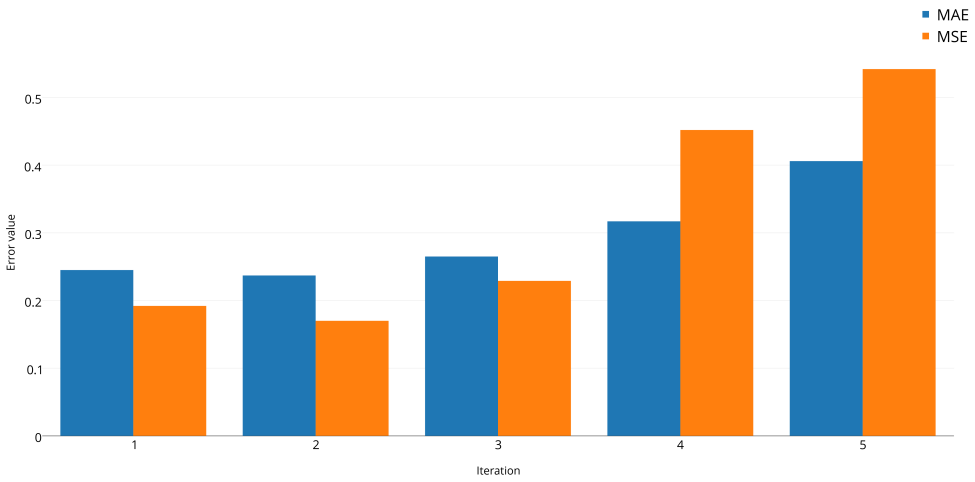
Table 7.3: Decision tree regressor. Values of averaged error values for k-fold cross-validation

Since results for $k = 5$ and $k = 10$ are insignificantly different (by order of 0.001), the rest of the regressors are only tested by 10-fold cross-validation.

Since cross-validation shows similar results for different k (for corresponding parameter combinations), the estimation of the model may be reliable.

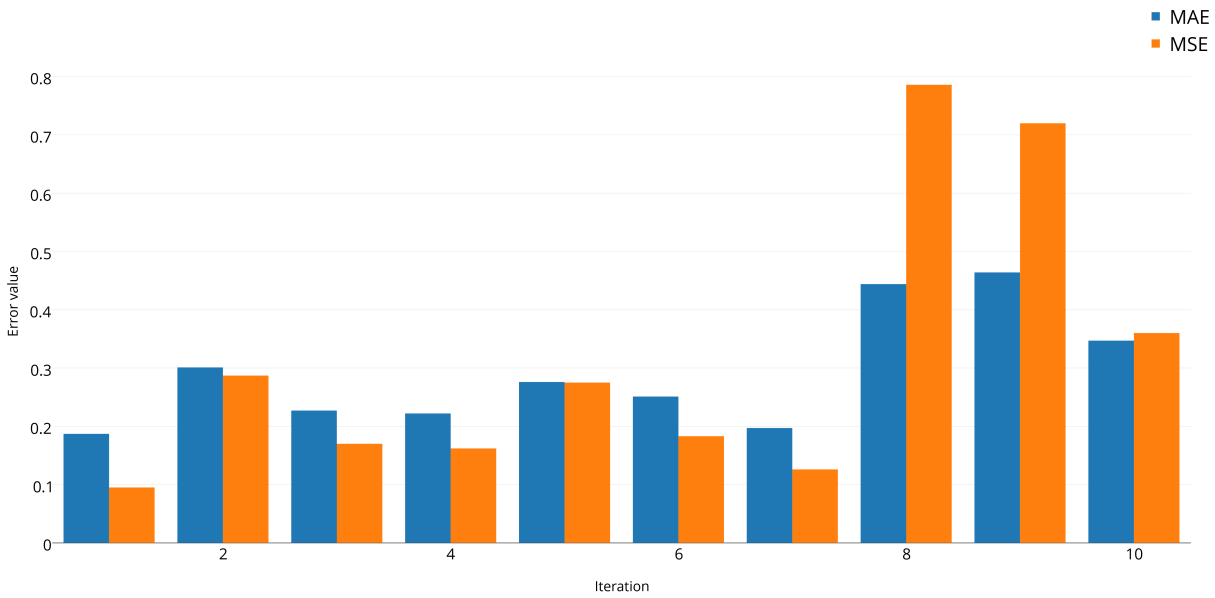
The least error values are reached for the tree depth $d = 4$, thus according to section 5.2 Performance criteria, it is the optimal parameter for decision tree regressor.

An interesting observation can be made based on the results of the k-fold cross-validation: latest error values (two latest for $k = 5$, three latest for $k = 10$) are higher than the others. The results are depicted on the figure 7.4.



(a) Results for $k = 5$

Figure 7.4 (continued)



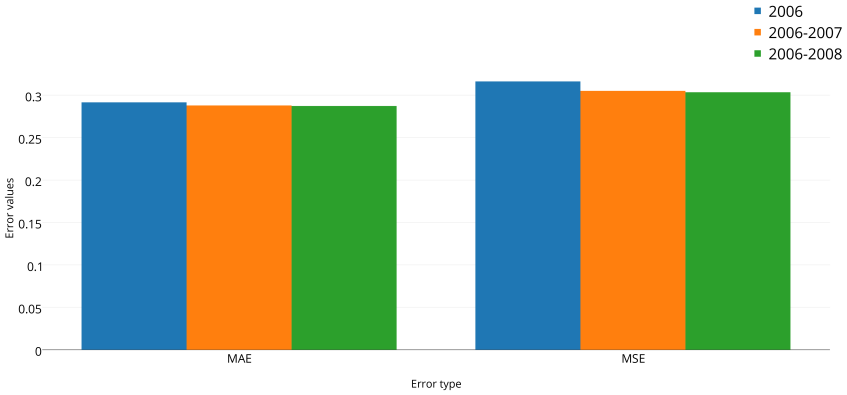
(b) Results for $k = 10$

Figure 7.4: Decision tree regressor. Iterative error values of k-fold cross-validation for best parameter combination - window pair $(20, 10)$, tree depth $d = 4$

It can be speculated that data samples within those subsets possess valuable patterns and/or properties of given data set.

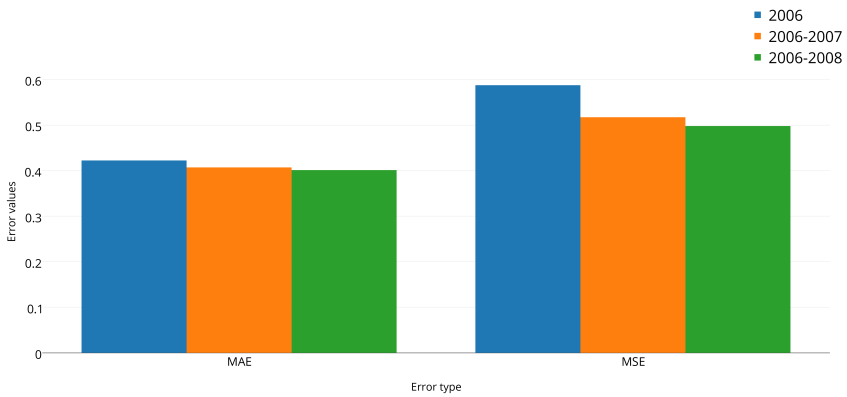
Cross-validation is also performed on 2-year and 3-year data files (`Bird_2006`+`Bird_2007` and `Bird_2006`+`Bird_2007`+`Bird_2008`). Larger data sets provides error values smaller by order of 0.001 and 0.01 for MAE and MSE respectively.

It can be concluded that larger training data sets for tree regressors can improve results by a small degree. For visual representations see figure 7.5.



(a) Error values for window pair $(20, 10)$

Figure 7.5 (continued)



(b) Error values for window pair $(40, 20)$

Figure 7.5: Decision tree regressor. Results of k-fold cross-validation for the best window pair, $k = 10$

For the results of k-fold cross-validation for the best window pair, $k = 5$ see file `cross_validation_DT (1-3 years).xlsx`, Appendix C.

Summary The following observations are made after a series of experiments on decision tree regressor on available data:

1. Increased tree depth leads to increased execution time (table 7.1 and figure 7.2)
2. Larger window sizes need more time for processing (table 7.1)
3. Tree regressors are more suitable for short-term prediction according to tests conducted on high-frequency stock data 7.3)
4. Larger training data set may provide minor prediction improvements (7.5)

7.2 Multilayer perceptron regressor

MLP regressor has quite a number of parameters that may be altered in different combinations with each other (subsection 5.3.1). In order to limit the number of possible parameter combinations, an attempt to discover optimal settings for different data shapes (window sizes and training/testing data set proportions) is made.

The first series of experiments is conducted for all combinations within parameters listed below:

1. Data proportions (training/testing): 0.5/0.5, 0.7/0.3 and 0.9/0.1
2. Window size pairs $(w_{\text{in}}, w_{\text{out}})$: $(40, 20)$ and $(200, 60)$

3. Hidden layers structure: $(100,)$[1], $(100, 100)$[2] and $(100, 100, 100)$[3]

4. Activation function: logistic

5. Weight optimization algorithm: sgd

6. Learning rate: 0.001

The three latter parameters are also changeable, but remained fixed for this series of experiment. The goal is to determine the optimal window size pair.

Data proportion	Hidden layers structure	MAE	MSE	Time, sec
0.5/0.5	(100,)	0.48	0.805	7.437
	(100,100)	0.488	0.842	25.097
	(100,100,100)	0.491	0.866	67.147
0.7/0.3	(100,)	0.61	1.196	9.720
	(100,100)	0.619	1.272	28.553
	(100,100,100)	0.629	1.327	80.931
0.9/0.1	(100,)	0.485	0.602	12.148
	(100,100)	0.489	0.609	37.957
	(100,100,100)	0.493	0.614	66.079

(a) Results for window pair $(40, 20)$

Data proportion	Hidden layers structure	MAE	MSE	Time, sec
0.5/0.5	(100,)	0.922	2.204	31.038
	(100,100)	0.915	2.223	31.777
	(100,100,100)	0.923	2.275	38.848
0.7/0.3	(100,)	1.125	3.177	37.421
	(100,100)	1.137	3.251	35.447
	(100,100,100)	1.16	3.37	52.270
0.9/0.1	(100,)	1.043	2.077	165.485
	(100,100)	1.089	2.188	223.827
	(100,100,100)	1.145	2.402	269.129

(b) Results for window pair $(200, 60)$

Table 7.4: Multilayer perceptron regressor. Results for the first series of experiments for fixed parameters: logistic activation function with sgd weight optimization algorithm, learning rate 0.001

Experiments are conducted on one-year data sets `Bird_2006`, `Bird_2007` and `Bird_2008` and yield similar results with only difference in performance time, which resembles behaviour

[1]One hidden layer with 100 neurons

[2]Two hidden layers with 100 neurons on each

[3]Three hidden layers with 100 neurons on each

of the DT regressor (see section 7.1). Further experiments are conducted on the data set `Bird_2006`.

All the test results for the MLP regressor are stored in tables (see file `MLP.xlsx`, Appendix C for more information).

According to the test results (table 7.4) the following observations are made:

1. The smallest errors are reached on data set proportions 0.5/0.5 and 0.9/0.1.
2. Increased number of hidden layers leads to higher error values (by order of 0.01) and time expenses (approximately proportional to number of layers).
3. Increased training set requires more time to operate.
4. Window pair of smaller sizes yields approximately twice better results.

Hence, only small window sizes are used for further manipulations. Choice of data proportions is narrowed down to 0.5/0.5 and 0.9/0.1 (table 7.4).

The impact of the learning rate on performance quality is studied on window pair $(40, 20)$ with data proportion 0.5/0.5 for all possible combinations of parameters:

1. Hidden layers structure: $(100,)$, $(100, 100)$ and $(100, 100, 100)$
2. Activation function: logistic, tanh, relu
3. Weight optimization algorithm: sgd, adam
4. Learning rate: 0.1, 0.001, 0.0001

Smaller rate requires more time for regressor to be trained, while greater rate require less time. Similar behaviour is observed within all experiments.

Variations of the learning rate affect error values by order of 0.001 and 0.01 (see `MLP.xlsx`, Appendix C). It is decided to proceed with the learning rate 0.001, defined as default in Scikit-learn MLP implementation.

It is observed that greater number of hidden layers in the regressor's structure results in greater error values similar to results in table 7.4. Only the structure with one hidden layer is further studied.

The goal of the next series of experiments is to reveal the best combinations of activation function and weight optimization algorithm. The results are presented in the table 7.5.

The smallest error values are reached for logistic activation function. It must be noted though, that sgd optimization algorithm requires more time, but is not always more precise than adam (table 7.5).

Activation function	Weight optimization algorithm	MAE	MSE	Time, sec
Logistic	sgd	0.48	0.805	7.437
	adam	0.479	0.788	2.035
Tanh	sgd	0.488	0.833	5.835
	adam	0.497	0.817	3.394
Relu	sgd	0.486	0.77	2.756
	adam	0.501	0.795	3.27

(a) Results for data proportion 0.5/0.5

Activation function	Weight optimization algorithm	MAE	MSE	Time, sec
Logistic	sgd	0.485	0.602	12.148
	adam	0.487	0.613	4.348
Tanh	sgd	0.505	0.641	17.862
	adam	0.521	0.670	9.508
Relu	sgd	0.507	0.633	7.297
	adam	0.511	0.642	5.157

(b) Results for data proportion 0.9/0.1

Table 7.5: Multilayer perceptron regressor. Results for window pair $(40, 20)$ with fixed parameters: learning rate 0.001, one hidden layer with 100 nodes: $(100,)$

Based on the results of the previous experiments, further tests are held for a limited choice of parameters:

1. Data proportion: 0.5/0.5 and 0.9/0.1
2. Hidden layers structure: one layer
3. Activation function: logistic
4. Weight optimization algorithm: sgd, adam
5. Learning rate: 0.001

Two more window size pairs are studied with all possible combinations of parameters determined above: $(20, 10)$ and $(60, 60)$.

According to the table 7.6, the best results are reached for window pair $(20, 10)$. Further experiments are only performed on this pair.

The number of neurons in the hidden layer is varied to see if there is a possibility of improvement within this particular parameter (table 7.7).

Increasing the number of nodes up to 500 negatively affects the results, but results for 200 nodes are slightly better than those for default 100 nodes. It is also observed that the layers with more neurons require more time to operate. See table 7.7 for more detailed information about the experiment results.

Data proportion	Weight optimization algorithm	MAE	MSE	Time, sec
0.5/0.5	sgd	0.331	0.45	6.573
	adam	0.324	0.435	1.684
0.9/0.1	sgd	0.322	0.342	10.421
	adam	0.322	0.343	2.313

(a) Results for window pair $(20, 10)$

Data proportion	Weight optimization algorithm	MAE	MSE	Time, sec
0.5/0.5	sgd	0.48	0.805	7.437
	adam	0.488	0.842	25.097
0.9/0.1	sgd	0.61	1.196	9.720
	adam	0.619	1.272	28.553

(b) Results for window pair $(40, 20)$

Data proportion	Weight optimization algorithm	MAE	MSE	Time, sec
0.5/0.5	sgd	0.866	2.126	12.034
	adam	0.872	2.14	4.373
0.9/0.1	sgd	0.85	1.578	14.728
	adam	0.864	1.598	5.032

(c) Results for window pair $(60, 60)$

Table 7.6: Multilayer perceptron regressor. Results for different small size windows with fixed parameters: logistic activation function, learning rate 0.001, one hidden layer with 100 nodes: $(100,)$

Since changes in error values are insignificant in real world, hidden layer with 100 neurons may be considered an optimal parameter for particular data set (`Bird_2006`). Precision of nodes quantity may be improved via more thorough experiments.

