FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Master's Thesis

Christian Birch Okkels

Financial Forecasting

Stock Market Prediction

Supervisor: Bjarne Andresen

Submitted: 01/06/2014

Abstract

This thesis considers financial forecasting, and more specifically the problem of stock market prediction. Beginning with an introduction to financial markets, and their data and analysis, we proceed by discussing the fundamental question of their predictability, including the Efficient Market Hypothesis. The question is further elucidated through a conceptual and methodological comparison to forecasting in physics, e.g. weather forecasting, etc. Inspired by oscillatory systems in physics, we also build a model for the fluctuation of stock prices.

The main financial asset under consideration is the popular NASDAQ Index. Analysing the distribution of returns, we find that the daily returns are not normally distributed—as is often assumed—and that a better model accounting for the heavier tails is provided by the t distribution. In a time series analysis, we find empirical evidence for autocorrelation and conditional heteroscedasticity (volatility clustering). This makes it plausible to apply ARMA-GARCH models; we construct, select, and diagnose such models and use them for both multi-step and one-step ahead prediction.

Used even more extensively are the three machine learning models, Artificial Neural Networks, Support Vector Machines, and Random Forests. In several model-specific analyses we explore the effects of varying parameters, showing among other things the crucial problem of overfitting. We reduce this phenomenon via cross-validation. In numerous experiments we then investigate the effect of varying certain aspects of the data: e.g. the amount of lags, exogenous inputs (mostly technical indicators), training observations, and the frequency (daily, weekly, monthly), etc. Prediction performance is assessed quantitatively via the MSE, MAPE, and Hit Rate measures, and qualitatively using plots.

Our results show excellent performances, with one-day ahead predictions very close to the actual prices. In the best cases, the models correctly predict the price direction (up/down) almost 80% of the time. For comparison, we have implemented two benchmarks: the Random Walk model ("tomorrow's price equals today's"), and the Random Guess model ("50/50"). Our models significantly outperform the latter in all regards and the former in terms of MSE and MAPE. In terms of Hit Rate, however, the Random Walk model comes quite close. This somewhat reduces the impressiveness of the results, and helps to highlight a crucial feature in the predictions: that successful predictions are often due to the price moving in a particular direction, and that failures mainly occur when the price suddenly makes a larger move in the opposite direction. Thus, although the models exhibit some degree of success, they seem to have great difficulty predicting sudden day-to-day corrections. This result—the difficulty in, or even impossibility of, prediction—may be ascribed to suboptimal predictors and models, or to the fact that the financial markets may actually be unpredictable.

Contents

Chapter 1

Introduction

Financial forecasting is a broad discipline with many different facets and subgroups; depending on the part of finance involved, it may refer to e.g. business valuation, estimating a company's future revenue, the future financial state of a country, the risk associated with an investment, etc. In this thesis we focus on the problem of stock market prediction; that is, the prediction of future prices of assets traded on the worldwide stock exchanges. As the main aspects of the study, we investigate the predictability of the markets and perform a comparative analysis of selected prediction models.

As with many disciplines in present day, financial forecasting—and especially the type considered here—is an interdisciplinary field combining both physics, statistics, finance, mathematics, economics, and computer science. The interdisciplinary aspect becomes even more pronounced when looking at the prediction models we will employ in this study, including technical and fundamental analysis from finance, conditional mean and variance models from traditional time series analysis and econometrics, statistical machine learning methods commonly applied in physics and computer science, as well as a physics-inspired oscillator model.

1.1 Structure of the Thesis

The thesis is organized as follows. The remaining sections in Chapter 1 state our purposes of the study and give a historical overview of some previous results in stock market prediction.

Chapter 2 describes the financial markets. We raise the fundamental question of their predictability and discuss different opinions and beliefs. In this regard, we also discuss how the markets have changed throughout time, especially with the advances in computer technology, and how these changes may affect the question of predictability. We then list the different approaches to stock analysis and prediction in general, as well as the particular selection of methods chosen here. Then we make a detailed description of financial data; its acquisition and availability, its different types and forms, its use for calculating additional data, and the ways in which it should be pre-processed and transformed for some of our methods to better understand it. Finally, we describe the financial disciplines of technical and fundamental analysis, and how these can be utilized for prediction on their own as well as in combination with other methods.

Chapter 3 deals with forecasting in physics. We discuss several fields in physics where forecasting and prediction is relevant—and in some cases even of vital importance—including weather, sunspot activity, climate change, and natural disasters such as earthquakes and volcanic eruptions. In each case we discuss the similarities to financial forecasting, while also raising the crucial differences between forecasting natural phenomena such as the weather and human-created con-

structs such as stock markets. Afterwards, we also show how some problems in finance and physics are deeply connected and almost identical (e.g. option pricing). Inspired by the differential equation framework commonly used in both disciplines, and in particular by oscillatory systems in physics, we build an oscillator model for the fluctuation of stock prices and argue for the plausibility of the underlying assumptions.

Chapter 4 introduces the necessary theory of traditional time series analysis. We go through such concepts as stationarity, autocorrelation, and conditional heteroscedasticity. These basics are then used to describe the theory behind the simple autoregressive and moving average models. These simple cases lay the groundwork for the more advanced combined models and the composite conditional mean and variance models, which is what we will ultimately use for prediction.

Chapter 5 provides the theory behind our selection of statistical machine learning methods; Artificial Neural Networks, Support Vector Machines, and Random Forests. In each case we describe the relevant theory, the strengths, weaknesses and problems, and how the method is applied to stock market prediction.

Chapter 6 contains all of our analyses and results. Our main focus is the popular NASDAQ Index, and we start out with a distribution analysis in which we compute statistical properties and visually explore the dataset. We then proceed with a time series analysis where we investigate the presence of autocorrelation and conditional heteroscedasticity (volatility clustering). Afterwards, we apply the conditional mean and variance models for predicting the future prices and trends in the NASDAQ Index. We employ information criteria-based model selection to find the optimal parameters, and assess the prediction performance via different error and performance measures. We then turn our attention to the machine learning methods. To begin with, we explore the workings of the models and analyse how their prediction performances vary as we change certain parameters. This gives a better feel for the methods and at the same time showcases one of the most important problems in machine learning—overfitting[1]. After this, we proceed with numerous experiments. We investigate how well the models perform in one-day ahead prediction of stock prices. In the different experiments we also analyse the effects of varying e.g.: the amount of lagged data (both response and external) used to predict the future price; the exogenous inputs, especially different technical indicators; the amount of data used to train and build the prediction models; etc.

Chapter 7 is the conclusion where we summarize the study as a whole and list our main results. We also discuss topics for future work and research.

Finally, the Appendix contains a description of some technical indicators and their interpretation, as well as a description of the MATLAB programs we have written as part of the thesis.

1.2 Purposes of the Thesis

Drawing inspiration from a multitude of disciplines, each with a vast amount of different approaches to the intricate goal of financial forecasting, a study like this can be pulled in an endless number of directions. The limited size of this thesis naturally means that we have to narrow down the scope to some selected areas of focus. More specifically, we define the purposes of the study as follows:

- Obtain and present an overview of similar research in stock market prediction through an examination of the available literature.

[1]When the model memorizes the patterns in the data used to train it, and thus fails to be able to generalize to unseen data.

- Describe the financial markets, their predictability, the available data, and the disciplines of fundamental and technical analysis. Furthermore, discuss the evolution through time of the markets and how this may have changed their psychology and predictability.
- Discuss areas in physics where forecasting plays an important role, and compare them to financial forecasting. Raise interesting conceptual and methodological similarities as well as fundamental, crucial differences.
- Examine and review relevant aspects of traditional time series analysis.
- Describe the relevant theory of selected statistical machine learning methods—Artificial Neural Networks, Support Vector Machines, and Random Forests—and discuss their general strengths, weaknesses and problems, as well as their application to financial forecasting.
- Write several computer programs (in MATLAB) for the analysis and prediction of stocks and other assets.
- Analyse quantitatively and qualitatively the distribution of returns for a selected financial asset (the NASDAQ Index). Confirm or refute the common assumption of a Gaussian model and, in the latter case, assess whether another distribution provides a better fit.
- Perform a time series analysis of the NASDAQ Index, with a focus on determining both quantitatively and qualitatively the presence or absence of autocorrelation and conditional heteroscedasticity.
- Construct, optimize, and diagnose conditional mean and variance models for the daily NASDAQ Index data, and compare the forecasting performances for a number of different models.
- Using as predictors lagged response data and technical indicators, train and test Artificial Neural Networks, Support Vector Machines, and Random Forests on the daily NASDAQ data, with a focus on investigating the effects of varying the model parameters and settings.
- In numerous experiments, analyse the performance effects of changing quantities including e.g. the type and amount of exogenous inputs, the number of lags to use for prediction, and the amount and frequency of the data, etc.
- Compare the performances for the different models to assess which is the best prediction method. Furthermore, compare the results to those obtained by some benchmarks to assess the question of the predictability of stock markets.

1.3 Historical Overview of the Literature

This section gives an overview of some of the relevant literature in the field of financial forecasting. The main points of interest are the results and accuracies obtained by other researchers, using the same or different methods, in predicting the stock market. The results of our investigation are presented in Table 1.3 below, listing the year, the title, and the author(s) of the particular research articles as well as a short description and the main conclusions and results.

Table 1.1: Historical overview of the literature in the field of financial forecasting.

Year	Title and author(s)	Description	Conclusions and results
2000	Financial Time Series Forecasting by Neural Network Using Conjugate Gradient Learning Algorithm and Multiple Linear Regression Weight Initialization, Man-Chung, Chi-Cheong, Chi-Chung [6].	A neural network using a conjugate gradient (CG) learning algorithm and multiple linear regression weight initialization (MLRI) is compared to the standard case of a neural network using steepest descent (SD) and random weight initialization (RI). Data: daily quotes of eleven stocks on the Shanghai Stock Exchange. Inputs: Ten technical indicators, including lagged values of the exponential moving average. All values normalized to $[0.05, 0.95]$. Training/test set split: 500/150 (650 in total) $\approx$ 75%/25%. Performance measures: Mean Squared Error (MSE) and hit ratio.	Best hit ratios: CG + RI: 73% CG + MLRI: 73.5% SD + RI: 72.5% SD + MLRI: 69% Although the hit rates are rather close, the CG learning algorithm was roughly ten times faster than SD.
2001	Financial Forecasting using Support Vector Machines, Cao & Tay [24]	A comparison of SVMs using Gaussian kernels to a multi-layer perceptron (MLP) trained by the back propagation (BP) algorithm. Data: daily quotes of the S&P 500 stock index pre-processed into Relative Differences in Percentages (RDP). Inputs: (1) lagged RDP values and (2) technical indicators: MACD, OBV, and Volatility. Training/test set split: 500/200 (700 in total) $\approx$ 70%/30%. Performance measures: Normalized Mean Squared Error (NMSE), Mean Absolute Error (MAE), Directional Symmetry (DS), and Correct Up/Down Trend (CP/CD).	SVM outperforms BP networks in each performance measure. Hit rates (successful predictions of up/down movements) on test set: SVM: 46% BP: 40% Compared to BP networks, SVMs have fewer free parameters, train faster, and potentially reach global minima (whereas BP networks easily get stuck in local minima).
2001	An Empirical Analysis of Data Requirements for Financial Forecasting with Neural Networks, S. Walczak [15]	Examination of the impact of training data sample size on ANN performance. Data: daily data of the US Dollar/British Pound exchange rate. Inputs: prices with lags of 1, 2, and 5. Training/test set split: Different total sample sizes are examined, from 1 year to 21.75 years in one- or two-year steps. Test set size is 125 observations. Performance measures: Direction of change (hit rate).	ANNs using a suitable amount of *recent* historical data outperform (with $\sim$ 60% hit rate) ANNs that use even larger, older training sets (with $<$ 60%). Reduced training set size also reduce development cost and time. The ANNs with best performances used training sets comprising daily data for 1 or 2 years. However, the optimal data samples sizes vary with application and the particular asset under examination.

Continued on next page

Table 1.1 – *Continued from previous page*

Year	Title and author(s)	Description	Conclusions and results
2002	A Comparison of Neural Networks with Time Series Models for Forecasting Returns on a Stock Market, J. Yim [14]	A comparison of feedforward BP ANNs with ARMA-GARCH, ARMA-EGARCH, and structural time series (STS) models. Data: daily quotes and logarithmic returns of the IBOVESPA index on the Sao Paolo Stock Exchange. Inputs: logreturns with lags 1 and 9, and GARCH volatility. Training/test set split: 948/40 (988 in total) $\approx$ 96%/4%. Performance measures: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE)	ANNs are superior to ARCH-GARCH, ARMA-EGARCH, and STS. Volatility derived from ARMA-GARCH is useful as an input to ANNs.
2003	Financial Time Series Forecasting using Support Vector Machines, Kim [25]	SVMs with Gaussian (and polynomial) kernels are compared to BP networks and case-based reasoning (CBR) for predicting the direction of change. Data: daily quotes of the Korean stock price index, KOSPI, linearly scaled to $[-1, 1]$. Inputs: 12 technical indicators. Training/test set split: 2347/581 (2928 in total) $\approx$ 80%/20%. Performance measures: hit ratio (successful up/down predictions in %)	SVM outperforms both BP and CBR. Best hit rates obtained: SVM: 58% BP: 55% CBR: 52% The SVM performance depends on the cost parameter C and the kernel parameter γ.
2005	Forecasting Stock Market Movement Direction with Support Vector Machines, Huang, Nakamori, & Wang [26]	SVMs with Gaussian kernels are compared to Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Elman Backpropagation Neural Networks (EBNN), and the Random Walk (RW) model (from the EMH) for predicting the movement direction. Target data: weekly changes of the Nikkei 225 stock index. Inputs: S&P 500 index and USD-JPY exchange rate, pre-processed through logarithmic transformations. Training/test set split: 640/36 (676 in total) $\approx$ 95%/5%. Performance measures: hit ratio. A hybrid/expert model was also made by combining various methods.	SVM outperforms LDA, QDA, EBNN, and RW. Hit ratios: RW: 50% LDA: 55% QDA: 69% EBNN: 69% SVM: 73% Expert model: 75% SVM is strong in itself due to its ability to reach global minima. Even better results can be obtained by combining methods.
2005	Mining Stock Market Tendency using GA-based Support Vector Machines, Yu, Wang, & Lai [27]	SVMs using Genetic Algorithms for parameter determination and input feature selection are compared to Random Walk (RW), Autoregressive Integrated Moving Average (ARIMA), Backpropagation neural networks (BP), and normal SVMs for the prediction of movement direction. Data: daily quotes of the S&P 500 stock index. Inputs: 18 technical indicators. Training/test set split: 800/200 (1000 in total) $\approx$ 80%/20%. Performance measures: hit ratio.	SVM outperforms LDA, QDA, EBNN, and RW. Hit ratios: RW: 51% ARIMA: 56% BP: 70% SVM: 79% GA-SVM: 85% GA-SVM not only performs better, but also computes faster than BP and SVM.

Continued on next page

Table 1.1 – *Continued from previous page*

Year	Title and author(s)	Description	Conclusions and results
2007	Forecasting Financial Time Series with Support Vector Machines Based on Dynamic Kernels, Mager, Paasche, & Sick [28]	Different types of SVMs with dynamic kernels are compared to each other and to a naïve forecasting method for the prediction of trends and movement direction. Data: daily quotes of the FDAX stock index future and the FGBL German government bond future, pre-processed into rates of change (ROC). Inputs: open, high, low, and closing prices as well as lagged ROC values. Training/test set split: 750/250 (1000 in total) $\approx$ 75%/25%. Performance measures: Mean Absolute Scaled Error (MASE) and hit ratio.	ν-SVR outperforms the other SVM variants and the nave forecast. Best hit ratios: Naïve forecast: 48% ν-SVR: 83% SVMs with dynamic kernels are particularly good for time series forecasting. More input features and patterns do not necessarily increase performance.
2007	Stock Price Forecasting using PSO-trained Neural Networks, Junyou [7]	A feedforward neural network using particle swarm optimizaton (PSO) for learning is compared to a normal backpropagation (BP) neural network. Data: daily quotes of the Singapore stock market index, scaled to $[0.1, 0.9]$. Inputs: lagged prices. Training/test set split: 506/252 (758 in total) $\approx$ 67%/33%. Performance measures: Mean Absolute Percentage Error (MAPE).	MAPE results (lower are better): PSO: 0.75 BP: 0.91 Using PSO is not only faster but also yields better results than the usual BP learning algorithm.
N/A	Forecasting of Indian Stock Market Index Using Artificial Neural Network, Majumder & Hussian, [18]	An examination of neural networks with different structures, inputs, and training algorithms. Data: daily closing prices (pre-processed into logarithmic returns) of the S&P CNX Nifty 50 index. Inputs: lagged response data. Training/test set split: N/A. Performance measures: Direction of movement (hit rate).	Accuracy / hit rates: Best: $\sim$ 89%. Average: $\sim$ 70%.
2009	Forecasting Model for Crude Oil Price Using Artificial Neural Networks and Commodity Futures Prices, S. Kulkarni, I. Haidar [17]	Multi-layer feedforward NN Data: daily quotes of crude oil spot prices. Inputs: lagged response data and futures prices as exogenous input (also with lags). Training/test set split: $\approx$ 70%/30%. Performance measures: Price direction accuracy (hit rate).	Accuracy / hit rates: 1-day ahead: 78%. 2-day ahead: 66%. 3-day ahead: 53%.
2012	Comparative Study of Static and Dynamic Neural Network Models for Nonlinear Time Series Forecasting, Abounoori et al. [16]	Comparison between two static networks (Adaptive Neuro-Fuzzy Inference Systems (ANFIS) and Multi-layer Feedforward NN (MFNN)) and one dynamic model (Nonlinear Neural Network Autoregressive model (NNAR)). Data: returns of the Tehran Stock Exchange (TSE) index. Inputs: response data with lags from 1 to 5 (for ANFIS and MFNN) and 1 to 10 (for NNAR). Training/test set split: $\approx$ 70%/30%. Performance measures: MSE and RMSE.	ANFIS has a higher forecasting ability (lower MSE and RMSE) than MFNN and NNAR.