Optimization algorithm	Number of neurons	MAE	MSE	Time, sec
sgd	50	0.331	0.463	4.606
	100	0.331	0.45	6.573
	200	0.329	0.438	12.203
	500	0.342	0.438	26.763
adam	50	0.327	0.443	1.445
	100	0.324	0.435	1.684
	200	0.32	0.432	2.43
	500	0.338	0.426	6.038

(a) Results for data proportion 0.5/0.5

Optimization algorithm	Number of neurons	MAE	MSE	Time, sec
sgd	50	0.32	0.342	7.512
	100	0.322	0.342	10.421
	200	0.319	0.34	16.052
	500	0.323	0.341	34.177
adam	50	0.32	0.436	3.049
	100	0.322	0.344	2.313
	200	0.322	0.344	3.896
	500	0.355	0.353	6.951

(b) Results for data proportion 0.9/0.1

Table 7.7: Multilayer perceptron regressor. Results window pair $(20, 10)$ with fixed parameters: logistic activation function, learning rate 0.001, one hidden layer

The following combinations of parameters that yield the best results within all the experiments are further evaluated by k-fold cross-validation:

1. Window pair $(20, 10)$, one hidden layer $(100,)$, logistic activation function, sgd weight optimization algorithm, learning rate 0.001

2. Window pair $(20, 10)$, one hidden layer $(100,)$, logistic activation function, adam weight optimization algorithm, learning rate 0.001

Optimization algorithm	MAE	MSE
sgd	0.274	0.311
adam	0.276	0.311

(a) Results for $k = 5$

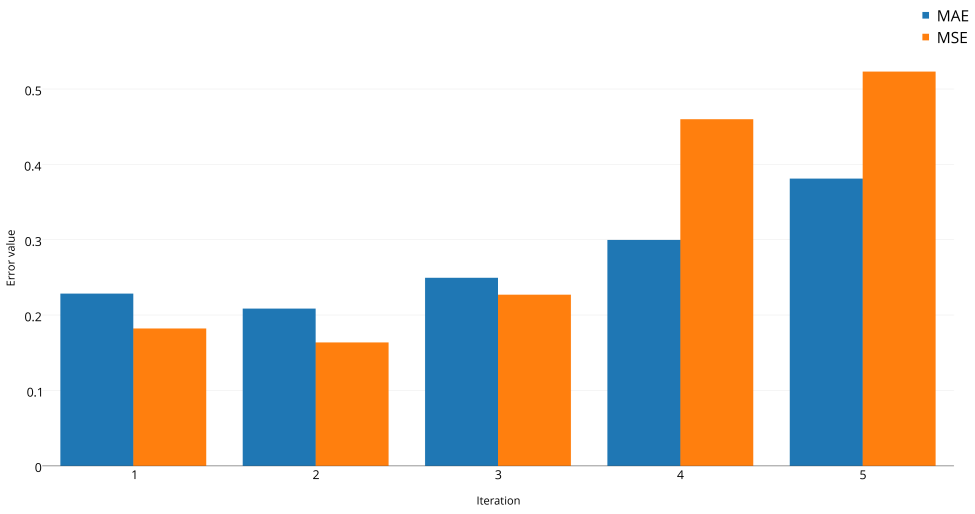
Optimization algorithm	MAE	MSE
sgd	0.273	0.311
adam	0.275	0.31

(b) Results for $k = 10$

Table 7.8: Multilayer perceptron regressor. Results of k-fold cross validation for the best paramter combinations: window pair $(20, 10)$, logistic activation function, one hidden layer $(100,)$, learning rate 0.001

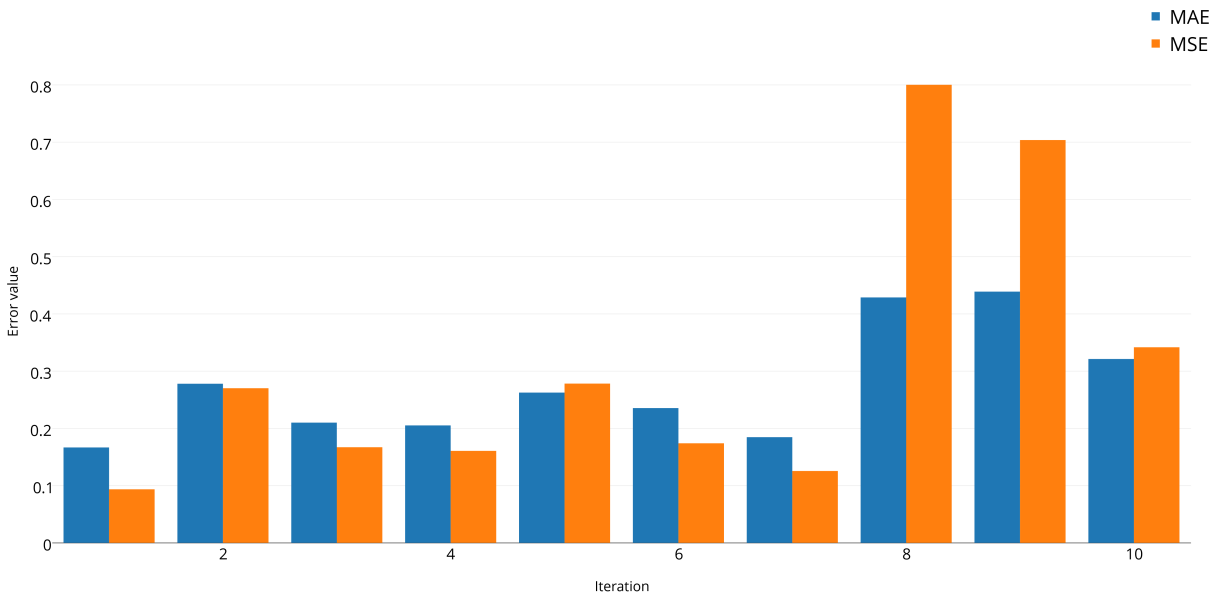
Cross-validation tests demonstrate better results than the experiments described earlier. Both sets of parameters that are evaluated, may be considered as optimal for particular data set. However, time expenses for sgd and adam differ depending on the training/testing data set proportion and window sizes (tables 7.5, 7.6 and 7.7).

Another observation can be made based on the results of the cross-validation: latest error values are higher than the others. The results are depicted on the figure 7.6. It can be speculated that these data subsets are more representative than other.



(a) Results for $k = 5$

Figure 7.6 (continued)



(b) Results for $k = 10$

Figure 7.6: Multilayer perceptron regressor. Iterative error values of k-fold cross-validation for best parameter combination - window pair $(20, 10)$, one hidden layer $(100,)$, logistic activation function, sgd weight optimization algorithm, learning rate 0.001

Cross-validation technique is held for data sets combined of two and three one-year data sets. According to the results (file `cross_validation_MLP (1-3 years).xlsx`), larger data set may improve results for sgd algorithm. But the data set size must be chosen carefully: error values for three-year data set demonstrates higher error values than those for one-year. Overfitting may be one of the reasons for such behaviour.

For adam algorithm the best results are reached on the one-year data set.

Summary The following observations are made for the MLP regressor based on a series of experiments:

1. Wider windows require more time to operate (table 7.4).

2. MLP regressor is more suitable for short-term prediction (tables 7.4 and 7.6).

3. Optimal learning rate with respect to error values and time expenses is 0.001 (file `MLP.xlsx`, Appendix C).

4. Larger training data sets require more time for processing, but may show minor prediction quality improvements (tables 7.4 and 7.5).

5. Greater number of hidden layers does not provide better results and requires more time to process (table 7.4).

6. Smaller error values are achieved for logistic activation function, however time expenses are not the lowest (table 7.5, file `MLP.xlsx`, Appendix C.

7.3 k Nearest neighbour regressor

The first series of experiments is aimed to discover the most suitable window size pairs and proportion on training data set. Combinations of the following parameters are used (table 7.9):

1. Data set proportions (training/testing): 0.5/0.5, 0.7/0.3 and 0.9/0.1

2. Window size pairs $(w_{\text{in}}, w_{\text{out}})$: $(40, 20)$ and $(200, 60)$

3. Amount of neighbours: 5, 10 and 50

4. Weight functions used for prediction: uniform and distance

Several observations are made based on test results (table 7.9):

1. The best results are achieved for smaller window pair, that is $(40, 20)$, on 0.5 and 0.9 proportions of the training set.

2. Error values that are reached with uniform weight function for all window pairs are slightly better (by order of 0.001) than those of distance function.

3. Greater number of neighbours results in growth of time expenses (figure 7.8), however error values are smaller.

It is decided to investigate smaller window sizes $(20, 10)$, $(40, 20)$ and $(60, 60)$ with a greater number of neighbours (100, 250 and 500) on 0.9 proportion of the training set.

Results for uniform weight function are presented in the table 7.10). Results for distance weight function are presented in the file `kNN.xlsx`, Appendix C.

The lowest error values are reached for 50 nearest neighbours. Precision of neighbour quantity may be obtained via further experiments around 50 neighbours (figure 7.7 and table 7.10).

As observed earlier (table 7.9), increased number of nearest neighbours leads to higher time expenses for all window size pairs (figure 7.8).

Window pair with the smallest sizes again showed the best results, in terms of both error values and time cost (table 7.10).

Data proportion	Weight function	Number of neighbours	MAE	MSE	Time, sec
0.5/0.5	uniform	5	0.607	1.031	14.664
		10	0.584	1.011	17.331
		50	0.574	1.088	23.327
	distance	5	0.617	1.045	14.058
		10	0.598	1.024	17.071
		50	0.591	1.093	24.218
0.7/0.3	uniform	5	0.72	1.459	12.978
		10	0.712	1.472	14.241
		50	0.724	1.633	19.519
	distance	5	0.722	1.462	12.437
		10	0.715	1.461	13.894
		50	0.729	1.622	19.074
0.9/0.1	uniform	5	0.602	0.872	5.144
		10	0.578	0.82	6.076
		50	0.559	0.758	8.352
	distance	5	0.61	0.879	5.612
		10	0.591	0.835	6.597
		50	0.57	0.775	8.384

(a) Results for window pair $(40, 20)$

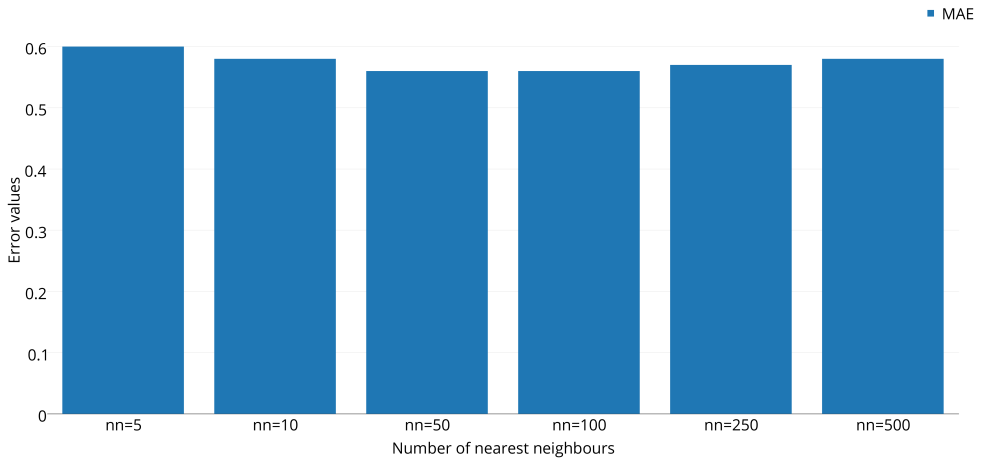
Data proportion	Weight function	Number of neighbours	MAE	MSE	Time, sec
0.5/0.5	uniform	5	1.282	3.73	131.064
		10	1.241	3.547	143.863
		50	1.154	3.321	163.642
	distance	5	1.282	3.731	114.89
		10	1.242	3.549	144.09
		50	1.154	3.305	150.5
0.7/0.3	uniform	5	1.516	5.047	80.198
		10	1.475	4.857	82.596
		50	1.405	4.747	101.002
	distance	5	1.516	5.048	97.899
		10	1.476	4.858	102.019
		50	1.403	4.718	114.258
0.9/0.1	uniform	5	1.18	2.806	45.628
		10	1.13	2.591	53.96
		50	1.013	2.092	61.483
	distance	5	1.180	2.807	53.203
		10	1.131	2.596	54.329
		50	1.015	2.096	60.769

(b) Results for window pair $(200, 60)$

Table 7.9: k Nearest neighbour regressor. Results for the first series of experiments

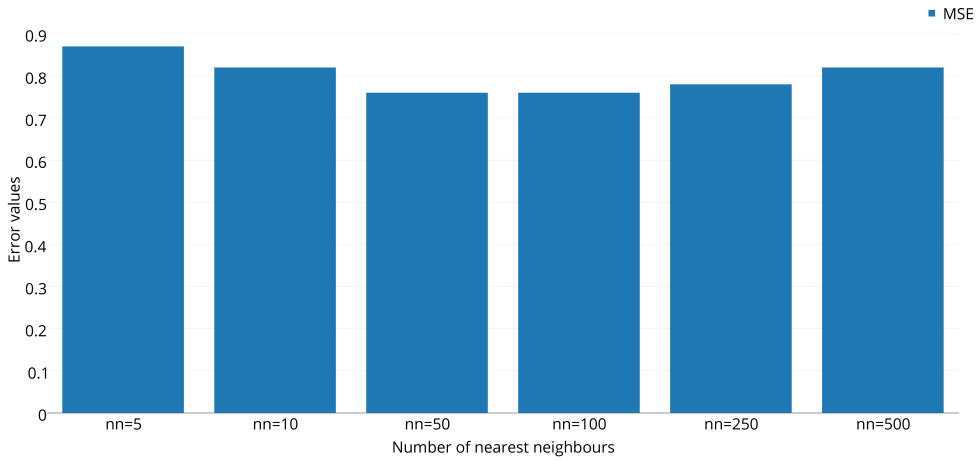
Window pair	Number of neighbours	MAE	MSE	Time, sec
$(20, 10)$	10	0.394	0.396	1.158
	50	0.379	0.405	1.868
	100	0.381	0.412	2.293
	250	0.39	0.435	3.505
	500	0.395	0.455	4.662
$(40, 20)$	10	0.578	0.82	6.076
	50	0.559	0.758	8.352
	100	0.56	0.76	8.6
	250	0.568	0.783	13.008
	500	0.579	0.816	15.296
$(60, 60)$	10	0.961	1.969	10.646
	50	0.895	1.728	16.914
	100	0.9	1.755	18.999
	250	0.908	1.814	24.938
	500	0.91	1.846	25.887

Table 7.10: k Nearest neighbour regressor. Results for the experiments on smaller window pairs, data proportion 0.9/0.1, uniform weight function



(a) Mean absolute error

Figure 7.7 (continued)

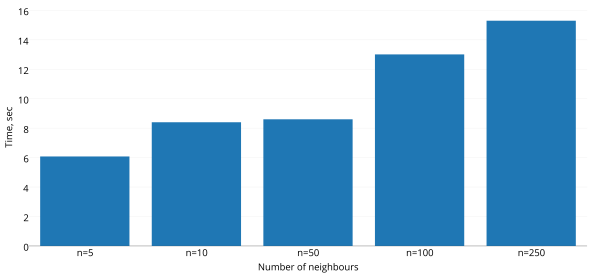


(b) Mean squared error

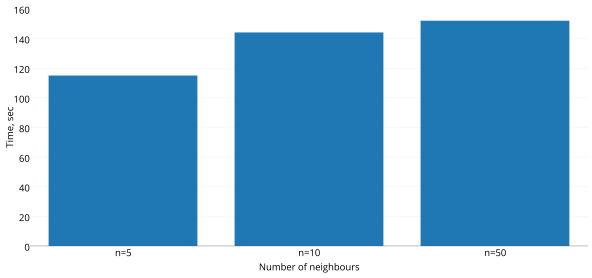
Figure 7.7: k Nearest neighbour regressor. Error values for window pair $(40, 20)$, data proportion 0.9/0.1, uniform weight function

The following parameters may be considered as optimal for particular data set with respect to high frequency trading: small width for both in- and out- windows (such as $(20, 10)$, training set consisting of 90% of the whole data set, uniform weight function and approximately 50 nearest neighbours.

The cross-validation technique is applied to estimate performance of the best combination of parameters (table 7.11).