Chapter 2

Financial Markets and Data

This chapter concerns the financial markets. The term refers to the worldwide exchanges on which both private investors and professional corporations alike can buy, sell, and trade stocks, commodities, currency, and countless other types of financial instruments.

We begin by discussing the fundamental question of whether or not the markets can be predicted. Later, we delve into the data related to financial markets and stocks in particular. We describe how the data can be acquired, how the raw version can be used to compute additional types of data, and how it should be pre-processed and transformed before being used for prediction. We also describe technical and fundamental analysis and how we have made use of these fields and their related data.

2.1 Predictability of the Markets

The prediction of stocks, and financial markets in general, is a delicate topic, subject of much debate. The fundamental question is whether stocks and markets can be predicted at all. There are different opinions in this regard; some say that it is possible, some say that it is possible to some extent, and others believe that it is wholly impossible. In this regard, there exists something called the *Efficient Market Hypothesis*.

2.1.1 The Efficient Market Hypothesis

The Efficient Market Hypothesis (EMH) has three forms of varying strength:

- The *weak* form of the EMH claims that stock prices already reflect all past publicly available information. Future prices cannot be predicted by analysing prices from the past, rendering it impossible to consistently produce excess returns using technical analysis (though some forms of fundamental analysis can still work consistently). There are no serial dependencies between share prices, meaning that there are no "patterns"; future price movements are determined entirely by information not contained in the price series.
- The *semi-strong* EMH incorporates the weak form and further says that prices instantly change to reflect new public information. Thus, no excess returns can be made by trading on this new information. This implies that neither technical analysis nor fundamental analysis will be able to consistently produce excess returns.

- The *strong* EMH contains both of the weaker forms and further states that prices instantly reflect even hidden or "insider" information. It is thus wholly impossible for anyone to earn excess returns.

From the three forms above, the basic idea behind the EMH is that stocks (and other traded assets) always trade at their fair price, which entails that it is impossible to outperform the overall market, and, relevant for our case, that it is pointless to try to predict trends. Thus, with our "humble" goal of predicting the stock markets, successful predictions will support a refutation of the Efficient Market Hypothesis (at least the weak form), whereas failure and poor predictions will support the EMH itself.

Random Walk Benchmark Model

Later, in our experiments in Section 6.2, we employ different models for stock market prediction. To assess the performance of these models we compare them to each other. But to see how well these models fare overall, and to assess the question of the predictability of the markets, we also compare them to some benchmarks. We call one of these benchmarks the Random Walk model, or the "tomorrow equals today" model, and it has close ties to the weak form of the EMH discussed above.

The model's prediction for tomorrow's stock price is just today's price. As such, it says that stock prices are a *martingale*. A martingale is a sequence of random numbers $y_0, y_1, \ldots$ with finite means such that the conditional expectation of y_{n+1} given $y_0, y_1, \ldots, y_n$ is equal to y_n, i.e.,[1]

$$E[y_{n+1}|y_0, y_1, \ldots, y_n] = y_n. \tag{2.1}$$

If y_i denotes a stock price at time i, then this equation says that the expected value of next period's stock price y_{n+1} is equal to the current period's stock price y_n.

Now enters the "Random Walk" part of the model. An example of a martingale is a one-dimensional Random Walk with steps equally likely in either direction. A Random Walk is basically just a series of a number of random steps.[2] Stock price fluctuations may be modelled in this way; for example, the Black-Scholes formula for option pricing uses a Gaussian Random Walk to model the price of the underlying stock.[3] For stocks, equally likely steps in either direction entails that the price is equally likely to increase or decrease, which is quite plausible if one assumes random price fluctuations. If stock prices follow a Random Walk, then we cannot hope predict them, which is in line with the EMH.

We now connect the Random Walk, martingale prices, and the EMH. First, modelling stock prices as a Random Walk as explained above means that they are a martingale. And to say that stock prices are a martingale is essentially to say that they are weak-form efficient. To see this, recall that the weak form of the EMH says that knowledge of all past prices is not informative regarding the expectation of future prices. A martingale is a special case of weak-form efficiency which says that the expected future price next period is equal to the current price.

The Random Walk model thus provides a highly interesting benchmark that our other models can be compared against. If our other models (ARMA-GARCH, Neural Networks, Support Vector Machines, and Random Forests) yield significantly better prediction performances than

[1] [85]

[2] The random walk also has plenty of ties to e.g. physics. First, a random walk with very small steps provides an approximation to a so-called Wiener process, which is a stochastic process with similar behaviour to Brownian motion. And Brownian motion is exactly the physical phenomenon describing the random motion of a particle in a fluid (liquid or gas) resulting from its collision with atoms or molecules in the gas or liquid.

[3] We discuss option pricing a bit more in Section 3.3. See also [48].

the Random Walk model, then we have empirical evidence that the stock markets may be predicted to some extent. On the other hand, if our models fail to outperform the Random Walk benchmark, then this may be viewed as support for the Efficient Market Hypothesis and the unpredictability of the markets.

2.1.2 Human Psychology vs. "Computer" Psychology

The previous sections discussed the efficiency of the markets through the EMH. One might also argue against the predictability of the markets from another, perhaps more intuitive point of view; namely, the nature of financial markets and the human psychology.

Weather and earthquakes, planet orbits and particle interactions, and physical reactions in general are all natural phenomena. We can predict these to some extent. Equity, stocks, currency, etc., on the other hand, and the financial markets on which these are traded, are all human constructs. As such, they are largely dominated by human psychology; our opinions of the people sitting on the board of a company, our expectations for the future earnings, etc. The human mind is complex, and our behaviour varying and unpredictable—sometimes erratic, but always influenced by feelings and emotions. Hence, it may be a vain hope trying to predict our actions—especially when something like money is involved, as is certainly the case for financial markets.

In recent years, however, with the advent of new technology and rapidly increasing computing power, automated trading systems are seeing more and more use. These are computer-programmed systems that trade after certain rules or algorithms. With the increase in such systems, the role of human psychology on the financial markets may be less influential—replaced, instead, by some sort of "computer psychology." Instead of clicking the "buy" or "sell" button with our own nervously shaking fingers, the computer—without a moment's hesitation—takes the decision for us. A decision made on the basis of certain patterns in the data, including e.g. technical indicators, etc. Thus, with more and more people—and, indeed, the large investment banks—using automated trading systems, some of the signals and patterns observed in the markets almost become self-fulfilling prophecies. For example, assume that a lot of traders (humans or their automated computer systems) use a particular trading system—based e.g. on one or more technical indicators—and these indicators show that a certain stock is oversold and a trend reversal is imminent. Then, if the traders collectively act on this by buying said stock, the stock may actually rise and do as the trading algorithm predicted. This "herd"-like behaviour of many market agents acting the same can thus cause computerized, automated trading systems and algorithms to be self-fulfilling prophecies. As a whole, this technological development, or evolution, may to some extent increase the plausibility of and possibility of predicting the markets.

2.1.3 Overview of Methods

The discussions above of the Efficient Market Hypothesis, human psychology, and the technological advances in our modern world all deal with the question asking *if* the financial markets can be predicted. Assuming, of course, that they to some extent can be predicted[4], the question now arises of *how*. There are various ways to go about this, but they can generally be summarized by the following four major classes:

- *Technical Analysis* attempts to predict trends and future prices by means of various technical indicators such as moving averages, momentum, trend lines, pattern analysis, etc. It

[4] "Prediction" here is a loose term, since we do not at all expect, or aim to, predict the exact values of future stock prices; rather, we are more interested in trying to determine the general trends, whether the stock will increase or decrease, etc.

is used mainly for investments with shorter time horizons.

- *Fundamental Analysis* uses an in-depth analysis of a company's performance and growth prospect compared to the overall economic conditions. It is used mainly for long-term investment decisions and strategies.
- *Time Series Forecasting* analyses past data to estimate future values. This is done by modelling a non-linear function by a recurrence relation, the parameters of which are derived using past data. This recurrence relation is then used to estimate future, unknown prices.
- *Machine Learning* trains a computer to learn patterns and trends in data. This is done by establishing a model based on some training data, and then facing it with unseen data to test its generalization ability and performance.

The approach pursued in this study uses ideas, aspects, and methods from all of the above disciplines and more, making us span an array of fields including physics, statistics, computer science, mathematics, finance, and economics. More specifically, we will use technical and fundamental data as input in prediction models taken from time series analysis and machine learning. Selected in part on the basis of similar research as well as our own experience and curiosity in the field, the methods and theoretical frameworks employed here are:

- Technical and Fundamental Analysis[5]
- Conditional mean and variance (ARMA-GARCH) models
- Artificial Neural Networks
- Support Vector Machines
- Random Forests
- Oscillatory systems

2.2 Financial Data

A truly massive amount of financial data exists, and the number just keeps increasing with every passing day on the financial markets. Everything is recorded somewhere, from huge-scale company buyouts to tiny personal transactions on the stock exchanges, changes in both nationwide and world-wide economies, etc. Consequently, many different formats of data exist—and considering also the many ways of processing these data, countless more data types do so.

This section explores some of the different types of data, both raw and derived alike. Furthermore, we describe which data formats have been used in the later analysis, the reasons for these choices, and how and from where the data have been acquired.

2.2.1 Data Acquisition

Depending on the type and format sought, financial data are not the easiest to come by. Of course, checking the stock price of a major company for the current day is easily done by opening a newspaper—or, more likely, the Internet. Many websites also provide visual representations of

[5]We focus primarily on technical analysis. The use of fundamental analysis is a topic for future work.

the stock price evolution through time, and some even have analysis tools, technical indicators, etc. However, if one wishes to perform one's own analysis from scratch, using e.g. historical time series data ranging many years back, then a further search through cyberspace is needed. Now, if one is not directly in contact with a bank or investment company and has access to their market databases, getting financial market data can require payments if the data are to have a certain quality. However, free financial data are in fact available on the Internet; some good sources, we have found, include Yahoo! Finance, Google Finance, and Quandl.[6] These sources provide data for stocks, indices, commodities, currencies, bonds, futures, options, etc. Most importantly for our case, however, they provide daily, weekly, and monthly data for an abundance of different stocks. Regarding data for shorter time scales—so-called *high-frequency data*—we have, after browsing through numerous websites, come to the conclusion that it generally comes at a price, and free data of this sort seems very difficult and almost impossible to come by.

2.2.2 Raw Data

In this study we are mainly concerned with the stock market—individual stocks as well as indexes comprising many stocks. However, for the sake of broadening the analysis, we will also consider some commodities and currencies to see if forecasting and prediction might work better on these than on stocks. The data available for currencies and commodities is slightly different than for stocks, with fewer types available; often, only the price itself is available. For stocks, however, which is our main focus, a wide array of data are available. This includes fundamental company-related numbers like debt, earnings, and other data than can be extracted from budgets and accounts. This type of data is used in fundamental analysis of stocks. On the other side, we have the more technical data associated with the stock as it is quoted on the markets. This, and the data than can be derived from it, is what we will mainly be using in the forecasting and prediction models. Recordings of this raw data consists of the following:

Date: The date, or time, of the recording.

Open: The price of the stock upon the opening of the market on which it is quoted; i.e. the stock price at the beginning of the time period.

High: The highest value of the stock price during the time period under consideration.

Low: The lowest value of the stock price during the time period under consideration.

Close: The price of the stock when the market closes; i.e. the stock price at the end of the time period.

Adjusted Close: The closing stock price adjusted for any dividends or stock splits.

Volume: The amount of stocks traded in the time period.

Evidently, these data depend on the time period, or time frame, considered. This results in even more available data, and even more choices to make as to what type of data to use. If the stock prices are quoted on a daily basis, the opening price is, of course, the stock price when the financial markets open on a particular trading day, and likewise for all the other data. Similarly for e.g. weekly and monthly data. As mentioned earlier, there exists also high-frequency data where the prices are quoted every few minutes, or seconds even. Finally, there exists so-called *tick data* which records the price changes from trade to trade. All of these different possibilities

[6] http://finance.yahoo.com/, https://www.google.com/finance, http://www.quandl.com/.

lead to the problem of which time frame to use. Due to the restricted availability of (free) high-frequency data, we will mainly be concerned with daily data.

Related to the problem of choosing the appropriate time scale is also the question of whether to use the physical time, the trading (or market) time (this is the most common type, which we also use here), or the number of transactions. There are merits and drawbacks for each choice, some of which are related to e.g. the fact that stock exchanges close at night, over weekends, during holidays, etc., as well as the fact that activity on the markets (when they are open) heavily depends on the social organization of business, time zones, etc. For a further discussion on these aspects—as well as some empirical evidence—see e.g. [3]. The raw data that will be used in this study belongs to the time scale of trading/market time, and it will mainly consist of the adjusted closing price for daily, weekly, and monthly data.

2.2.3 Derived Data and Data Preprocessing

The type of data mentioned above is seldom used in its raw form. Rather, it is often processed, or transformed, in one way or another, as well as used for the computation and derivation of other data. This expands the pool of available data even further, giving the researcher or practitioner even more options to choose from.

Now, consider a time series of stock prices, with the price at time t defined by $p(t) \equiv p_t$. As will be discussed later (in e.g. 4.1.1), it is inadequate to use this data for time series analysis and as input in machine learning methods, the main reason being its non-stationarity). The question then arises of which variable to investigate instead. Below, we list the most common choices and describe their respective merits and problems.

Price change:

In this case, one considers the simple price difference d_t between two subsequent times:

$$d_t = p_t - p_{t-1} \tag{2.2}$$

The positive thing about this approach is that nonlinear or stochastic transformations are not needed. The problem, however, is that the definition is seriously affected by changes in scale.

Discounted price change:

The discounted, or deflated, price change is defined by

$$\tilde{d}_t = (p_t - p_{t-1})\, D_t, \tag{2.3}$$

where D_t is a discounting, or deflation, factor that makes the prices be given in terms of "constant" money. Again, nonlinear transformations are not needed. As for the problem, the discounting factor is unpredictable over the long term, and nor is there a unique choice for it.

Return:

The return is defined by

$$r_t = \frac{p_t - p_{t-1}}{p_{t-1}} \tag{2.4}$$

This approach has the merit that returns provide a percentage gain or loss between subsequent times. On the other hand, the problem is that returns are sensitive to scale changes for long time horizons.

In fact, the above formula is the one-period return; that is, we consider the return in the period between $t-1$ and t. We can readily generalize this to a multiperiod return. If we consider the return for the k periods between $t-k$ and t, the k-period return is given by

$$r_t^k = \frac{p_t - p_{t-k}}{p_{t-k}} \tag{2.5}$$

Logarithmic return:

This approach uses the difference between two subsequent times of the logarithm of price:

$$r_t = \log p_t - \log p_{t-1} = \log \frac{p_t}{p_{t-1}} \tag{2.6}$$

With this approach the average correction of scale changes is incorporated without the need for discounting factors. The problems are related to the fact that the definition partly relies on the growth rate of the economy being constant (which it generally is not) as well as the fact that it utilizes a nonlinear transformation through the logarithm.

As the simple return above, the logarithmic return can also be generalized to multiple periods. The k period logarithmic return is, of course, just

$$r_t^k = \log p_t - \log p_{t-k} = \log \frac{p_t}{p_{t-k}} \tag{2.7}$$

Logarithmic returns, also called continuously compounded returns, enjoy some advantages over simple returns. First, it can be shown [40] that

$$r_t^k = r_1 + r_{t-1} + \ldots + r_{t-k+1}. \tag{2.8}$$

Thus, the logarithmic multiperiod return is simply the sum of the logarithmic one-period returns. Second, statistical properties of logarithmic returns are more tractable.

In the high-frequency domain, all four definitions are approximately equal [3]. For investigations concerning longer time periods, however, the most common choices are the latter two definitions; the return and the logarithmic return. In this case, we will use the logarithmic return.