(a) Time expenses for window pair $(40, 20)$



(b) Time expenses for window pair $(200, 60)$

Figure 7.8: k Nearest neighbour regressor. Time expenses for different amount of nearest neighbours

Data set	MAE	MSE
Bird_2006	0.322	0.361
Bird_2006+Bird_2007	0.316	0.334
Bird_2006+Bird_2007+Bird_2008	0.312	0.317

(a) $k = 5$

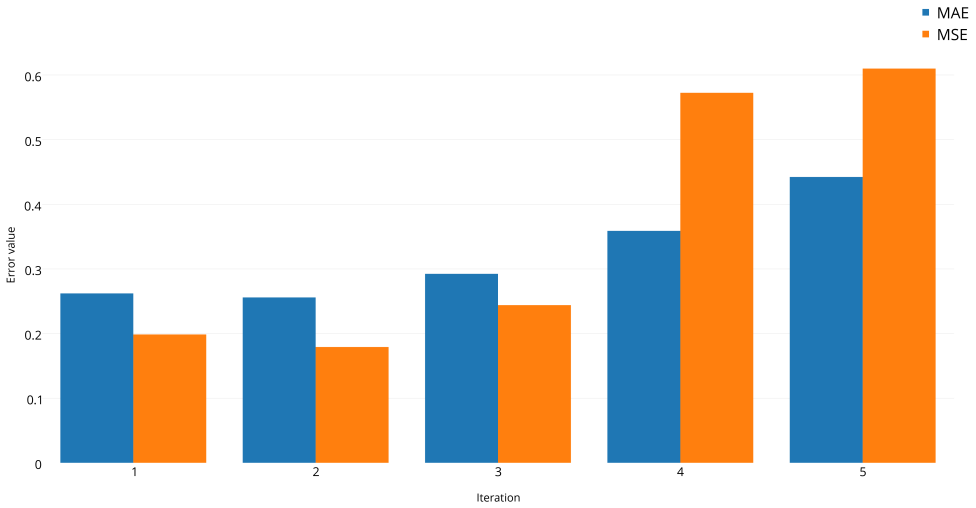
Data set	MAE	MSE
Bird_2006	0.321	0.36
Bird_2006+Bird_2007	0.316	0.332
Bird_2006+Bird_2007+Bird_2008	0.314	0.319

(b) $k = 10$

Table 7.11: k Nearest neighbour regressor. Results of k-fold cross-validation for single and combined data sets.

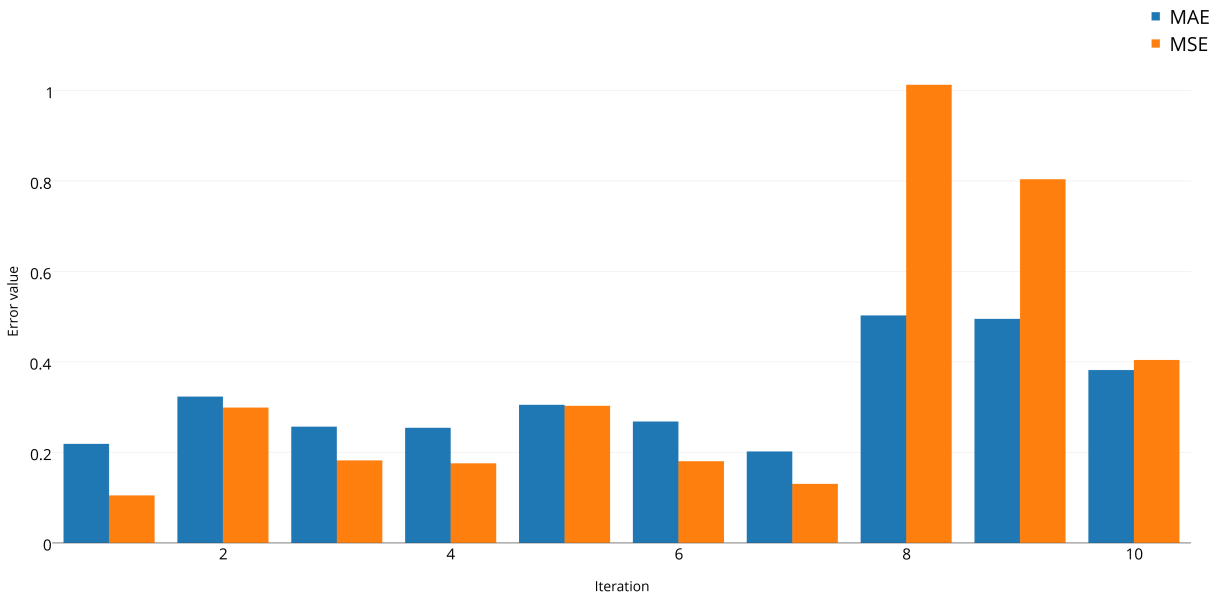
According to cross-validation results (table 7.11), performance of the kNN regressor may be improved by a small order by increasing the size of training set.

Error values within iterations of cross-validation show similar behaviour as for DT and MLP regressors: latter values are higher than previous (figure 7.9). It may be suspected that price patterns which are closer to the end of the year possess valuable information about data structure.



(a) Results for $k = 5$

Figure 7.9 (continued)



(b) Results for $k = 10$

Figure 7.9: k Nearest neighbour regressor. Iterative error values of k-fold cross-validation for best parameter combination - window pair $(20, 10)$, 50 neighbours and uniform weight function

Detailed information about test results is in the file `kNN.xslx`, Appendix C.

Summary Several important observations about the k-nearest neighbour regressor are listed below:

1. kNN is more efficient while making predictions on small window sizes. In other words, it appears to be more suitable for high frequency data analysis (tables 7.9, 7.10).

2. Greater amount of nearest neighbours does not necessarily imply better results, however require more time for regressor to operate (tables 7.9, 7.10, figures 7.7 and 7.8).

3. Size and composition of the training set are important as may either improve the performance or cause overfitting (tables 7.9 and 7.11).

7.4 Support vector regression

Computational complexity of SVR is of order n^2, where n - total amount of samples in the data set, and is due to dot products involved in kernel functions. Thus, the regressor may be efficient in terms of error values, but not time.

The first series of experiments is conducted to observe behaviour of the regressor on windows of different sizes - $(40, 20)$ and $(200, 60)$. Because of computational complexity only data proportion 0.5/0.5 was tested. Default[4] values are used for penalty and tube parameters: $C = 1$ and $\varepsilon = 0.1$ (see section 5.3.1 for more information).

Experiments are performed on three data sets: `Bird_2006`, `Bird_2007` and `Bird_2008` (see Appendix A). The results are nearly identical except for time expenses that can be explained by different amount of samples within each data set. Results for `Bird_2006` are displayed in the table 7.12.

All the test results are presented in the file `SVR.xlsx`, Appendix C.

Kernel	MAE	MSE	Time, sec
linear	0.469	0.747	104.091
poly, degree 2	0.982	12.166	50.996
poly, degree 3	0.982	12.166	47.291
poly, degree 4	0.982	12.166	48.657
rbf	0.556	1.246	39.785
sigmoid	0.822	2.004	32.382

(a) Results for window pair $(40, 20)$

Kernel	MAE	MSE	Time, sec
linear	0.865	2.101	1 145.661
poly, degree 2	4.414	249.029	307.122
poly, degree 3	4.414	249.029	296.486
poly, degree 4	4.414	249.029	307.413
rbf	1.356	5.297	276.542
sigmoid	1.771	8.163	171.909

(b) Results for window pair $(200, 60)$

Table 7.12: Support vector regression. Results for data proportion 0.5/0.5 with fixed parameters: $C = 1$, $\varepsilon = 0.1$

Experiments on smaller window display better results (table 7.12). It is decided to work with smaller windows, that is $(20, 10)$, $(40, 20)$ and $(60, 60)$.

The least error values are observed for linear kernel, however time expenses are high. Further experiments are held on both linear and rbf kernels for training set proportions 0.5, 0.7 and 0.9 (table 7.13).

The smallest error values are provided by the linear kernel for the smallest window pair $(20, 10)$. Between the two options of the training set proportion - 0.5 and 0.9 - the first one is chosen as it requires less time to operate.

A series of experiments is conducted on the window pair $(20, 10)$ for training set of 0.5 using linear kernel. Parameters C and ε are altered to seek for a possibility of performance improvement. The following results were obtained:

[4]Default values are the values suggested in Scikit-learn implementation documentation.

Kernel	Data proportion	MAE	MSE	Time, sec
linear	0.5/0.5	0.323	0.417	33.404
	0.7/0.3	0.397	0.597	51.32
	0.9/0.1	0.326	0.34	148.9
rbf	0.5/0.5	0.373	0.597	21.024
	0.7/0.3	0.463	0.891	33.429
	0.9/0.1	0.343	0.351	51.061

(a) Results for window pair $(20, 10)$

Kernel	Data proportion	MAE	MSE	Time, sec
linear	0.5/0.5	0.469	0.747	104.091
	0.7/0.3	0.582	1.092	173.886
	0.9/0.1	0.484	0.602	450.654
rbf	0.5/0.5	0.556	1.246	39.785
	0.7/0.3	0.718	1.925	63.063
	0.9/0.1	0.533	0.533	95.64

(b) Results for window pair $(40, 20)$

Kernel	Data proportion	MAE	MSE	Time, sec
linear	0.5/0.5	0.861	2.088	155.672
	0.7/0.3	1.084	3.117	282.735
	0.9/0.1	0.842	1.558	802.038
rbf	0.5/0.5	0.968	2.845	77.121
	0.7/0.3	1.241	4.304	114.503
	0.9/0.1	0.904	1.833	167.415

(c) Results for window pair $(60, 60)$

Table 7.13: Support vector regression. Results for different data proportions with fixed parameters: $C = 1$, $\varepsilon = 0.1$

Penalty parameter C	Tube parameter ε	MAE	MSE	Time, sec
1	0.1	0.323	0.417	33.404
1	0.01	0.31	0.413	92.752
1	0.001	0.308	0.414	601.378
0.5	0.1	0.323	0.417	21.216

Table 7.14: Support vector regression. Results for the experiments on smaller window pairs, data proportion 0.9/0.1

The penalty parameter C does not significantly affect performance quality, however reduces time expenses.

The tube parameter ε improves regressor's performance by a small order of 0.01, however time expenses grows significantly.

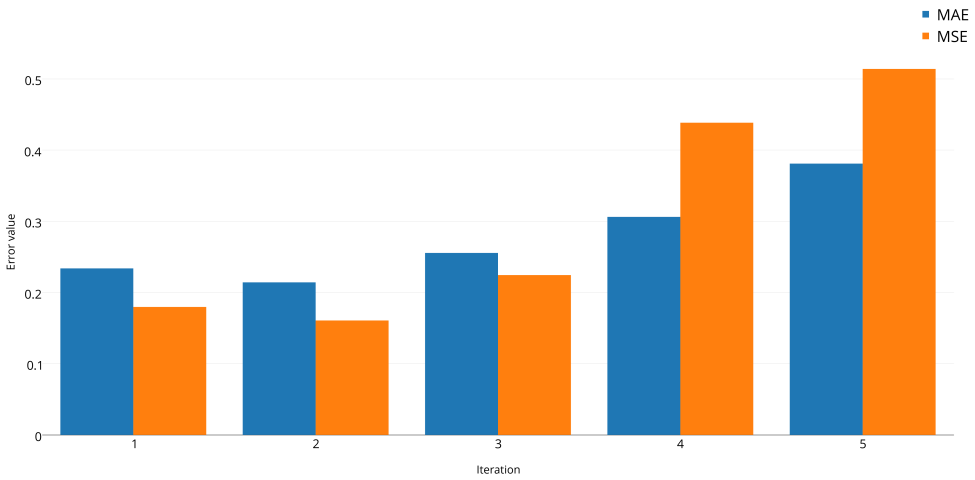
It may be concluded that default parameters of penalty and tube parameters ($C = 1$ and $\varepsilon = 0.1$ respectively) are the most suitable for particular data set.

Cross-validation is applied for window pair $(20, 10)$, linear kernel with default parameter values $C = 1$ and $\varepsilon = 0.1$ (table 7.15). Only one-year data set is used for cross-validation because of computational complexity.

Amount of folds	MAE	MSE
k=5	0.278	0.304
k=10	0.278	0.303

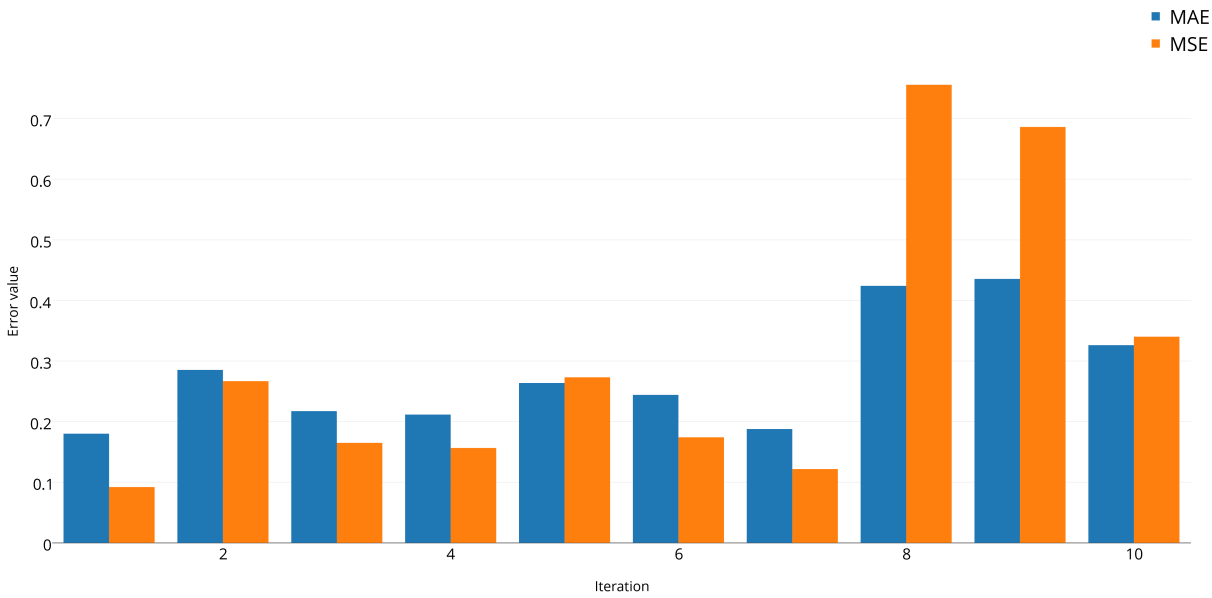
Table 7.15: Support vector regression. Results of k-fold cross-validation for one-year data set

Error values within iterations of cross-validation show similar behaviour as for DT, MLP and kNN regressors: latter values are higher than previous (figure 7.10). Presence of valuable information and price patterns around end of the year may be assumed according to the results of cross-validation.



(a) Results for $k = 5$

Figure 7.10 (continued)



(b) Results for $k = 10$

Figure 7.10: Support vector regression. Iterative error values of k-fold cross-validation for best parameter combination - window pair $(20, 10)$, linear kernel, $C = 1$, $\varepsilon = 0.1$

Summary The following observations are made about support vector regression:

1. SVR predicts better on small window sizes. It appears to be more suitable for high frequency data analysis (table 7.12).

2. Size and composition of the training set are of great importance as may either improve the performance or lead to overfitting (table 7.12).

3. The penalty parameter C does not affect the performance quality in terms of error values but reduces time expenses by several seconds (table 7.14).

4. The tube parameter ε improves the performance in terms of error values by a small order, however time expenses significantly increases (table 7.14).

5. Polynomial kernels (examined degrees: 2, 3 and 4) produce the highest error values and are time demanding (table 7.12.

7.5 Comparative analysis

Comparative analysis of four examined regressors is presented in this section. The goal is to reveal the most suitable regressor for high frequency data analysis.

Combinations of parameters that yields the best results for each regressor are listed below:

Decision tree: tree depth $d = 4$

Multilayer perceptron: one hidden layer with 100 neurons $(100,)$, logistic activation function, sgd and adam weight optimization algorithms, learning rate $\mu = 0.001$

***k* Nearest neighbour:** uniform weight function, 50 nearest neighbours

Support vector regression: linear kernel, penalty parameter $C = 1$, tube parameter $\varepsilon = 0.1$

Cross-validation demonstrated that all four regressors may reach better results with already determined parameters via better choice of the training set. Certain price patterns captured in the training set invokes performance improvements. Size of the training set also matters. It may either contain valuable price patterns and develop the performance quality or imply overfitting and worse results.