Finally, an interesting and highly relevant type of data is the **volatility**. A common concept in financial analysis, the volatility V describes the variability of a stock price p, or more specifically the variability in returns r, and is used primarily as an estimate of investment risk. The standard definition is

$$V = \sqrt{\frac{1}{N-1}\sum_{t=1}^{N}(r_t - \bar{r})}, \tag{2.9}$$

where N is the number of observations in the time series, r_t is e.g. the logarithmic return $r_t = \log \frac{p_t}{p_{t-1}}$, and $\bar{r}$ is the mean $\bar{r} = \frac{1}{N}\sum_{t=1}^{N} r_t$.

Evidently, with this definition, the volatility equals the standard deviation of the return series r_t. Now, the volatility of stocks and other financial assets is generally seen to be time dependent; that is, the financial markets often exhibit periods of large swings—so-called volatile periods—followed by periods of smaller ups and downs. For this reason, if the volatility is to be used as an input in a model, a sliding window approach is often applied; here, the series is split into "windows," or periods, of a certain time length, and then the volatility is calculated for each

window to encompass the time-varying nature. As mentioned, the volatility is of great interest for financial analysts in regards to e.g. estimating investment risk. Moreover, relevant for our particular case, the volatility may be used as both input and output in prediction models. In fact, it has been shown empirically that volatility and predictability of financial time series are connected, although a theoretical explanation does not exist [12].

2.2.4 Technical and Fundamental Data

The above description and discussion dealt with transformations of the raw data in order to obtain a series that is better and easier to work with. But the raw data can also be used to derive an abundance of other data. Mainly of a technical nature, these forms of derived data are used extensively in technical analysis. They include indicators and oscillators such as moving averages, momentum, relative strength index, etc. We return to this shortly in Section 2.3 and give a detailed review of numerous technical indicators in Appendix A.

Apart from the raw data, its transformations, and the technical data that can be derived from it, we have another type of data: fundamental data. This is information concerning the activities and the financial situation of companies and the general economy. Most companies that are quoted on the markets are analysed on a regular basis by professional analysts at financial institutions. Supposed to give an indication of the "true" value of the company's stock, the analyses are often accompanied by a "buy" or "sell" recommendation. Such a fundamental analysis of a company typically focuses on the following three factors.

The general economy:
Some of the general economical factors that can affect the company (e.g. through import/export, etc.) include:

- Inflation.
- Interest rates.
- Trade balance.

The state of the relevant industry:
The condition of the industry to which the company belongs can have a significant effect. Influencing factors may include:

- Stock indexes (i.e. weighted means of a wide array of individual stocks) such as the US "S&P 500" or "Dow Jones", the German "DAX", the Danish "C20", etc.
- The value of competitors' stocks.
- The prices of related commodities such as oil and metals, as well as currencies.

The state of the company:
This information is extracted from the company's financial statements, budgets, and accounts, from which a number of useful variables can be calculated:

- P/E (Price/Earning): The stock price divided by the earnings per share during the last 12 months.
- Book value per share: Net assets (i.e. assets minus liabilities) divided by the total number of shares.
- Net profit margin: Net income divided by total sales.
- Debt ratio: Liabilities divided by total assets.

- Prognosis of future profits.
- Prognosis of future sales.

Some of the work performed by professional analysts is not publicly available, but there is still plenty of fundamental data available in financial magazines and on the Internet. However, when it comes to huge amounts of data in machine readable form, the availability of technical data is far better than for fundamental data.

Even though it is an interesting aspect of research, it is beyond the scope of this thesis to go into detail on the theory of fundamental analysis. We will include some fundamental data in a few experiments, but in terms of finance theory our focus is still on technical analysis.

2.2.5 Data Transformations

Above, we have described a wide array of raw, readily available data as well as computed and derived data. However, before this data is used as inputs in the prediction models it is often transformed in certain ways.

First, and perhaps most importantly, there is **normalization**, the purpose of which is to reduce range differences in the input variables. Since we work with both cross-sectional data in the form of different input variables as well as longitudinal data in the form of time series, we distinguish between two major types of normalization. First, the size differences between different input variables must be reduced to enforce similar behaviour in the modelling algorithms. For instance, before feeding data to an artificial neural network, all variables are often scaled to the range $[0, 1]$. Second, the size differences over time in an input time series must be reduced to remove the non-stationary effect, as will be discussed in more detail in 4.1.1. For example, if one considers the evolution of a stock price over several years, the values will often vary by large percentages; e.g. if a company started out really small but then worked its way up to be a major player on the markets. Several normalization procedures exist, the most common of which include transformations to relative changes (returns) as in (2.4) or logarithmic returns as in (2.6), or normalization using the mean and standard deviation. The return transformations were described and discussed earlier. As for the normalization using the mean and standard deviation, this is defined for some time-dependent variable $y(t)$ as

$$y_n(t) = \frac{y(t) - \mu_y(t)}{\sigma_y(t)}, \tag{2.10}$$

where it is important to note that the mean value $\mu_y(t)$ and standard deviation $\sigma_y(t)$ are computed using a running window of a certain length n, e.g. 30 days backwards. As mentioned in the earlier discussion of volatility, this better takes into account the time-dependent nature of the data. The resulting, normalized variable $y_n(t)$ then measures how many standard deviations the y values differ from their running mean.

Another transformation is **linearization**, which achieves the often advantageous task of removing obvious nonlinear factors that would otherwise have to be taken care of in the modelling. As an example we have the volume $V(t)$, which denotes the number of assets traded. This quantity is often distorted by extremely high figures resulting from individual trades by large institutions such as hedge funds, investment banks, etc. The linearization in this case may be a type of pre-processing function that squashes the data and suppresses values outside a certain boundary. For example, the input $V(t)$ may be transformed to $\tilde{V}(t)$ via

$$\tilde{V}(t) = \frac{1}{1 + \exp(-V(t))}. \tag{2.11}$$

Finally, a good way to reduce the complexity of the prediction models—and thus optimize and improve them—is through **dimensionality reduction**. We will most likely have a lot of different inputs, which somewhat contrasts the goal of simple, parsimonious models. Therefore, it is sometimes useful to combine some of these many inputs and compute aggregate entities to be used instead. Examples are running means (moving averages) of the closing price $p(t)$, the high $p^H(t)$, the low $p^L(t)$, and the volume $V(t)$. In this regard, the three prices $p(t)$, $p^H(t)$, and $p^L(t)$ may also be combined into a so-called *typical price* $\left(p(t) + p^H(t) + p^L(t)\right)/3$. The volume may also be included in such a new aggregate variable.

An efficient technique for dimensionality reduction of a dataset is also provided by so-called Principal Component Analysis (PCA). This works by finding a linear transformation T which is applied to the original dataset consisting of a number of p-dimensional vectors. The transformed vectors are still p-dimensional, but T is computed in such a way that the variances along each of the new p dimensions are sorted in descending order by their sizes. A dimensionality reduction of the dataset can then be performed by ignoring the dimensions with the smallest variances. The mean-squared error can be shown to be the sum of the variances of the truncated dimensions. The transformation T is computed by solving an *eigenvalue* problem, a common occurrence in linear algebra. The *eigenvectors* of the resulting correlation matrix correspond to the axes in the new transformed coordinate system, and the *eigenvalues* correspond to the variances measured along each of these axes. PCA has been successfully applied for dimensionality reduction in many areas including both image and speech recognition [13].

2.3 Technical and Fundamental Analysis

Technical analysis is the study and analysis of a security's[7] historical price and volume data in an effort to determine, or forecast, the direction of future prices.

Technical analysis has been around for a very long time, some aspects of it dating as far back as the 17th century with Joseph de la Vega's account of the Dutch markets [88]. Accounts on technical analysis were also developed in Asia in the 18th century and are said to be the precursor of today's immensely popular technique of candlestick charting[8] [89]. However, the strongest, most influential roots of modern-day technical analysis stem from Dow Theory, developed around 1900 by Charles Dow and further organized and collectively represented by William Peter Hamilton and later by Richard Wallace Schabacker [90], [91]. Other pioneers in the early 20th century include also Ralph Nelson Elliott, the developer of the popular Elliot Wave Theory[9]. More technical analysis tools and theories have been developed and enhanced in the recent decades, with an increasing emphasis on the use of computers for automated techniques and trading systems.