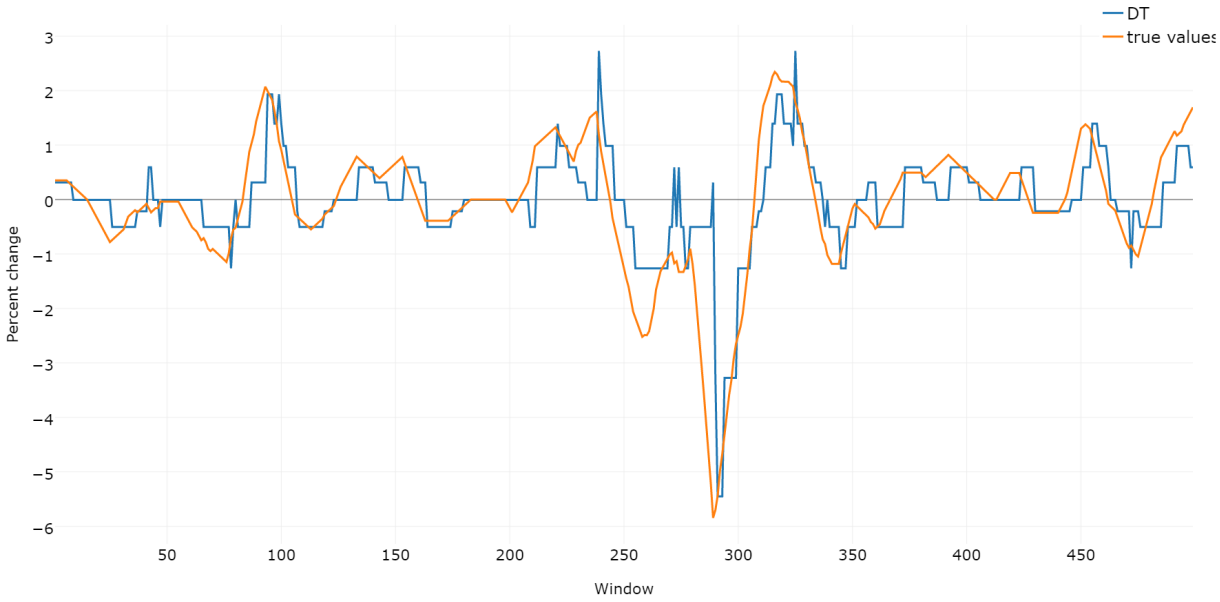
The results of k-fold cross validations for the best options within each regressor are presented in the table below.

Regressor	Parameter combination	MAE	MSE
DT	$d = 4$	0.292	0.316
MLP	$(100,)$, logistic, sgd, $\mu = 0.001$	0.273	0.311
MLP	$(100,)$, logistic, adam, $\mu = 0.001$	0.275	0.31
kNN	uniform, 50 neighbours	0.321	0.36
SVR	linear, $C = 1$, $\varepsilon = 0.1$	0.278	0.303

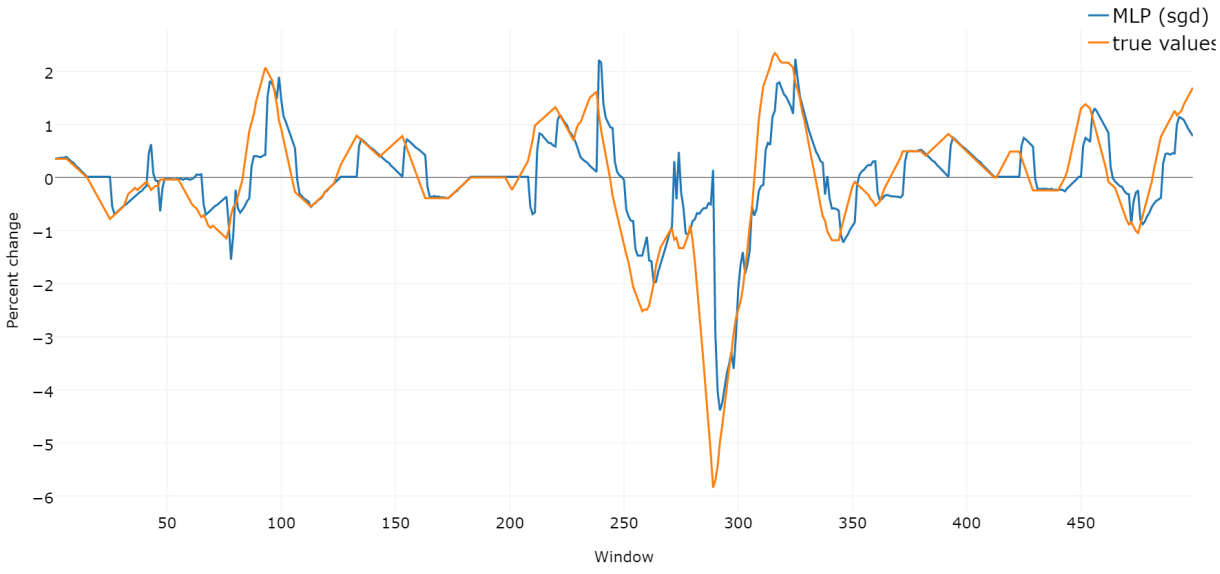
Table 7.16: Results of k-fold cross-validation for all examined regressors, $k = 10$

All the results in table 7.16 are reached within window pair $(20, 10)$, which is the smallest of examined.

True and predicted values for all studied regressors are depicted in the figure 7.11. Plotting interval is 500 windows, that is $\approx 5 - 8$ minutes.

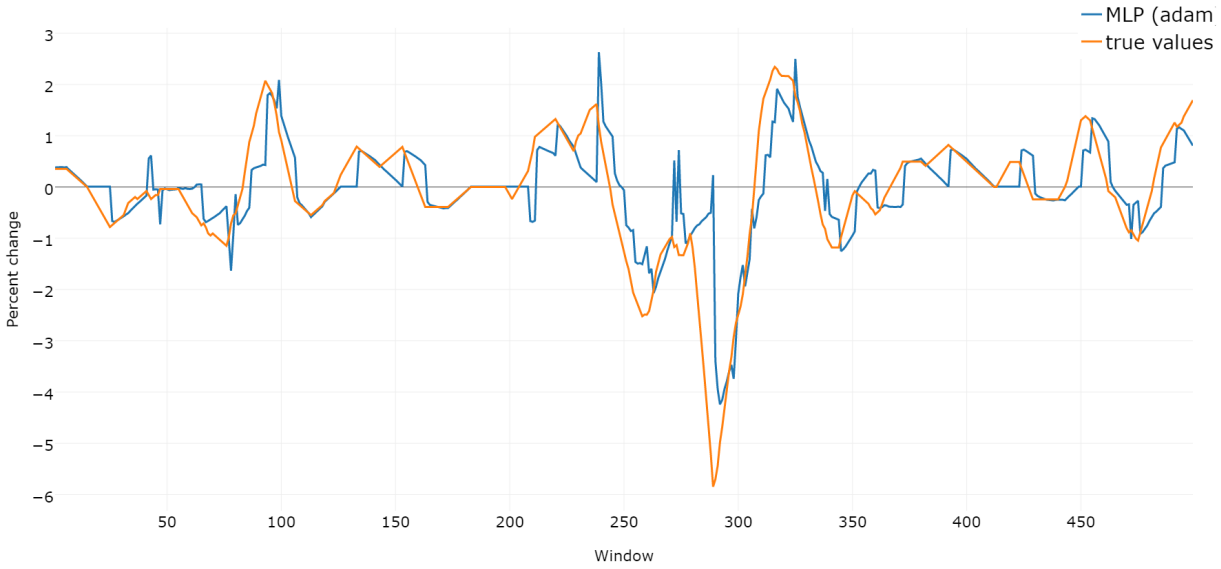


(a) Decision tree (DT) regressor

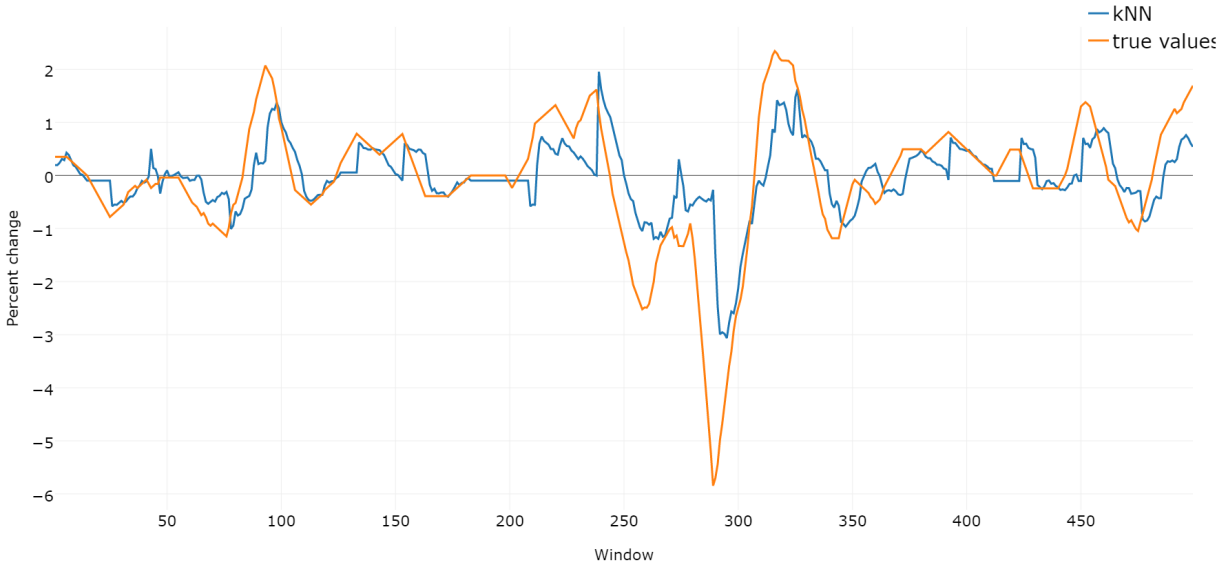


(b) Multilayer perceptron (MLP) regressor, sgd weight optimization algorithm

Figure 7.11 (continued)

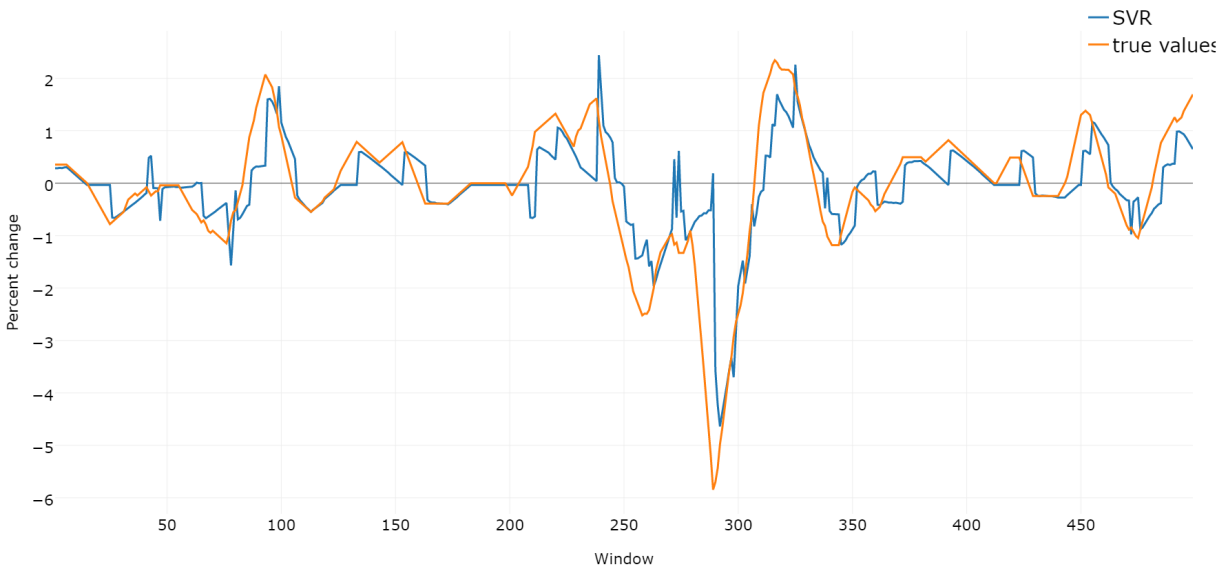


(c) Multilayer perceptron (MLP) regressor, adam weight optimization algorithm



(d) k Nearest neighbour regressor

Figure 7.11 (continued)



(e) Support vector regression

Figure 7.11: True and predicted values for the examined regressors

As one may observe (table 7.16), the smallest error values are achieved by the MLP regressor.

Another performance criteria that is evaluated is time expenses. It should be noted that window pair $(20, 10)$, which yielded the least error values, is aimed for prediction of prices $\approx 0.6 - 1$ minute in advance.

Not all the regressors may be applied to solve forecasting problem for given time interval. Training of SVR requires notably more time due to computational complexity which questions its suitability for real time trading applications with on-line learning.

However, if assigned problem allows higher error rate, rbf kernel may be used. Its error values is higher by order of 0.01, but time expenses are several times smaller.

The rest of the regressors demonstrates adequate time costs.

Time expenses for all four regressors are depicted in the figure 7.12.

All the examined regressors demonstrate reliability around 70%. Time expenses and implementation complexity may be the key criteria.

Since SVR is already excluded by time criteria, the rest of the regressors are considered in terms of complexity. DT and kNN have small amount of parameters that may be varied. On the other side, even precision of order 0.01 may be useful with respect to trading activity and thus MLP may be a better option.

Most of MLP's default parameters fit the assigned task very well. Within this research it was proven (see experiment results in the section 7.2) that default values for learning rate

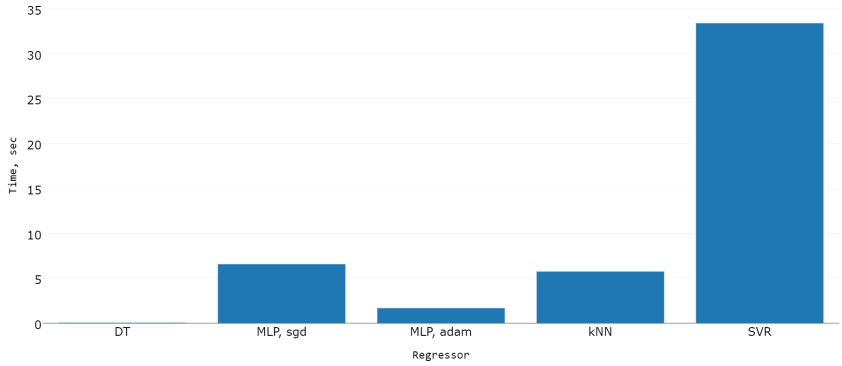


Figure 7.12: Time expenses for the examined regressors on training/testing set proportion 0.5/0.5

μ and structure of hidden layers coincide with the best options (at least for particular data set and in terms of high frequency data analysis).

Moreover, Scikit-learn implementation documentation [5] suggests to scale down the input within interval [0,1] for better results. It should be noted that the results were obtained from raw data, without any preprocessing.

According to results obtained within this research project, MLP regressor appears to be the most suitable solution for predicting short-term price fluctuations on stock market.

7.6 Summary

Four types of regressors were examined in this chapter: decision tree, multilayer perceptron, k nearest neighbour and support vector regression.

The main goal of all the experiments was to define a combination of regressor's parameters that yields the smallest error values and time expenses. Scikit-learn offers greater choice of parameters. Only the principal parameters were examined (see section 5.3.1).

Results of all the experiments were recorded in separate `.csv` files (see Appendix C) and thoroughly analysed. This chapter includes results that may be important for choosing the most fitting regressor for high frequency financial data analysis. Regressor analysis was based on performance criteria (see subsection 5.2).

MLP regressor demonstrated the best results in short-term prediction with respect to time expenses and prediction accuracy.

Chapter 8

Discussion

8.1 Findings from experiments

Results of experiments demonstrated that the best results were obtained for windows with the smallest width. This suggests short-term forecasting for high frequency financial data.

Certain regressor parameters imply better results in terms of error values, however requiring more time to operate. It must be decided whether it is more important to have higher precision of predicted values or lower time expenses. The latter question is of great importance with respect to high frequency trading.

8.2 Data related issues

Data preprocessing As it was mentioned before, all experiments were held on raw data. It is a known fact that raw data may contain noise, outliers, anomalies and other forms of disturbance. Data preprocessing may solve some of these problems.

The choice of the preprocessing technique must take into consideration whether the data analysis is going to be applied in real time or not. Preprocessing procedures with significant time expenses may not be the best solution for on-line learning.

Regressor type also matters. Particular regressor may prefer certain preprocessing technique to another or even a combination of those. However, some approaches require minimal or no preprocessing in order to function properly.

Choice of the training set The results of cross-validation demonstrated that subsets containing year-end data samples possess either valuable patterns of price movements important for learning/training or patterns that were not previously observed.

Statistical analysis may be applied to choose the training set that contains the most representative data samples.

The size of the training data set affects time expenses required for learning: greater data set requires more time to be processed. If the training set was chosen properly and

contains the most representative data samples and is small enough, time expenses for learning procedure will be minimal (for given regressor). This may be beneficial for an on-line learning application.

8.3 Issues concerning regressors

Parameters Each regressor has its own set of parameters. The only known way to choose the best parameters for particular problem is via experiments. This is a very time consuming task, especially if the data set is significantly large.

Such problems as overfitting, existence of local minima, vulnerability to noisy data and many others (see chapters 2 and 3) are still valid for regressors. More careful/accurate choice of training set, data preprocessing and regressor parameters may help to avoid some of the problems but there is still no universal solution.

Not all regressors provide the only solution for particular regression task. Results may vary every time the regressor is launched.

Chapter 9

Conclusions and future work

9.1 Conclusions

The motivation behind the research was to apply machine learning techniques for prediction of high frequency financial time series.

The goal was to experiment with open-source regressor implementations and reveal the most suitable for given data set.

In this paper, possible solution for price fluctuations prediction was proposed. This approach was tested on four regressors of different nature: decision tree, multilayer perceptron, k nearest neighbours and support vector. Experimental sessions revealed combinations of parameters for regressors that resulted in the best results in terms of predefined performance criteria for each regressor.

Comparative analysis of examined regressors helped to identify the best regressor with a set of corresponding parameters. Multilayer perceptron regressor demonstrated good performance and was chosen as the best. However, other regressors did not fall back dramatically and should still be considered as potential candidates.

Experiments and analysis provided valuable information about the most suitable prediction intervals. It was discovered that predictions for short time intervals (that is approximately 1 minute) provide better results. In other words, short-term forecasting using the proposed regressors may be more precise.

Variation of parameters during experimentation helped to understand the degree of their impact on performance of particular regressors. This knowledge can be used in further researches regarding analysis of high frequency financial data.

9.2 Future work

Data quality As was previously discussed (see chapter 8) introducing data preprocessing may positively affect the results. However, some regressors (such as MLP) are sensitive to particular preprocessing techniques.

Validation techniques Two metrics were applied for regression model validation (mean squared error and mean absolute error 5.2) but it is an ambiguous question which one reflects performance quality of regressor the best.

Applying other model estimation techniques may improve overview of particular regressor and display undiscovered benefits or drawbacks.

Possible directions for future work are further described.

1. It should be mentioned that within this research only historical data was considered as a source of information. None of the external factors (e.g. political, economical, etc.) were taken into consideration.

   Query analysis, preceding the regression modelling, may collect information about trends in financial market. This information may be used to assign corresponding weights to feature vectors that are used as input to the regressor. It may be possible to achieve better results with more complex composite prediction techniques that consist of several modules and/or layers instead of one regressor.

2. This research was aimed to predict rate of ask price fluctuations based on ask prices. The same procedure may be held for bid prices or volumes. This experiment may display if other patterns (bid price or volumes) can contribute to prediction quality.

3. Using a combination of ask and bid prices to predict (as well as volume data) the price movement may reveal hidden correlations and provide better results.

4. Only price values were used as the initial data source. Analysis may also be based on other values such as technical indicators or statistical measures.

5. Some of the examined regressors (such as MLP and kNN) support multiple output. In other words, more than one item may be predicted at once. It may be possible to predict values other than ask price movement.

6. Significant amount of scientific papers support the idea of combining approaches. One of the options is to integrate statistical modelling and machine learning techniques. Another is to combine several machine learning algorithms. Both approaches may be tested and evaluated.

7. Preliminary analysis of the data set may be used for choosing proper parameters for the regressor.

8. Using GPU-accelerated algorithms is a possibility to decrease time expenses for those regressors that provide good results but have high computational cost (such as SVR).

9. It maybe possible to create a self-trained system that would adjust itself based on on-line data stream from the stock market.

Appendix A

List of data files

The following data files were used for experimenting within the research:

1. `Bird_2006.csv` - raw data
2. `Bird_2006.csv` - split values
3. `Bird_2006.csv` - selected columns
4. `Bird_2007.csv` - raw data
5. `Bird_2008.csv` - raw data

Data files can be accessed at `https://dl.dropboxusercontent.com/u/29418662/thesis_appendices.zip`.

Appendix B

List of source code files

Files with Python implementations are listed below:

1. `split_file.py`
2. `select_column.py`
3. `in_means_and_perc_changes.py`
4. `out_means.py`
5. `out_perc_changes.py`
6. `read_split_data.py`
7. `regressors.py`
8. `implementation.py`
9. `cross_validation.py`
10. `cross_validation_2_files.py`
11. `cross_validation_3_files.py`

The source code is further presented and is also accessible at `https://dl.dropboxusercontent.com/u/29418662/thesis_appendices.zip`.

The source code for `split_file.py`:

```
import csv
import os

p = "D:\\thesis_data\\Raw\\"
pOut = "D:\\thesis_data\\Split\\"

files = os.listdir(path=p)

def splitData(filename):
    # open file to write split data
    with open(pOut + filename, 'w', newline='') as csvfile:

        csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                               quotechar=csv.excel.quotechar,
                               quoting=csv.excel.quoting)

        csvreader = csv.reader(open(p+filename, "rt"))

        while True:
            row = next(csvreader, 0)
            if row != 0:
                # split the row and write it down
                splitrow = row[0].split(';')
                csvwriter.writerow(splitrow)
            else:
                break

for filename in files:
    splitData(filename)
```

The source code for `select_column.py`:

```
import csv

# extracts 5 columns (date, ask, bid, ask volume, bid volume) from split data
    file and writes it down in a new file

def select_column(path_in, path_out, files_list, columns):
    for i in range(0,len(files_list)):

        with open(path_out + files_list[i], 'w', newline='') as csvfile:

            csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                                   quotechar=csv.excel.quotechar,
                                   quoting=csv.excel.quoting)

            csvreader = csv.reader(open(path_in+files_list[i], "rt"))

            while True:
                row = next(csvreader, 0)
                if row != 0:
                    importdata = []
                    importdata_temp = []

                    for j in range(0,len(columns)):
                        importdata_temp.append(row[columns[j]])

                    importdata.extend(importdata_temp)
                    csvwriter.writerow(importdata)
                else:
                    break

path_in = "D:\\thesis_data\\Split\\"
path_out = "D:\\thesis_data\\Selected 5 columns\\"
files_list = ['Bird_2006.csv','Bird_2007.csv','Bird_2008.csv']
columns = [0,1,6,11,16]

select_column(path_in, path_out, files_list, columns)
```

The source code for `in_means_and_perc_changes.py`:

```
import csv
import numpy as np

# calculates average value
def find_mean(arr):
    return np.average(arr, axis=0)

# calculates amount of rows in a file
def rows_amount(filename):
    with open(filename) as f:
        for i, line in enumerate(f, 1):
            pass
    return i

# calculates amount of columns in a file
def cols_amount(filename):
    with open(filename) as f:
        csvreader = csv.reader(open(filename, "rt"))
        num_cols = len(next(csvreader, 0))
        return num_cols

# calculates change between two values in percent
def percent_change(old,new): # old = average
    return np.divide((new - old), old)*100.00

# calculates mean of each window
# calculates percent change between each value of the window and its mean
def means_and_perc_ch(path_in, path_out, files_list, in_wind_width):
    # array of rows
    data_file = []

    for i in range(0,len(files_list)):
        total_rows = rows_amount(path_in+files_list[i])
        total_cols = cols_amount(path_in+files_list[i])

        # reading the file
        with open(path_in+files_list[i], "rt") as f:
            csvreader = csv.reader(f)

            for row in csvreader:
                data_file.append(row)

    means = [] # contains mean for each window
    perc_changes = [] # contains percent changes for each value in window
```

```
# calculate means and percent changes
for i in range(0,total_rows-in_wind_width-1):
    # array is one window
    array = np.zeros((in_wind_width,total_cols-1))

    for j in range(0,in_wind_width):
        for k in range(1,total_cols):
            # write columns to array
            array[j,k-1] = data_file[j+i][k]

    means_arr = find_mean(array)
    means.append(means_arr)
    perc_changes_arr = percent_change(means_arr, array)
    perc_changes.append(perc_changes_arr)

# write results to file
for i in range(0,len(files_list)):

    # write means
    print('writing file ' + files_list[i])

    with open(path_out + 'mean_' + 'in_' + str(in_wind_width) + '_' +
        files_list[i], 'w', newline='') as csvfile:

        csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                               quotechar=csv.excel.quotechar,
                               quoting=csv.excel.quoting)
        for j in range(0,len(means)):
            csvwriter.writerow(means[j])

    print('done ' + files_list[i])

    # write percent changes
    print('writing file ' + files_list[i] + ' 2')

    with open(path_out + 'perc_' + 'in_' + str(in_wind_width) + '_' +
        files_list[i], 'w', newline='') as csvfile:

        csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                               quotechar=csv.excel.quotechar,
                               quoting=csv.excel.quoting)

        for k in range(0,len(perc_changes)):
            for l in range(0,in_wind_width):
```

```
                    csvwriter.writerow(perc_changes[k][l])

        print('done ' + files_list[i] + ' 2')

path_in = "D:\\thesis_data\\Selected 5 columns\\"
path_out = "D:\\thesis_data\\Means and Percent changes (in)\\"
files_list = ['Bird_2006.csv','Bird_2007.csv','Bird_2008.csv']
in_wind_width = 200

means_and_perc_ch(path_in, path_out, files_list, in_wind_width)
```

The source code for out_means.py:

```
import csv
import numpy as np
from in_means_and_perc_changes import rows_amount, cols_amount, find_mean,
    percent_change

def means_out(path_in, path_out, files_list, in_wind_width, out_wind_width):
    # array of rows
    data_file = []

    for i in range(0,len(files_list)):
        total_rows = rows_amount(path_in+files_list[i])
        total_cols = cols_amount(path_in+files_list[i])

        # reading the file
        with open(path_in+files_list[i], "rt") as f:
            csvreader = csv.reader(f)

            for row in csvreader:
                data_file.append(row)

    out_means = [] # contains mean for each out_window

    for i in range(0,total_rows-in_wind_width-out_wind_width-1):
        out_array = np.zeros((out_wind_width,total_cols-1))

        for j in range(0,out_wind_width):
            for k in range(1,total_cols):
                #write cols to array
                out_array[j,k-1] = data_file[j+i+in_wind_width][k]

        # finds percent change between original data (in out_window) and its mean
        out_means_arr = find_means(out_array)
        out_means.append(out_means_arr)

    # write results to file
    for i in range(0,len(files_list)):

        # write means
        print('writing file ' + files_list[i])

        with open(path_out + 'mean_' + 'in_' + str(in_wind_width) + '_' + 'out_'
            + str(out_wind_width) + '_' + files_list[i], 'w', newline='') as
            csvfile:
```

```
            csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                                   quotechar=csv.excel.quotechar,
                                   quoting=csv.excel.quoting)
            for j in range(0,len(out_means)):
                csvwriter.writerow(out_means[j])

        print('done ' + files_list[i])

path_in = "D:\\thesis_data\\Selected 5 columns\\"
path_out = "D:\\thesis_data\\Means (out)\\"
files_list = ['Bird_2006.csv','Bird_2007.csv','Bird_2008.csv']
in_wind_width = 200
out_wind_width = 60

means_out(path_in, path_out, files_list, in_wind_width, out_wind_width)
```

The source code for out_perc_changes.py:

```
import csv
import numpy as np
from in_means_and_perc_changes import rows_amount, cols_amount, find_mean,
    percent_change

def perc_ch_out(path_in_1, path_in_2, path_out, files_pairs):

    for i in range(0,len(files_pairs)):
        total_rows_in = rows_amount(path_in_1+files_pairs[i][0])
        total_rows_out = rows_amount(path_in_2+files_pairs[i][1])

        means_in = []
        means_out = []

        out_perc_changes = []

        # reading the file means_in, means_out
        with open(path_in_1+files_pairs[i][0], "rt") as f:
            csvreader = csv.reader(f)

            for row in csvreader:
                means_in.append(np.asarray(row).astype(np.float))

        with open(path_in_2+files_pairs[i][1], "rt") as f:
            csvreader = csv.reader(f)

            for row in csvreader:
                means_out.append(np.asarray(row).astype(np.float))

        for j in range(0,total_rows_out):
            perc_changes_arr = percent_change(means_in[j], means_out[j])
            out_perc_changes.append(perc_changes_arr)

        # write results to file
        # write perc changes for out window
        print('writing file ', files_pairs[i])

        with open(path_out + 'perc_' + files_pairs[i][1], 'w', newline='') as
            csvfile:

            csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                                   quotechar=csv.excel.quotechar,
                                   quoting=csv.excel.quoting)
```

```
            for j in range(0,len(out_perc_changes)):
                csvwriter.writerow(out_perc_changes[j])

        print('done ', files_pairs[i])

path_in_1 = "D:\\thesis_data\\Means and Percent changes (in)\\"
path_in_2 = "D:\\thesis_data\\Means (out)\\"
path_out = "D:\\thesis_data\\Percent changes (out)\\"
files_pairs = [['mean_in_200_Bird_2006.csv','mean_in_200_out_60_Bird_2006.csv'],
               ['mean_in_200_Bird_2007.csv','mean_in_200_out_60_Bird_2007.csv'],
               ['mean_in_200_Bird_2008.csv','mean_in_200_out_60_Bird_2008.csv']]


perc_ch_out(path_in_1, path_in_2, path_out, files_pairs)
```

The source code for `read_split_data.py`:

```
import csv
import numpy as np

def rows_amount(filename):
    with open(filename) as f:
        for i, line in enumerate(f, 1):
            pass
    return i

def read_data(pathfile_in, filename_in, pathfile_out, filename_out,
    in_wind_width, col_number):
    X = []
    y = [] # only bid column

    # count rows
    total_rows = rows_amount(pathfile_in + filename_in)
    total_rows_Y = rows_amount(pathfile_out + filename_out)
    n_windows = total_rows_Y

    X = np.zeros((n_windows, in_wind_width))

    with open(pathfile_in+filename_in, "rt") as f:

        csvreader = csv.reader(f)
        for i in range (0, n_windows):
            for j in range (0, in_wind_width):
                row_index = i*in_wind_width + j
                row = next(csvreader)
                # parametrize number of column if needed
                X[i,j] = float(row[col_number])

    with open(pathfile_out+filename_out, "rt") as f:

            csvreader = csv.reader(f)
            for row in csvreader:
                y.append(float(row[col_number]))

    return X, np.asarray(y)

def split_data(X, y, tr_percent):
    length_tr = int(np.floor(tr_percent*(X.shape[0])))
    X_tr = np.zeros((length_tr,X.shape[1]))
    y_tr = np.zeros((length_tr,1))
```

```
for i in range(0,length_tr):
    for j in range(0,X.shape[1]):
        X_tr[i,j] = X[i][j]

    y_tr[i] = y[i]

length_test = int(np.floor((1-tr_percent)*(X.shape[0])))
X_test = np.zeros((length_test,X.shape[1]))
y_test = np.zeros((length_test,1))

for i in range(0,length_test):
    for j in range(0,X.shape[1]):
        X_test[i,j] = X[i+length_tr][j]

    y_test[i] = y[i+length_tr]

y_tr = y_tr.ravel()
y_test = y_test.ravel()

return X_tr, y_tr, X_test, y_test, length_test
```