Contrasting fundamental analysis—the study of economic factors, such as e.g. earnings, dividends, and other company-related data as well as interest and exchange rates, etc., that influence the way investors price financial markets—technical analysis relies almost exclusively on raw historical data of price and volume[10]. In this regard, technical analysts employ a variety of methods, tools, and techniques, a popular one of which is the use of charts. Using charts, which plot the prices as a simple line, or in a more advanced way like in the candlestick technique

[7] Here, the term security refers to any tradable financial instrument; e.g. stocks, bonds, commodities, futures, indices, options, etc.

[8] Rather than showing the price time series as a simple line, candlestick techniques utilize several technical data, plotting them all in a combination of a line and a bar plot.

[9] In short, the technical analysis form of Elliot Wave Theory asserts that market prices unfold in specific patterns, or so-called Elliot waves, which consequently allows for forecasts of future prices [92].

[10] The volume of a financial asset is the amount of trades made in a given time period.

described above, and the volume as a bar graph, analysts seek to identify price patterns and trends in a stock, or other security, for the purpose of forecasting future prices—and ultimately for the purpose of financial gain. These price patterns range everywhere between simple ones to the more obscure, exotic formations bearing odd names such as "flag and pennant" or "cup and handle" patterns. This latter class of technical analysis techniques will not be considered here, as it often requires the practitioner to closely monitor the price chart, draw lines and other shapes on it, and then wait for it to resemble a given pattern.[11] This renders it very difficult to properly implement the techniques in a computer routine. Rather, what we will be concerned with in regards to technical analysis is the study of moving averages and other technical indicators that give information on e.g. the strength and direction of price trends. These indicators are all derived and calculated from the historical price and volume data, and can thus be coded somewhat easily as computer routines. Based on these indicators and their interpretations, we will attempt to forecast the future movement directions and prices of a selection of assets. In addition to this, our goal is also to employ these technical indicators and their interpretations as input in other forecasting methods; especially interesting is the use in machine learning methods such as Artificial Neural Networks, Support Vector Machines, and Random Forests, all of which we shall return to in Chapter 5.

Finally, we give a thorough description and discussion of a wide array of technical indicators in Appendix A.

[11] For an exhaustive description and discussion of technical analysis, see e.g. [101]

Chapter 3

Forecasting in Physics

This chapter highlights some of the areas in physics that make use of forecasting and prediction. We also describe, mostly in words, some of the methods applied in these areas.

Our primary focus is on weather forecasting, and we begin by making a thorough comparison between this discipline and financial forecasting. We highlight numerous similarities, but later on also some crucial differences. In these discussions we also touch upon chaos theory—a field of study very often encountered in physics. Later, after going through some mathematical methodologies shared by physics and finance (the numerical solution of differential equations), we construct a differential equation framework for modelling—and predicting—the fluctuation of stock prices. Our model draws inspiration from oscillatory systems in physics.

3.1 Weather Forecasting

Forecasting in general is an age-old discipline; for centuries, humans have speculated about many different aspects of the future and tried to improve their conjectures by observing nature, each other, and the entire world and what lies beyond. Indeed, forecasting is not mere speculation or blind guessing; it is qualified hypotheses founded on sound reason, thorough considerations, and, not to forget, science.

Financial forecasting, as considered in this thesis, is a relatively new area in this regard. Of course, speculation about the future prices of grain and gold has been around since the beginning of trade, but the modern version of financial forecasting—the systematic, technical approach—only dates back so far as the proper establishment of the financial markets and the invention of computers.

Turning to weather forecasting, this discipline actually has a lot in common with financial forecasting. Human beings have attempted to predict the weather informally for millennia—and formally so since the nineteenth century with the availability of more modern tools. The history of both the informal and formal approach is rather similar to that of financial forecasting. Indeed, the two disciplines are closely connected in certain regards; for example, the weather can mean the difference between a profitable harvest or a ruined one, and knowledge about the weather can significantly aid one in properly preparing for the season. Likewise, it can be an incredibly helpful tool when speculating in stocks and commodities related to e.g. agriculture.

Now, focusing on the more modern versions of the disciplines, the object of prediction in financial forecasting is the financial markets, quantified by e.g. prices, volatility, risk, etc. In weather forecasting, the object of prediction is the atmosphere, and in more quantifiable terms such things as temperature, precipitation, and wind. Other indicators that are used to forecast

the weather include: humidity, air pressure, solar radiation, Earth's rotation, water cycle, sea level pressure, sky cover, dew point, etc. Some aspects of weather forecasting employ past values of these indicators in order to forecast the future. The resemblance to the technical approach to financial forecasting is remarkable; here, one attempts to predict e.g. the future stock price using past values of both the price and a number of so-called *technical indicators*. Derived from data such as the high, low, and closing prices of the stock, and the amount of assets traded (the volume), these technical indicators can be viewed as the financial pendant to e.g. humidity and pressure in weather forecasting.[1] For example, one of the weather indicators is wind velocity (direction and speed), which is clearly a sort of rate of change. We meet similar indicators in finance; for instance, a popular tool in technical analysis is a stock's Momentum—an indicator measuring the rate of change (direction and speed) of the price.[2] A most common quantity in e.g. mechanics and many other fields in physics is also the acceleration. Similarly, one may compute, based on historical data, the acceleration of a stock's price. Both the momentum and acceleration of a stock may be used as technical tools to estimate "where the price is headed" (the trend) and how quickly and strongly it may be headed there.

In finance, the technical indicators are derived from the available technical data (see Section 2.2) and the fundamental data comes from the companies' budgets, accounts, etc. The case is naturally quite different when it comes to the weather. Of course, data are obtained by observing the atmosphere, and one of tools that help with this task are satellites. Weather satellites are invaluable for viewing large weather systems on a worldwide scale. They show cloud formations, large weather events such as hurricanes, and other global weather systems. With satellites, forecasters are able to see weather across the entire globe: the oceans, continents, and poles. Recent satellite data is very detailed, even to the point of showing states and cities.

In financial forecasting, the pendant to satellites could be, say, the news media (newspapers a decade or two ago, but the Internet in present time), banks, and other institutions, or perhaps more specifically the people (or machines, rather) that aggregate all the available financial data and relay it to the public, including private and professional practitioners and forecasters. Just as weather satellites began to see the light of day during the second half of the 20th century, so is the organized availability and accessibility of financial data a thing of the last few decades.

So far, we have only discussed weather forecasting in loose, general terms. We now dive a little bit deeper by considering the different layers and the workings of how forecasts are actually made.

The numerical branch of weather forecasting is basically the attempted prediction of the future state of the atmosphere. The atmosphere is a fluid, and as such, the prediction of its future state is done by sampling its current state and using the equations of fluid dynamics and thermodynamics. The solution to a fluid dynamics problem typically involves calculating various properties of the fluid, e.g. temperature, velocity, pressure, density, as functions of space (how the governing quantities vary across the globe, for different continents, countries, etc.) and time (how the properties vary e.g. from day to day). These properties are exactly some of the aforementioned weather indicators. Solving the fluid dynamics problem, and thus obtaining the weather forecast, is then done by first *initializing* models using this observed data. This starting point for the forecast is then fed to a set of equations—the so-called *primitive equations*—governing the physics and dynamics of the atmosphere. The primitive equations are a set of nonlinear partial differential equations (PDEs) comprising:

[1] Of course, fundamental indicators such as a company's earnings, debt, etc. should also be included here. This thesis just mainly considers the use of technical indicators for financial forecasting.

[2] We give a detailed description and discussion of a wide array of technical indicators in Appendix A.

- **Conservation of momentum**, consisting of a form of the Navier-Stokes equations.[3][4]
- A **thermal energy equation** relating the temperature (as a function of time and place) of the system to heat sources and sinks.
- A **continuity equation** representing the conservation of mass.

These nonlinear PDEs are generally impossible to solve exactly through analytical methods. Therefore, numerical methods are used to obtain approximate solutions. In numerous fields including weather forecasting, a very common and popular approach to the numerical solution of differential equations are Finite-Difference Methods[5]. Here, the main idea is to discretize the continuous temporal and spatial dimensions into a discrete grid. The observed, current data are used as initial conditions. Using these values in the discretized equations, one can compute the state of the atmosphere at the next time step—i.e. predict the state a short time into the future. The equations are then applied to this new atmospheric state, which yields predictions at a further time step into the future. Using this *time stepping* procedure, we can continue to advance forward in time until the solution reaches the desired forecast time (e.g. a forecast of tomorrow's weather, or perhaps a weekly 5-day forecast).