The source code for `regressors.py`:

```
from sklearn.neural_network.multilayer_perceptron import MLPRegressor
from sklearn.svm import SVR
from sklearn import neighbors
from sklearn.tree import DecisionTreeRegressor

def MLPRegr(X_tr, y_tr, X_test, hidden_layers, learn_rate, activ_func, algorthm,
    n_iter):
    regr = MLPRegressor(hidden_layer_sizes=hidden_layers, activation=activ_func,
        algorithm=algorthm, learning_rate_init=learn_rate, max_iter=n_iter)
    # fit the model
    regr.fit(X_tr, y_tr)
    # predict
    y_pred = regr.predict(X_test)
    return y_pred

def SVRegr(X_tr, y_tr, X_test, kernl, eps, Cc):
    regr = SVR(kernel=kernl, epsilon=eps, C=Cc)
    # fit the model
    regr.fit(X_tr, y_tr)
    # predict
    y_pred = regr.predict(X_test)
    return y_pred

def kNNRegr(X_tr, y_tr, X_test, neighhbors, weight_type):
    regr = neighbors.KNeighborsRegressor(neighhbors, weights=weight_type)
    # fit the model
    regr.fit(X_tr, y_tr)
    # predict
    y_pred = regr.predict(X_test)
    return y_pred

def DTRegr(X_tr, y_tr, X_test, depth):
    regr = DecisionTreeRegressor(max_depth=depth)
    # fit the model
    regr.fit(X_tr, y_tr)
    # predict
    y_pred = regr.predict(X_test)
    return y_pred
```

The source code for `implementation.py`:

```
from read_split_data import read_data, split_data
from regressors import MLPRegr, SVRegr, kNNRegr, DTRegr
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import csv
import time
import numpy as np

pairs_list = [
             ["D:\\thesis_data\\Means and Percent changes (in)\\",
             "perc_in_20_Bird_2006.csv",
             "D:\\thesis_data\\Percent changes (out)\\",
             "perc_mean_in_20_out_10_Bird_2006.csv",
             20,
             0],
             ["D:\\thesis_data\\Means and Percent changes (in)\\",
             "perc_in_20_Bird_2007.csv",
             "D:\\thesis_data\\Percent changes (out)\\",
             "perc_mean_in_20_out_10_Bird_2007.csv",
             20,
             0],
             ["D:\\thesis_data\\Means and Percent changes (in)\\",
             "perc_in_20_Bird_2008.csv",
             "D:\\thesis_data\\Percent changes (out)\\",
             "perc_mean_in_20_out_10_Bird_2008.csv",
             20,
             0],
            ]

path_txt = "D:\\thesis_data\\Results\\"

data_proportion = 0.5

# parameters for regressors
MLP_hidden_layers = (500,)
MLP_learn_rate = 0.001
MLP_act_func = 'logistic'
MLP_alg = 'adam'
MLP_iterations = 10000

SVR_kernel = 'linear'
SVR_epsilon = 0.001
SVR_C = 0.5
```

```
kNN_neighhbors = 5
kNN_weights_type = 'uniform'

DT_depth = 4

for pair in pairs_list:
    # read data
    X, y = read_data(pair[0], pair[1], pair[2], pair[3], pair[4], pair[5])

    # split data into training and testing sets
    X_tr, y_tr, X_test, y_test, length_test = split_data(X, y, data_proportion)

    #1 MLP regressor
    start_time = time.perf_counter()

    y_pred = MLPRegr(X_tr, y_tr, X_test, MLP_hidden_layers, MLP_learn_rate,
        MLP_act_func, MLP_alg, MLP_iterations)

    y_test = np.nan_to_num(y_test)
    y_pred = np.nan_to_num(y_pred)

    end_time = time.perf_counter()

    MAE = mean_absolute_error(y_test, y_pred)
    MSE = mean_squared_error(y_test, y_pred)

    # write down the results
    with open(path_txt + 'MLP_res_' + 'set_' + str(data_proportion) + '_' +
        'hidden_' + str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg +
        '_' + 'rate_' + str(MLP_learn_rate) + '_' + pair[3] , 'w', newline='') as
        csvfile:

        csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                               quotechar=csv.excel.quotechar,
                               quoting=csv.excel.quoting)

        for i in range(0,len(y_pred)):
            csvwriter.writerow([y_pred[i],y_test[i]])

    with open(path_txt + 'MLP_err_' + 'set_' + str(data_proportion) + '_' +
        'hidden_' + str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg +
        '_' + 'rate_' + str(MLP_learn_rate) + '_' + pair[3] , 'w', newline='') as
        csvfile:
```

```
    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

    csvwriter.writerow(['MAE', MAE])
    csvwriter.writerow(['MSE', MSE])
    csvwriter.writerow(['number of hidden layers', MLP_hidden_layers])
    csvwriter.writerow(['learning rate', MLP_learn_rate])
    csvwriter.writerow(['activation function', MLP_act_func])
    csvwriter.writerow(['algorithm', MLP_alg])
    csvwriter.writerow(['execution time (sec)', end_time-start_time])

print('done with MLP')

#2 Support vector regressor
start_time = time.perf_counter()

y_pred = SVRegr(X_tr, y_tr, X_test, SVR_kernel, SVR_epsilon, SVR_C)

end_time = time.perf_counter()

MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)

# write down the results
with open(path_txt + 'SVR_res_' + 'set_' + str(data_proportion) + '_' +
    SVR_kernel + '_eps_' + str(SVR_epsilon) + '_C_' + str(SVR_C) + '_' +
    pair[3] , 'w', newline='') as csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

    for i in range(0,len(y_pred)):
        csvwriter.writerow([y_pred[i],y_test[i]])

with open(path_txt + 'SVR_err_' + 'set_' + str(data_proportion) + '_' +
    SVR_kernel + '_eps_' + str(SVR_epsilon) + '_C_' + str(SVR_C) + '_' +
    pair[3] , 'w', newline='') as csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

    csvwriter.writerow(['MAE', MAE])
```

```
    csvwriter.writerow(['MSE', MSE])
    csvwriter.writerow(['kernel', SVR_kernel])
    csvwriter.writerow(['epsilon', SVR_epsilon])
    csvwriter.writerow(['C', SVR_C])
    csvwriter.writerow(['execution time (sec)', end_time-start_time])

print('done with SVR')

#3 k-Nearest neighbours regressor
start_time = time.perf_counter()

y_pred = kNNRegr(X_tr, y_tr, X_test, kNN_neighhbors, kNN_weights_type)

end_time = time.perf_counter()

MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)

# write down the results
with open(path_txt + 'kNN_res_' + 'set_' + str(data_proportion) + '_neigh_' +
    str(kNN_neighhbors) + '_' + kNN_weights_type + '_' + pair[3] , 'w',
    newline='') as csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

    for i in range(0,len(y_pred)):
        csvwriter.writerow([y_pred[i],y_test[i]])

with open(path_txt + 'kNN_err_' + 'set_' + str(data_proportion) + '_' +
    '_neigh_' + str(kNN_neighhbors) + '_' + kNN_weights_type + '_' + pair[3]
    , 'w', newline='') as csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

    csvwriter.writerow(['MAE', MAE])
    csvwriter.writerow(['MSE', MSE])
    csvwriter.writerow(['number of neighbors', kNN_neighhbors])
    csvwriter.writerow(['weights', kNN_weights_type])
    csvwriter.writerow(['execution time (sec)', end_time-start_time])

print('done with kNN')
```

```
#4 Decision tree regressor

start_time = time.perf_counter()

y_pred = DTRegr(X_tr, y_tr, X_test, DT_depth)

MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)

end_time = time.perf_counter()

# write down the results
with open(path_txt + 'DT_res_' + 'set_' + str(data_proportion) + '_depth_' +
    str(DT_depth) + '_' + pair[3] , 'w', newline='') as csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

    for i in range(0,len(y_pred)):
        csvwriter.writerow([y_pred[i],y_test[i]])

with open(path_txt + 'DT_err_' + 'set_' + str(data_proportion) + '_depth_' +
    str(DT_depth) + '_' + pair[3] , 'w', newline='') as csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

    csvwriter.writerow(['MAE', MAE])
    csvwriter.writerow(['MSE', MSE])
    csvwriter.writerow(['DT_depth', DT_depth])
    csvwriter.writerow(['execution time (sec)', end_time-start_time])

print('done with DT')
print('-----------------')
```

The source code for `cross_validation.py`:

```
from read_split_data import read_data
from regressors import MLPRegr, SVRegr, kNNRegr, DTRegr
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
import csv

pairs_list = [
             ["D:\\thesis_data\\Means and Percent changes (in)\\",
             "perc_in_20_Bird_2006.csv",
             "D:\\thesis_data\\Percent changes (out)\\",
             "perc_mean_in_20_out_10_Bird_2006.csv",
             20,
             0]
             ]

path_out = "D:\\thesis_data\\Cross-validation\\"
short_name = "iw_20_ow_10_Bird_2006.csv"

q = 5 # number of subsets for cross-validation

# regressors' parameters
DT_depth = 6

MLP_hidden_layers = (100,)
MLP_learn_rate = 0.001
MLP_act_func = 'logistic'
MLP_alg = 'sgd'
MLP_iterations = 10000

kNN_neighhbors = 50
kNN_weights_type = 'uniform'

SVR_kernel = 'linear'
SVR_epsilon = 0.1
SVR_C = 1

for pair in pairs_list:
    X, y = read_data(pair[0], pair[1], pair[2], pair[3], pair[4], pair[5])

    subset_size = X.shape[0]//q

    X_arr = []
    y_arr = []
```

```
y_pred_arr = []
y_pred_internal_arr = []

MAE = []
MSE = []
MAE_internal = []
MSE_internal = []


# splitting data into q subsets
for i in range(0,q):
    X_temp = np.zeros((subset_size,X.shape[1]))

    for j in range(0,subset_size):
        for k in range(0,X.shape[1]):
            X_temp[j,k] = X[i*subset_size+j,k]

    X_arr.append(X_temp)

    y_temp = np.zeros((subset_size,1)).ravel()

    for j in range(0,subset_size):
        y_temp[j] = y[i*subset_size+j]

    y_arr.append(y_temp)

# regressor
for i in range(0,q):
    arr_x_toconcat = []
    arr_y_toconcat = []

    for j in range(0,q):
        if i!=j:
            arr_x_toconcat.append(X_arr[j])
            arr_y_toconcat.append(y_arr[j])

    X_tr = np.concatenate(arr_x_toconcat)
    y_tr = np.concatenate(arr_y_toconcat)
    X_test = X_arr[i]
    y_test = y_arr[i]

    #y_pred = DTRegr(X_tr, y_tr, X_test, DT_depth) # on q subset
    #y_pred_internal = DTRegr(X_tr, y_tr, X_tr, DT_depth) # on q-1 subsets
```

```
    #y_pred = MLPRegr(X_tr, y_tr, X_test, MLP_hidden_layers, MLP_learn_rate,
        MLP_act_func, MLP_alg, MLP_iterations) # on q subset
    #y_pred_internal = MLPRegr(X_tr, y_tr, X_tr, MLP_hidden_layers,
        MLP_learn_rate, MLP_act_func, MLP_alg, MLP_iterations) # on q-1 subsets

    #y_pred = kNNRegr(X_tr, y_tr, X_test, kNN_neighhbors, kNN_weights_type)
    #y_pred_internal = kNNRegr(X_tr, y_tr, X_tr, kNN_neighhbors,
        kNN_weights_type)

    y_pred = SVRegr(X_tr, y_tr, X_test, SVR_kernel, SVR_epsilon, SVR_C)
    y_pred_internal = SVRegr(X_tr, y_tr, X_tr, SVR_kernel, SVR_epsilon, SVR_C)

    y_pred_arr.append(y_pred)
    y_pred_internal_arr.append(y_pred_internal)

    MAE.append(mean_absolute_error(y_test, y_pred))
    MAE_internal.append(mean_absolute_error(y_test,
        y_pred_internal[:len(y_test)]))

    MSE.append(mean_squared_error(y_test, y_pred))
    MSE_internal.append(mean_squared_error(y_test,
        y_pred_internal[:len(y_test)]))

    with open(path_out + 'SVR_internal_' + 'q_' + str(q) + '_' + SVR_kernel +
        '_C_' + str(SVR_C) + '_eps_' + str(SVR_epsilon) + '_' + short_name ,
        'w', newline='') as csvfile:
#with open(path_out + 'kNN_internal_' + 'q_' + str(q) + '_' + 'neigh_' +
    str(kNN_neighhbors) + '_' + kNN_weights_type + '_' + short_name , 'w',
    newline='') as csvfile:
# with open(path_out + 'MLP_internal_' + 'q_' + str(q) + '_' + 'hidden_' +
    str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg + '_' +
    'rate_' + str(MLP_learn_rate) + '_' + short_name , 'w', newline='') as
    csvfile:
# with open(path_out + 'DT_internal_' + 'q_' + str(q) + '_' + 'depth_' +
    str(DT_depth) + '_' + short_name , 'w', newline='') as csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

    for i in range(0,len(y_pred)):
        csvwriter.writerow([y_pred_internal[i],y_test[i]])

with open(path_out + 'SVR_single_' + 'q_' + str(q) + '_' + SVR_kernel + '_C_'
    + str(SVR_C) + '_eps_' + str(SVR_epsilon) + '_' + short_name , 'w',
```

```
        newline='') as csvfile:
    #with open(path_out + 'kNN_single_' + 'q_' + str(q) + '_' + 'neigh_' +
        str(kNN_neighhbors) + '_' + kNN_weights_type + '_' + short_name, 'w',
        newline='') as csvfile:
    #with open(path_out + 'MLP_single_' + 'q_' + str(q) + '_' + 'hidden_' +
        str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg + '_' +
        'rate_' + str(MLP_learn_rate) + '_' + short_name, 'w', newline='') as
        csvfile:


        csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                               quotechar=csv.excel.quotechar,
                               quoting=csv.excel.quoting)

        for i in range(0,len(y_pred)):
            csvwriter.writerow([y_pred[i],y_test[i]])

    mean_MAE = np.mean(MAE)
    mean_MAE_internal = np.mean(MAE_internal)

    mean_MSE = np.mean(MSE)
    mean_MSE_internal = np.mean(MSE_internal)

    with open(path_out + 'SVR_cross_validation_err_' + 'q_' + str(q) + '_' +
        SVR_kernel + '_C_' + str(SVR_C) + '_eps_' + str(SVR_epsilon) + '_' +
        short_name , 'w', newline='') as csvfile:
    #with open(path_out + 'kNN_cross_validation_err_' + 'q_' + str(q) + '_' +
        'neigh_' + str(kNN_neighhbors) + '_' + kNN_weights_type + '_' +
        short_name , 'w', newline='') as csvfile:
    #with open(path_out + 'MLP_cross_validation_err_' + 'q_' + str(q) + '_' +
        'hidden_' + str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg +
        '_' + 'rate_' + str(MLP_learn_rate) + '_' + short_name , 'w', newline='')
        as csvfile:

        csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                               quotechar=csv.excel.quotechar,
                               quoting=csv.excel.quoting)

        csvwriter.writerow(['MAE', MAE])
        csvwriter.writerow(['averaged MAE', mean_MAE])
        csvwriter.writerow([' '])
        csvwriter.writerow(['MSE', MSE])
        csvwriter.writerow(['averaged MSE', mean_MSE])

print('done')
```