Now, had the system been deterministic *and* insensitive to initial conditions, we would have been able to precisely predict the weather for all time to come. But the reality is not so. The atmosphere has a *chaotic* nature; it is a nonlinear dynamical system, and even though it is deterministic[6], it cannot be precisely predicted because it is highly sensitive to initial conditions. Small differences in initial conditions (such as those due to rounding errors in numerical computation) yield widely diverging outcomes for such chaotic systems, rendering long-term prediction impossible in general. The heart of chaos theory was beautifully summarized by Edward Lorenz in 1963: [52]

> *"When the present determines the future, but the approximate present does not approximately determine the future."*

Thus, due to the chaotic nature of the atmosphere, it is limited how far we can step forward in time before the errors become too large, and hence the results and predictions too uncertain and unusable. As also proposed by Edward Lorenz in 1963, long range forecasts, those made at a range of two weeks or more, are impossible to definitively predict the state of the atmosphere. It has been found that, in numerical weather models, extremely small errors in initial values double roughly every five days for variables such as temperature and wind velocity [51]. Nowadays, the horizon in weather forecasting is often about 5-7 days. This is also what we can read and see in the news with weekly 5-day weather prognoses. But as we also experience from time to time, these forecasts do not always hold exactly—which is the chaotic nature of the atmosphere and the errors in initial conditions that are making themselves felt. However, with the constant technological advances (satellites, computers, etc.), we continually push the borders of precision and may soon see more accurate forecasts for even longer horizons.

[3]The Navier-Stokes equations are nonlinear partial differential equations dictating not position but rather velocity. A solution of the equations is in the form of a velocity field, which is a description of the velocity of the fluid at any given point in space and time.

[4]It is beyond the scope of this thesis to go into any further detail on e.g. fluid dynamics and the underlying theory. For a thorough examination, we refer to e.g. [50].

[5]For a good examination of finite-difference methods, and numerical methods for differential equations in general, we refer to e.g. [4].

[6]That a system is deterministic means that its future behaviour is fully determined by its initial conditions, with no random elements involved.

3.1.1 Crucial Differences

With the above discussion of the chaotic nature of the atmosphere and the resulting implications, we are reaching one of the crucial differences between weather forecasting and financial forecasting.

The atmosphere, and thus the weather, evolves according to the laws of physics—regardless of our observations and predictions. It has got its life, so to speak. Its chaotic nature has two important implications: (i) it is deterministic (non-random), and (ii) it is highly sensitive to initial conditions. The second point means that the accuracy and usability of our predictions depend solely on the numerical precision we are able to obtain. That is, the limitations of our forecasts are due to the limitations in our technology. Consequently, technological improvements will provide us with better tools to observe nature, and thus enable us to more accurately predict its future state.

Turning to the financial markets, one may raise the question whether these are chaotic or random. If they are chaotic, then the same must apply as for the atmosphere; namely that the markets are fully determined by the initial conditions and that any limitations in forecasts are entirely due to slight errors in these conditions. Intuitively, this case sounds rather implausible. On the other side, of course, one may argue that the right initial conditions are just extremely complex and entirely or partly unknown to us. The question also arises of what exactly the initial conditions are. A plausible candidate would be the history of the markets, i.e. all past prices and other financial data. This is what we do in this thesis; using past and current data to predict the future. Naturally, for reasons of availability and computing power, we are only able to use so much data. Poor results may therefore be attributed to either unpredictable markets or, from the point of view of firm believers in chaotic markets, insufficient data. In the latter case, one may argue that the reason for poor results is too little data, and that the future state of the markets is still governed by all the available past data. The discussion of chaotic versus random markets is extremely interesting, and both has been, and still is, the subject of much debate and research. As mentioned in Section 2.1, people are largely split in two camps: (i) those who cling to one of the versions of the Efficient Market Hypothesis and thus believe in unpredictable, random markets; and (ii) those who reject the Efficient Market Hypothesis. We note that the latter point is not necessarily synonymous with believing in chaotic markets; there have been numerous attempts—not related to chaos theory—at predicting the markets (see e.g. our overview in Section 1.3). The attempts show greatly varying degrees of success, which is why the "war" rages on. In any case, though, some researchers have in fact found interesting empirical evidence that points to some degree of chaos [53], [54], [55].

A related interesting discussion of weather forecasting versus financial forecasting deals with the influence of our observations and predictions. As briefly mentioned above, the evolution of the atmosphere is governed by the laws of physics. The future states are wholly uninfluenced by what we observe and predict. The financial markets, on the other hand, are very much influenced by the predictions, especially if the predictions are made public or if multiple agents, traders, and investors employ the same systems and algorithms. For example, banks and investment companies, and even independent specialists or professionals, all provide an abundance of "stock picks" and recommendations that many private investors follow. Also, if multiple traders act on the same signals[7], then the stock often ends up going in the indicated direction if enough money is involved. Thus, stock predictions are to some extent self-fulfilling. This makes a big impact on the correctness of the predictions and for their value and usability; i.e. if there is enough belief in something, then that something is likely to happen. For example, if all the investment banks

[7] The signals often come from the interpretation of one or more technical indicators, which we cover in Appendix A.

want the markets to rise further, and send out positive recommendations, then the markets are likely to rise. Similarly, if everyone sees good, prosperous times ahead, then those times may come ... for a while at least... Because in these cases we may observe what is called a financial "bubble". And eventually this bubble will burst, with a market crash to follow—just as we saw it with the "dot-com" bubble around the turn of the millennium and the recent financial crisis in 2008. The self-fulfilling nature of stock predictions is a little bit like observing a quantum system; the observation itself fixes the quantum system.

3.2 Prediction and Forecasting in Other Areas of Physics

Other areas, in physics and related sub-disciplines such as e.g. geophysics, where forecasting is relevant and important include sunspot activity, climate change, and natural disasters (e.g. volcano eruptions and earthquakes), etc. We describe these points below and discuss similarities and differences to financial forecasting and its prediction methods.[8]

Sunspot Activity

Caused by intense magnetic activity, sunspots are temporary phenomena on the photosphere[9] of the Sun. They form areas of reduced surface temperature compared to the surrounding material and thus appear as dark spots. Moreover, they usually come in pairs, with each sunspot having the opposite magnetic pole to the other. Sunspots are wild and violent phenomena that can be thought of as self-perpetuating storms, analogous in some ways to terrestrial hurricanes.

Sunspot activity has been found to come in cycles with a period of roughly 11 years [56]. We can quickly investigate this ourselves. We have found yearly sunspot activity data for the past ~ 300 years, ranging all the way back to 1700.[10] In Fig. 3.1 we plot the a quantity called the *Zürich sunspot relative number*, which measures both number and size of sunspots.

[8] For the sake of brevity and the limited scope of this thesis, we will not go into as much detail on these matters as we did with weather forecasting.

[9] The depth of a star's outer shell from which light is radiated.

[10] The dataset only contained data up to 1989. However, the important features—i.e. the cyclical nature—is still more than evident.

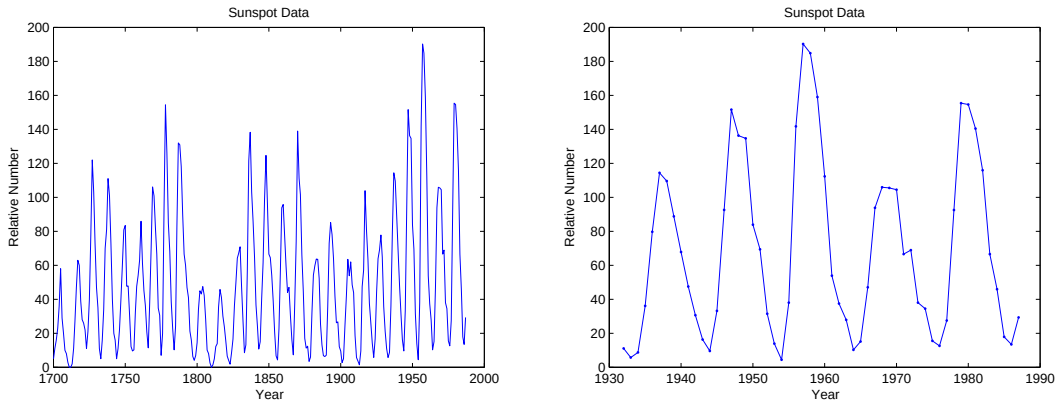

Figure 3.1: Observed sunspot activity. The vertical axis is the Zürich sunspot relative number, which measures both sunspot number and size. Left: the full period from 1700 to 1989. Right: zoom-in on the most recent 55 years.

We observe a clear indication of the cyclic nature of sunspot activity. In addition, looking at e.g. the peaks—the so-called *sunspot maxima*—we note that the period seems to be roughly the aforementioned 11 years.