The source code for `cross_validation_2_files.py`:

```
from read_split_data import read_data
from regressors import MLPRegr, SVRegr, kNNRegr, DTRegr
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
import csv

pathfile_in = "D:\\thesis_data\\Means and Percent changes (in)\\"

short_name = "iw_20_ow_10_Bird_2006-2007.csv"

filename_in_1 = "perc_in_20_Bird_2006.csv"
filename_in_2 = "perc_in_20_Bird_2007.csv"

pathfile_out = "D:\\thesis_data\\Percent changes (out)\\"

filename_out_1 = "perc_mean_in_20_out_10_Bird_2006.csv"
filename_out_2 = "perc_mean_in_20_out_10_Bird_2007.csv"

path_out = "D:\\thesis_data\\Cross-validation\\"

in_wind_width = 20
column_number = 0

# regressors parameters
DT_depth = 6

SVR_kernel = 'linear'
SVR_epsilon = 0.1
SVR_C = 1

kNN_neighhbors = 50
kNN_weights_type = 'uniform'

MLP_hidden_layers = (100,)
MLP_learn_rate = 0.001
MLP_act_func = 'logistic'
MLP_alg = 'sgd'
MLP_iterations = 10000

q = 10 # number of subsets for cross validation

# get data from one year data set
X_1, y_1 = read_data(pathfile_in, filename_in_1, pathfile_out, filename_out_1,
```

```
    in_wind_width, column_number)

# get data from another year data set
X_2, y_2 = read_data(pathfile_in, filename_in_2, pathfile_out, filename_out_2,
    in_wind_width, column_number)

# combine two one-year data sets
X = np.concatenate((X_1, X_2), axis=0)
y = np.concatenate((y_1, y_2), axis=0)

subset_size = X.shape[0]//q

X_arr = []
y_arr = []

y_pred_arr = []
y_pred_internal_arr = []

MAE = []
MSE = []
MAE_internal = []
MSE_internal = []

# splitting data into q subsets
for i in range(0,q):
    X_temp = np.zeros((subset_size,X.shape[1]))

    for j in range(0,subset_size):
        for k in range(0,X.shape[1]):
            X_temp[j,k] = X[i*subset_size+j,k]

    X_arr.append(X_temp)

    y_temp = np.zeros((subset_size,1)).ravel()

    for j in range(0,subset_size):
        y_temp[j] = y[i*subset_size+j]

    y_arr.append(y_temp)

# regressor
for i in range(0,q):
    arr_x_toconcat = []
    arr_y_toconcat = []
```

```
for j in range(0,q):
    if i!=j:
        arr_x_toconcat.append(X_arr[j])
        arr_y_toconcat.append(y_arr[j])

X_tr = np.concatenate(arr_x_toconcat)
y_tr = np.concatenate(arr_y_toconcat)
X_test = X_arr[i]
y_test = y_arr[i]

#y_pred = DTRegr(X_tr, y_tr, X_test, DT_depth) # on q subset
#y_pred_internal = DTRegr(X_tr, y_tr, X_tr, DT_depth) # on q-1 subsets

#y_pred = MLPRegr(X_tr, y_tr, X_test, MLP_hidden_layers, MLP_learn_rate,
    MLP_act_func, MLP_alg, MLP_iterations) # on q subset
#y_pred_internal = MLPRegr(X_tr, y_tr, X_tr, MLP_hidden_layers,
    MLP_learn_rate, MLP_act_func, MLP_alg, MLP_iterations) # on q-1 subsets

#y_pred = kNNRegr(X_tr, y_tr, X_test, kNN_neighhbors, kNN_weights_type)
#y_pred_internal = kNNRegr(X_tr, y_tr, X_tr, kNN_neighhbors, kNN_weights_type)

y_pred = SVRegr(X_tr, y_tr, X_test, SVR_kernel, SVR_epsilon, SVR_C)
y_pred_internal = SVRegr(X_tr, y_tr, X_tr, SVR_kernel, SVR_epsilon, SVR_C)

y_pred_arr.append(y_pred)
y_pred_internal_arr.append(y_pred_internal)

MAE.append(mean_absolute_error(y_test, y_pred))
MAE_internal.append(mean_absolute_error(y_test,
    y_pred_internal[:len(y_test)]))

MSE.append(mean_squared_error(y_test, y_pred))
MSE_internal.append(mean_squared_error(y_test, y_pred_internal[:len(y_test)]))

with open(path_out + 'SVR_internal_' + 'q_' + str(q) + '_' + SVR_kernel +
    '_C_' + str(SVR_C) + '_eps_' + str(SVR_epsilon) + '_' + short_name , 'w',
    newline='') as csvfile:
    #with open(path_out + 'kNN_internal_' + 'q_' + str(q) + '_' + 'neigh_' +
        str(kNN_neighhbors) + '_' + kNN_weights_type + '_' + short_name , 'w',
        newline='') as csvfile:
    #with open(path_out + 'DT_internal_' + 'q_' + str(q) + '_' + short_name ,
        'w', newline='') as csvfile:
    #with open(path_out + 'MLP_internal_' + 'q_' + str(q) + '_' + 'hidden_' +
        str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg + '_' +
        'rate_' + str(MLP_learn_rate) + '_' + short_name , 'w', newline='') as
```

```
            csvfile:

        csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                               quotechar=csv.excel.quotechar,
                               quoting=csv.excel.quoting)

        for i in range(0,len(y_pred)):
            csvwriter.writerow([y_pred_internal[i],y_test[i]])

#with open(path_out + 'DT_single_' + 'q_' + str(q) + '_' + short_name , 'w',
    newline='') as csvfile:
#with open(path_out + 'MLP_single_' + 'q_' + str(q) + '_' + 'hidden_' +
    str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg + '_' + 'rate_' +
    str(MLP_learn_rate) + '_' + short_name, 'w', newline='') as csvfile:
#with open(path_out + 'kNN_single_' + 'q_' + str(q) + '_' + 'neigh_' +
    str(kNN_neighhbors) + '_' + kNN_weights_type + '_' + short_name, 'w',
    newline='') as csvfile:
with open(path_out + 'SVR_single_' + 'q_' + str(q) + '_' + SVR_kernel + '_C_' +
    str(SVR_C) + '_eps_' + str(SVR_epsilon) + '_' + short_name , 'w', newline='')
    as csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

    for i in range(0,len(y_pred)):
        csvwriter.writerow([y_pred[i],y_test[i]])

mean_MAE = np.mean(MAE)
mean_MAE_internal = np.mean(MAE_internal)

mean_MSE = np.mean(MSE)
mean_MSE_internal = np.mean(MSE_internal)

#with open(path_out + 'DT_cross_validation_err_' + 'q_' + str(q) + '_' +
    short_name , 'w', newline='') as csvfile:
#with open(path_out + 'kNN_cross_validation_err_' + 'q_' + str(q) + '_' +
    'neigh_' + str(kNN_neighhbors) + '_' + kNN_weights_type + '_' + short_name ,
    'w', newline='') as csvfile:
#with open(path_out + 'MLP_cross_validation_err_' + 'q_' + str(q) + '_' +
    'hidden_' + str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg + '_'
    + 'rate_' + str(MLP_learn_rate) + '_' + short_name , 'w', newline='') as
    csvfile:
with open(path_out + 'SVR_cross_validation_err_' + 'q_' + str(q) + '_' +
    SVR_kernel + '_C_' + str(SVR_C) + '_eps_' + str(SVR_epsilon) + '_' +
```

```
    short_name , 'w', newline='') as csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

        csvwriter.writerow(['MAE', MAE])
        csvwriter.writerow(['averaged MAE', mean_MAE])
        csvwriter.writerow([' '])
        csvwriter.writerow(['MSE', MSE])
        csvwriter.writerow(['averaged MSE', mean_MSE])

print('done')
```

The source code for `cross_validation_3_files.py`:

```
from read_split_data import read_data
from regressors import MLPRegr, SVRegr, kNNRegr, DTRegr
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
import csv

pathfile_in = "D:\\thesis_data\\Means and Percent changes (in)\\"

short_name = "iw_20_ow_10_Bird_2006-2008.csv"

filename_in_1 = "perc_in_20_Bird_2006.csv"
filename_in_2 = "perc_in_20_Bird_2007.csv"
filename_in_3 = "perc_in_20_Bird_2008.csv"

pathfile_out = "D:\\thesis_data\\Percent changes (out)\\"

filename_out_1 = "perc_mean_in_20_out_10_Bird_2006.csv"
filename_out_2 = "perc_mean_in_20_out_10_Bird_2007.csv"
filename_out_3 = "perc_mean_in_20_out_10_Bird_2008.csv"

path_out = "D:\\thesis_data\\Cross-validation\\"

in_wind_width = 20
column_number = 0

DT_depth = 6

kNN_neighhbors = 50
kNN_weights_type = 'uniform'

MLP_hidden_layers = (100,)
MLP_learn_rate = 0.001
MLP_act_func = 'logistic'
MLP_alg = 'sgd'
MLP_iterations = 10000

SVR_kernel = 'linear'
SVR_epsilon = 0.1
SVR_C = 1

q = 10 # number of subsets for cross validation

# get data from one year data set
```

```
X_1, y_1 = read_data(pathfile_in, filename_in_1, pathfile_out, filename_out_1,
    in_wind_width, column_number)

# get data from another year data set
X_2, y_2 = read_data(pathfile_in, filename_in_2, pathfile_out, filename_out_2,
    in_wind_width, column_number)

# get data from one more year data set
X_3, y_3 = read_data(pathfile_in, filename_in_3, pathfile_out, filename_out_3,
    in_wind_width, column_number)

# combine three one-year data sets
X = np.concatenate((X_1, X_2, X_3), axis=0)
y = np.concatenate((y_1, y_2, y_3), axis=0)

subset_size = X.shape[0]//q

X_arr = []
y_arr = []

y_pred_arr = []
y_pred_internal_arr = []

MAE = []
MSE = []
MAE_internal = []
MSE_internal = []

# splitting data into q subsets
for i in range(0,q):
    X_temp = np.zeros((subset_size,X.shape[1]))

    for j in range(0,subset_size):
        for k in range(0,X.shape[1]):
            X_temp[j,k] = X[i*subset_size+j,k]

    X_arr.append(X_temp)

    y_temp = np.zeros((subset_size,1)).ravel()

    for j in range(0,subset_size):
        y_temp[j] = y[i*subset_size+j]

    y_arr.append(y_temp)
```

```
# regressor
for i in range(0,q):
    arr_x_toconcat = []
    arr_y_toconcat = []

    for j in range(0,q):
        if i!=j:
            arr_x_toconcat.append(X_arr[j])
            arr_y_toconcat.append(y_arr[j])

    X_tr = np.concatenate(arr_x_toconcat)
    y_tr = np.concatenate(arr_y_toconcat)
    X_test = X_arr[i]
    y_test = y_arr[i]

    #y_pred = DTRegr(X_tr, y_tr, X_test, DT_depth) # on q subset
    #y_pred_internal = DTRegr(X_tr, y_tr, X_tr, DT_depth) # on q-1 subsets

    #y_pred = kNNRegr(X_tr, y_tr, X_test, kNN_neighhbors, kNN_weights_type)
    #y_pred_internal = kNNRegr(X_tr, y_tr, X_tr, kNN_neighhbors, kNN_weights_type)

    y_pred = SVRegr(X_tr, y_tr, X_test, SVR_kernel, SVR_epsilon, SVR_C)
    y_pred_internal = SVRegr(X_tr, y_tr, X_tr, SVR_kernel, SVR_epsilon, SVR_C)

    y_pred_arr.append(y_pred)
    y_pred_internal_arr.append(y_pred_internal)

    MAE.append(mean_absolute_error(y_test, y_pred))
    MAE_internal.append(mean_absolute_error(y_test,
        y_pred_internal[:len(y_test)]))

    MSE.append(mean_squared_error(y_test, y_pred))
    MSE_internal.append(mean_squared_error(y_test, y_pred_internal[:len(y_test)]))

        #with open(path_out + 'kNN_internal_' + 'q_' + str(q) + '_' + 'neigh_' +
            str(kNN_neighhbors) + '_' + kNN_weights_type + '_' + short_name , 'w',
            newline='') as csvfile:
        #with open(path_out + 'DT_internal_' + 'q_' + str(q) + '_' + short_name ,
            'w', newline='') as csvfile:
        #with open(path_out + 'MLP_internal_' + 'q_' + str(q) + '_' + 'hidden_' +
            str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg + '_' +
            'rate_' + str(MLP_learn_rate) + '_' + short_name , 'w', newline='') as
            csvfile:
        with open(path_out + 'SVR_internal_' + 'q_' + str(q) + '_' + SVR_kernel +
            '_C_' + str(SVR_C) + '_eps_' + str(SVR_epsilon) + '_' + short_name ,
```

```
            'w', newline='') as csvfile:

        csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                               quotechar=csv.excel.quotechar,
                               quoting=csv.excel.quoting)

        for i in range(0,len(y_pred)):
            csvwriter.writerow([y_pred_internal[i],y_test[i]])

        #with open(path_out + 'DT_single_' + 'q_' + str(q) + '_' + short_name ,
            'w', newline='') as csvfile:
        #with open(path_out + 'kNN_single_' + 'q_' + str(q) + '_' + 'neigh_' +
            str(kNN_neighhbors) + '_' + kNN_weights_type + '_' + short_name, 'w',
            newline='') as csvfile:
        #with open(path_out + 'MLP_single_' + 'q_' + str(q) + '_' + 'hidden_' +
            str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg + '_' +
            'rate_' + str(MLP_learn_rate) + '_' + short_name, 'w', newline='') as
            csvfile:
        with open(path_out + 'SVR_single_' + 'q_' + str(q) + '_' + SVR_kernel +
            '_C_' + str(SVR_C) + '_eps_' + str(SVR_epsilon) + '_' + short_name ,
            'w', newline='') as csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

    for i in range(0,len(y_pred)):
        csvwriter.writerow([y_pred[i],y_test[i]])

mean_MAE = np.mean(MAE)
mean_MAE_internal = np.mean(MAE_internal)

mean_MSE = np.mean(MSE)
mean_MSE_internal = np.mean(MSE_internal)

#with open(path_out + 'DT_cross_validation_err_' + 'q_' + str(q) + '_' +
    short_name , 'w', newline='') as csvfile:
with open(path_out + 'SVR_cross_validation_err_' + 'q_' + str(q) + '_' +
    SVR_kernel + '_C_' + str(SVR_C) + '_eps_' + str(SVR_epsilon) + '_' +
    short_name , 'w', newline='') as csvfile:
#with open(path_out + 'kNN_cross_validation_err_' + 'q_' + str(q) + '_' +
    'neigh_' + str(kNN_neighhbors) + '_' + kNN_weights_type + '_' + short_name ,
    'w', newline='') as csvfile:
#with open(path_out + 'MLP_cross_validation_err_' + 'q_' + str(q) + '_' +
    'hidden_' + str(MLP_hidden_layers) + '_' + MLP_act_func + '_' + MLP_alg + '_'
```

```
    + 'rate_' + str(MLP_learn_rate) + '_' + short_name , 'w', newline='') as
    csvfile:

    csvwriter = csv.writer(csvfile, delimiter=csv.excel.delimiter,
                           quotechar=csv.excel.quotechar,
                           quoting=csv.excel.quoting)

        csvwriter.writerow(['MAE', MAE])
        csvwriter.writerow(['averaged MAE', mean_MAE])
        csvwriter.writerow([' '])
        csvwriter.writerow(['MSE', MSE])
        csvwriter.writerow(['averaged MSE', mean_MSE])

print('done')
```

Appendix C

List of files containing experiment results

Files that contain tables with experiment results for all tested regressors:

1. `DT.xlsx`
2. `MLP.xlsx`
3. `kNN.xlsx`
4. `SVR.xlsx`
5. `cross_validation_DT (1-3 years).xlsx`
6. `cross_validation_MLP (1-3 years).xlsx`
7. `cross_validation_kNN (1-3 years).xlsx`
8. `cross_validation_SVR (1 year).xlsx`

Files listed above can be accessed at `https://dl.dropboxusercontent.com/u/29418662/thesis_appendices.zip`.

Bibliography

[1] Scikit-learn 0.17.1 documentation. Support Vector Machines. `http://scikit-learn.org/stable/modules/svm.html#svm-kernels`, 2014.

[2] Scikit-learn 0.17.1 documentation. Decision Trees. `http://scikit-learn.org/stable/modules/tree.html#regression`, April 2016.

[3] Scikit-learn 0.17.1 documentation. Model evaluation: quantifying the quality of predictions. `http://scikit-learn.org/stable/modules/model_evaluation.html#mean-absolute-error`, April 2016.

[4] Scikit-learn 0.17.1 documentation. Nearest Neighbors. `http://scikit-learn.org/stable/modules/neighbors.html#regression`, April 2016.

[5] Scikit-learn 0.18.dev0 documentation. MLPRegressor. `http://scikit-learn.org/dev/modules/generated/sklearn.neural_network.MLPRegressor.html`, April 2016.

[6] Agarwal A. High-frequency trading: Evolution and the future. *Capgemini, London, UK*, 2012.

[7] Nesreen K. Ahmed, Amir F. Atiya, Neamat El Gayar, and Hisham El-Shishiny. An Empirical Comparison of Machine Learning Models for Time Series Forecasting. *Econometric Reviews*, 29(5-6):594–621, August 2010.

[8] Barış Akbaş, Dilek Karahoca, Adem Karahoca, and Ali Güngör. Predicting Newspaper Sales by Using Data Mining Techniques. In *Proceedings of the 15th International Conference on Computer Systems and Technologies*, CompSysTech '14, pages 158–165, New York, NY, USA, 2014. ACM.

[9] H. AL-Rubaiee, R. Qiu, and D. Li. Analysis of the relationship between Saudi twitter posts and the Saudi stock market. In *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 660–665, December 2015.

[10] Ricardo de A Araújo, Adriano L.I. Oliveira, and Silvio R. de L. Meira. A Model with Evolutionary Covariance-based Learning for High-Frequency Financial Forecasting. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 1175–1182, New York, NY, USA, 2015. ACM.

[11] Iyad Batal, Dmitriy Fradkin, James Harrison, Fabian Moerchen, and Milos Hauskrecht. Mining Recent Temporal Patterns for Event Detection in Multivariate Time Series Data. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 280–288, New York, NY, USA, 2012. ACM.

[12] Anouar Ben Mabrouk, Hedi Kortas, and Zouhaier Dhifaoui. A wavelet support vector machine coupled method for time series prediction. *International Journal of Wavelets, Multiresolution and Information Processing*, 6(06):851–868, 2008.

[13] Evangelos Benos and Satchit Sagade. High-frequency trading behaviour and its impact on market quality: evidence from the UK equity market. 2013.

[14] John Black, Nigar Hashimzade, and Gareth Myles. *A Dictionary of Economics*. OUP Oxford, March 2012.

[15] Bevan J. Blair, Ser-Huang Poon, and Stephen J. Taylor. Forecasting S&p 100 volatility: the incremental information content of implied volatilities and high-frequency index returns. In *Handbook of Quantitative Finance and Risk Management*, pages 1333–1344. Springer, 2010.

[16] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1–8, March 2011.

[17] Jonathan Brogaard. High frequency trading and its impact on market quality. *Northwestern University Kellogg School of Management Working Paper*, page 66, 2010.

[18] Jonathan Brogaard, Terrence Hendershott, and Ryan Riordan. High-frequency trading and price discovery. *Review of Financial Studies*, 27(8):2267–2306, 2014.

[19] Lijuan Cao and Qingming Gu. Dynamic support vector machines for non-stationary time series forecasting. *Intelligent Data Analysis*, 6(1):67–83, January 2002.

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

[21] Juan P. Caraça-Valente and Ignacio López-Chavarrías. Discovering Similar Patterns in Time Series. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 497–505, New York, NY, USA, 2000. ACM.

[22] Emilio Castillo, Cristóbal Camarero, Ana Borrego, and Jose Luis Bosque. Financial applications on multi-CPU and multi-GPU architectures. *The Journal of Supercomputing*, 71(2):729–739, November 2014.

[23] Pei-Chann Chang, Chi-Yang Tsai, Chiung-Hua Huang, and Chin-Yuan Fan. Application of a case base reasoning based support vector machine for financial time series data forecasting. In *Emerging Intelligent Computing Technology and Applications. With Aspects of Artificial Intelligence*, pages 294–304. Springer, 2009.

[24] Ray Chen and Marius Lazer. Sentiment analysis of twitter feeds for the prediction of stock market movement. *Stanford Education*, 25:2013, 2013.

[25] Xia Chen, Zhao Yang Dong, Ke Meng, Yan Xu, Kit Po Wong, and H. W. Ngan. Electricity price forecasting with extreme learning machine and bootstrapping. *Power Systems, IEEE Transactions on*, 27(4):2055–2062, 2012.

[26] Yuehui Chen, Bo Yang, and Jiwen Dong. Time-series prediction using a local linear wavelet neural network. *Neurocomputing*, 69(4–6):449–465, January 2006.

[27] Yuehui Chen, Bo Yang, Jiwen Dong, and Ajith Abraham. Time-series forecasting using flexible neural tree model. *Information Sciences*, 174(3–4):219–235, August 2005.

[28] Michael Chlistalla, Bernhard Speyer, Sabine Kaiser, and Thomas Mayer. High-frequency trading. *Deutsche Bank Research*, 7, 2011.

[29] Rohit Choudhry and Kumkum Garg. A hybrid machine learning system for stock market forecasting. *World Academy of Science, Engineering and Technology*, 39(3):315–318, 2008.

[30] Thomas Dimpfl and Stephan Jank. Can Internet Search Queries Help to Predict Stock Market Volatility? *European Financial Management*, 22(2):171–192, March 2016.

[31] David Easley, Marcos M. Lopez De Prado, and Maureen O'Hara. The microstructure of the "flash crash": Flow toxicity, liquidity crashes, and the probability of informed trading. *Journal of Portfolio Management*, 37(2):118, 2011.

[32] M. Ghiassi, H. Saidane, and D. K. Zimbra. A dynamic artificial neural network model for forecasting time series events. *International Journal of Forecasting*, 21(2):341–362, April 2005.

[33] Anton Golub. High Frequency Trading. *Manchester Business School, May*, 5, 2011.

[34] Peter Gomber, Björn Arndt, Marco Lutat, and Tim Uhle. High-Frequency Trading. SSRN Scholarly Paper ID 1858626, Social Science Research Network, Rochester, NY, 2011.

[35] C. Huang, L. l Huang, and T. t Han. Financial time series forecasting based on wavelet kernel support vector machine. In *2012 Eighth International Conference on Natural Computation (ICNC)*, pages 79–83, May 2012.

[36] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1–3):489–501, December 2006.

[37] Wei Huang, Yoshiteru Nakamori, and Shou-Yang Wang. Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, 32(10):2513–2522, 2005.

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

[39] Kyoung-jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1):307–319, 2003.

[40] Kyoung-jae Kim. Artificial neural networks with evolutionary instance selection for financial forecasting. *Expert Systems with Applications*, 30(3):519–526, 2006.

[41] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014.

[42] Andrei A. Kirilenko, Albert S. Kyle, Mehrdad Samadi, and Tugkan Tuzun. The Flash Crash: The Impact of High Frequency Trading on an Electronic Market. SSRN Scholarly Paper ID 1686004, Social Science Research Network, Rochester, NY, December 2015.

[43] V. Kodogiannis and A. Lolis. Forecasting Financial Time Series using Neural Network and Fuzzy System-based Techniques. *Neural Computing & Applications*, 11(2):90–102, October 2002.

[44] Bjoern Krollner, Bruce Vanstone, and Gavin Finnie. Financial time series forecasting with machine learning techniques: A survey. 2010.

[45] Lokesh Kumar, Anvita Pandey, Saakshi Srivastava, and Manuj Darbari. A hybrid machine learning system for stock market forecasting. *Journal of International Technology and Information Management*, 20(1):39–48, 2011.

[46] Robin Kumar and Amandeep Kaur Cheema. GPU Implementation of a Deep Learning Network for Financial Prediction. *The International Journal of Science and Technoledge*, 2(5):374–378, May 2014.

[47] Guang Li, Chen-Ching Liu, Chris Mattson, and Jacques Lawarrée. Day-ahead electricity price forecasting in a grid environment. *Power Systems, IEEE Transactions on*, 22(1):266–274, 2007.

[48] Rong-Jun Li and Zhi-Bin Xiong. Forecasting stock market with fuzzy neural networks. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 6, pages 3475–3479. IEEE, 2005.

[49] Wei-Min Li, Jian-Wei Liu, Jia-Jin Le, and Xiang-Rong Wang. The financial time series forecasting based on proposed ARMA-GRNN model. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 4. IEEE, 2005.

[50] Fabrizio Lillo, Salvatore Miccichè, Michele Tumminello, Jyrki Piilo, and Rosario N. Mantegna. How news affects the trading behaviour of different categories of investors in a financial market. *Quantitative Finance*, 15(2):213–229, February 2015.

[51] Jian-Yi Lin, Chun-Tian Cheng, and Kwok-Wing Chau. Using support vector machines for long-term discharge prediction. *Hydrological Sciences Journal*, 51(4):599–612, August 2006.

[52] Tom C. W. Lin. The New Financial Industry. SSRN Scholarly Paper ID 2417988, Social Science Research Network, Rochester, NY, March 2014.

[53] Chi-Jie Lu, Tian-Shyug Lee, and Chih-Chou Chiu. Financial time series forecasting using independent component analysis and support vector regression. *Decision Support Systems*, 47(2):115–125, 2009.

[54] Rosario N. Mantegna and H. Eugene Stanley. *Introduction to econophysics: correlations and complexity in finance*. Cambridge university press, 1999.

[55] Martin Martens. Measuring and forecasting S&p 500 index-futures volatility using high-frequency data. *Journal of Futures Markets*, 22(6):497–518, 2002.

[56] Martin Martens and Jason Zein. Predicting financial volatility: high-frequency time-series forecasts vis-a-vis implied volatility. *Available at SSRN 301382*, 2002.

[57] Geoffrey McLachlan, Kim-Anh Do, and Christophe Ambroise. *Analyzing Microarray Gene Expression Data*. John Wiley & Sons, March 2005.

[58] Helen Susannah Moat, Chester Curme, H. Eugene Stanley, and Tobias Preis. Anticipating Stock Market Movements with Google and Wikipedia. In Davron Matrasulov and H. Eugene Stanley, editors, *Nonlinear Phenomena in Complex Systems: From Nano to Macro Scale*, NATO Science for Peace and Security Series C: Environmental Security, pages 47–59. Springer Netherlands, 2014.

[59] Abdullah Mueen, Eamonn J. Keogh, Qiang Zhu, Sydney Cash, and M. Brandon Westover. Exact Discovery of Time Series Motifs. In *SDM*, pages 473–484. SIAM, 2009.

[60] F.J. Nogales, J. Contreras, A.J. Conejo, and R. Espinola. Forecasting next-day electricity prices by time series models. *IEEE Transactions on Power Systems*, 17(2):342–348, May 2002.

[61] J. O. Omeru and D. Thomas. A GPU accelerated hybrid lattice-grid algorithm for options pricing. In *2014 International Conference on High Performance Computing Simulation (HPCS)*, pages 758–765, July 2014.

[62] Spiros Papadimitriou and Philip Yu. Optimal Multi-scale Patterns in Time Series Streams. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 647–658, New York, NY, USA, 2006. ACM.

[63] Richard J. Povinelli. Identifying Temporal Patterns for Characterization and Prediction of Financial Time Series Events. In John F. Roddick and Kathleen Hornsby, editors, *Temporal, Spatial, and Spatio-Temporal Data Mining*, number 2007 in Lecture Notes in Computer Science, pages 46–61. Springer Berlin Heidelberg, 2001.

[64] Tobias Preis, Helen Susannah Moat, and H. Eugene Stanley. Quantifying Trading Behavior in Financial Markets Using Google Trends. *Scientific Reports*, 3, April 2013.

[65] Gabriele Ranco, Darko Aleksovski, Guido Caldarelli, Miha Grčar, and Igor Mozetič. The Effects of Twitter Sentiment on Stock Price Returns. *PLOS ONE*, 10(9):e0138441, September 2015.

[66] Thierry Rijper, Willem Sprenkeler, and Stefan Kip. High frequency trading. *Position Paper. Optiver Holding BV, Strawinskylaan*, 3095:1077, 2010.

[67] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[68] Nicholas Sapankevych, Ravi Sankar, and others. Time series prediction using support vector machines: a survey. *Computational Intelligence Magazine, IEEE*, 4(2):24–38, 2009.

[69] Jianfeng Si, Arjun Mukherjee, Bing Liu, Qing Li, Huayi Li, and Xiaotie Deng. Exploiting Topic based Twitter Sentiment for Stock Prediction. *ACL (2)*, 2013:24–29, 2013.

[70] Rampal Singh and S. Balasundaram. Application of extreme learning machine method for time series analysis. *International Journal of Intelligent Technology*, 2(4):256–262, 2007.

[71] Jasmina Smailović, Miha Grčar, Nada Lavrač, and Martin Žnidaršič. Predictive Sentiment Analysis of Tweets: A Stock Market Application. In Andreas Holzinger and Gabriella Pasi, editors, *Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data*, number 7947 in Lecture Notes in Computer Science, pages 77–88. Springer Berlin Heidelberg, 2013.

[72] Lee A. Smales. News sentiment in the gold futures market. *Journal of Banking & Finance*, 49:275–286, December 2014.

[73] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, August 2004.

[74] Antti Sorjamaa, Jin Hao, Nima Reyhani, Yongnan Ji, and Amaury Lendasse. Methodology for long-term prediction of time series. *Neurocomputing*, 70(16–18):2861–2869, October 2007.

[75] Stephan Spiegel, Brijnesh Johannes Jain, Ernesto William De Luca, and Sahin Albayrak. Pattern Recognition in Multivariate Time Series: Dissertation Proposal. In *Proceedings of the 4th Workshop on Workshop for Ph.D. Students in Information & Knowledge Management*, PIKM '11, pages 27–34, New York, NY, USA, 2011. ACM.

[76] Xiangyu Tang, Chunyu Yang, and Jie Zhou. Stock Price Forecasting by Combining News Mining and Time Series Analysis. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '09, pages 279–282, Washington, DC, USA, 2009. IEEE Computer Society.

[77] Francis E. H. Tay and L. J. Cao. Modified support vector machines in financial time series forecasting. *Neurocomputing*, 48(1–4):847–861, October 2002.

[78] Francis E.H. Tay and Lijuan Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309–317, 2001.

[79] Francis Eng Hock Tay and Li Juan Cao. Improved financial time series forecasting by combining Support Vector Machines with self-organizing feature map. *Intelligent Data Analysis*, 5(4):339–354, January 2001.

[80] Sergios Theodoridis and Konstantinos Koutroumbas. Pattern recognition. `http://cds.cern.ch/record/407577`, 1998.

[81] Sergios Theodoridis, Aggelos Pikrakis, Konstantinos Koutroumbas, and Dionisis Cavouras. *Introduction to Pattern Recognition: A Matlab Approach*. Academic Press, March 2010.

[82] U Thissen, R van Brakel, A. P de Weijer, W. J Melssen, and L. M. C Buydens. Using support vector machines for time series prediction. *Chemometrics and Intelligent Laboratory Systems*, 69(1–2):35–49, November 2003.

[83] T. Van Gestel, J.A.K. Suykens, D.-E. Baestaens, A. Lambrechts, G. Lanckriet, B. Vandaele, B. De Moor, and J. Vandewalle. Financial time series prediction using least squares support vector machines within the evidence framework. *IEEE Transactions on Neural Networks*, 12(4):809–821, July 2001.

[84] Steven Walczak. An empirical analysis of data requirements for financial forecasting with neural networks. *Journal of management information systems*, 17(4):203–222, 2001.

[85] Max Welling. Support vector regression. *Department of Computer Science, University of Toronto, Toronto (Kanada)*, 2004.

[86] Qiong Wu, Wen-ying Liu, and Yi-han Yang. Time series online prediction algorithm based on least squares support vector machine. *Journal of Central South University of Technology*, 14:442–446, 2007.

[87] Weizhong Yan. Toward Automatic Time-Series Forecasting Using Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 23(7):1028–1039, July 2012.

[88] Jin-Fang Yang, Yong-Jie Zhai, Da-ping Xu, and Pu Han. SMO Algorithm Applied in Time Series Model Building and Forecast. In *2007 International Conference on Machine Learning and Cybernetics*, volume 4, pages 2395–2400, August 2007.

[89] Lean Yu, Shouyang Wang, and Kin Keung Lai. A neural-network-based nonlinear metamodeling approach to financial time series forecasting. *Applied Soft Computing*, 9(2):563–574, March 2009.

[90] Mohammadsaleh Zakerinia and Seyed Farid Ghaderi. Short Term Wind Power Forecasting Using Time Series Neural Networks. In *Proceedings of the 2011 Emerging M&S Applications in Industry and Academia Symposium*, EAIA '11, pages 17–22, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[91] G. Peter Zhang. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50:159–175, January 2003.

[92] Linhao Zhang. *Sentiment analysis on Twitter with stock price and significant keyword correlation.* PhD thesis, 2013.