Due to its effect on weather (in space and on Earth) and climate, it is a relevant and important task to attempt to predict the future sunspot activity. There are various approaches to this task, and the prediction methods generally form three main groups: [57]

- **Precursor methods**, which rely on the value of some measure of solar activity or magnetism at a specified time to predict the amplitude of the following solar maximum (the peak). For an example of prediction using these methods, see e.g. [58].
- **Extrapolation methods**, which apply time series methods for analysis and forecasting. For an example, see e.g. [59].
- **Model-based methods**, which, instead of an analysis of observational data alone, use dynamo models to predict solar activity. [60] provides a practical research example.

[57] states that, in their overall performance during the course of the last few solar cycles, precursor methods have been superior to extrapolation methods. In addition, model-based forecasts from the last group have not yet had a chance to prove their skills.

The first two groups of methods are particularly interesting in a comparison to financial forecasting and our approach to it in this thesis.

The second bullet describes the extrapolation methods for sunspot prediction which employ time series models such as ARMA, ARIMA, etc. that model the conditional mean based on past observations. Such time series models are also very commonly used for analysis and forecasting in finance, and we also both review their theory (Chapter 4) and test their performance (Chapter 6) in this thesis. In particular, the use of past of observations for stock market prediction is interesting and plausible due to the autocorrelation observed in asset returns—something we also find empirical evidence for here.

Then there are the precursor methods for sunspot activity prediction which use some measures of solar activity or magnetism to predict future solar maxima. Thus, the methods generally use as

predictor variables one or more *indicators*. This is very similar to several approaches to financial forecasting (technical and fundamental alike), where one also employs a variety of indicators that may or may not possess predictive power. In regards to sunspot activity, the indicators are, as mentioned, related to solar activity or magnetism. In finance, the indicators may be of a technical nature, e.g. moving averages, momentum, volume, etc., or a fundamental nature, e.g. company accounts, budgets, etc.[11] There are several ways in which one can use these indicators for prediction, both in the case of sunspot activity and stock markets. One approach is to use statistical machine learning methods. This is what we do here (in addition to time series models); we use Artificial Neural Networks, Support Vector Machines, and Random Forests[12] for predicting future stock prices using both past prices and a wide array of different technical indicators. Such methods have also been widely put to use—and successfully so—for predicting sunspot activity; see e.g. [61], [62], [63], [64].

As briefly mentioned earlier, sunspot activity, and solar variation in general, causes changes in space weather and the weather on Earth. It thus shares a close connection to our previous section on weather forecasting. Sunspot activity also affects the climate on Earth, which has been a topic of hot debate especially in the recent decade. This leads us to the following section where we discuss climate change forecasting and compare it to financial forecasting.

Climate Change

Whereas the notion of weather and changes herein are often related to the temperature, precipitation, wind, etc. for periods ranging from one day to one week, the climate of the Earth deals with significantly longer periods. The term *climate change* denotes a significant and lasting change in the weather patterns over periods ranging from decades to millions of years. The change may be related to the average weather conditions (e.g. colder or warmer weather in general) or to variability of weather around the average conditions (e.g. more or fewer extreme events). As mentioned earlier in Section 3.1, the weather is governed by the state of the fluid, chaotic atmosphere. Changes in the climate, on the other hand, are caused by factors such as variations in solar radiation received by Earth (influenced by sunspot activity as described in the previous part), plate tectonics, and volcano eruptions (the latter two of which are covered in the next part). Certain human activities have also been identified as significant causes of recent climate change [65]—something which is often referred to as *global warming* and has been a hot (no pun intended) topic of debate in the recent decade.

Especially in recent years[13], few scientific topics evoke such general interest and public discussion as climate change [66]. It is a subject that has been highly politicized. New results enter the environmental debate as evidence supporting a particular position. And usually, the background and perspective needed to understand the result have been stripped away in order to form an appropriate sound bite for the public to digest and the media to relay and exaggerate. Political motivations aside, however, the attention is still understandable given the importance of climate to agriculture and energy use. The fear of global warming and the greenhouse effect[14]

[11] We describe and discuss technical and fundamental analysis in Chapter 2, and review a variety of technical indicators in Appendix A.

[12] See Chapter 5 for a review of the theory.

[13] Although, at the time of writing this, the trend—especially in the media—seems to be waning as compared to e.g. 5 years ago. Still, however, we observe in present day a focus on being "green" and eco-friendly unlike never before. So, perhaps the trend has changed from wide media attention to governments and people being more aware and taking the necessary steps to reduce pollution and carbon dioxide emission.

[14] The process by which thermal radiation from e.g. Earth's surface is absorbed by the greenhouse gasses in the atmosphere and partly re-radiated back to towards the surface, with an increase in the average surface temperature to follow.

has been used as justification for reducing the use of fossil fuels and instead focusing on more "green", eco-friendly alternatives.

Evidently, via its potentially destructive effects on human kind and our continued existence on Earth—e.g. the serious event of the poles melting, or, to take the opposite extreme, a new ice age—climate plays an extremely vital role to us humans, and hence to the political discussion. Regarding the prediction of climate change, there is thus a strong politically inspired tendency towards overinterpretation and/or subjective interpretation. Consequently, the predictions may vary significantly over time, depending on the dominant belief. For example, if there is strong focus on e.g. global warming and human-caused climate changes—something which the media is certain to relay to the public—then governments and other institutions—if holding the same belief—may even go so far as to funnel their grants and financial support to researchers who investigate and find support for this very aspect. There is also a clear connection to business and the industry; for instance, energy companies making use of fossil fuels certainly try their best to dampen the focus on global warming and show that this is not what is causing climate changes. Similarly, companies dealing with "green", eco-friendly energy alternatives clearly have incentive to push in the opposite direction. Here, money and power enter the picture, and as we have seen throughout history, these can be extremely strong sources of motivation.

Money, power, and politics is a hard-hitting trio whose constituent parts are inextricably linked together. The existence of politically inspired motivations in climate change forecasting is an irrevocable fact. And the same is true in finance and financial forecasting where, more than anything, money and power are primary driving forces. Predictions and stock recommendations may largely be influenced by underlying beliefs, hopes, and incentives. For example, banks or investment companies may announce strong "buy" or "sell" signals for certain stocks, and consequently publish recommendations to the public which encourage private investors to do as suggested. The companies may have honourable intentions, but their recommendations may also be motivated by their own wants and wishes; i.e., they may own a lot of shares in a given stock themselves, and then recommend others to buy this stock so that the price rises and their shares increase in value. Vice versa if they wanted a stock to drop. And there are numerous other examples and scenarios. In this way, the agents on the markets can have a strong influence on the actual direction of the market and what sort of predictions and recommendations are made. This is in a very similar vein as what we discussed for climate change predictions.[15]

Natural Disasters and Stock Market Crashes

Between scientists from e.g. physics, geology, etc., there are interdisciplinary attempts at predicting natural disasters such as earthquakes and volcanic eruptions. The motivation is obvious since such events can have terrible consequences for people living in the surrounding areas.

In regards to earthquakes, there was great optimism amongst scientists in the 1970s that some method of prediction might be found; however, by the end of the 1990s, continuing failure led many to question whether it was possible at all [67], [68]. This also appears to be the common belief today, even though many also hold that, given enough resources, prediction might still be possible [69].

Just as we saw it earlier with precursor methods for sunspot prediction, there has been, and still is, a great deal of research going into earthquake precursors, i.e. phenomenons that might give effective warning of an impending earthquake. Numerous precursor candidates have

[15] Of course, there is an important difference here in that climate change predictions do not cause the actual climate to change (except through the effects that people's environmental awareness and governments' pollution and emission policies have on global warming). In finance, predictions *may* (or may not) dictate the actual future evolution. However, the similarity still holds that the forecasts themselves are often strongly motivated by political and financial incentives.

been suggested—numbering in the several hundreds or even thousands [67]—but the search has still thus far been unsuccessful [70]. Some precursors that have been studied include e.g. animal behaviour, changes in the velocities of rock formations, radon emissions, electro-magnetic variations, etc. But none of these phenomena have been consistently or convincingly successful [70].

With all the research in the field and the mostly disappointing results, there is no doubt that earthquake prediction is extremely difficult. This difficulty may be the reason that the vast majority of attempts have failed so far. The other option is that earthquake prediction is entirely impossible—a belief that is strongly disputed [71], but still has yet to be concretely disproved [70].

This great difficulty in prediction is also what we observe in finance and the stock market; no attempt has been able to consistently predict the markets. No attempt that we know of, at least, since it is probably unlikely that people would share such "money machine" with others, consequently rendering it useless if many began to use it. This marks an important difference to earthquakes, and natural disasters in general, where correct predictions are beneficial for all and in everyone's interest (greed-fueled motivations are almost non-existing).

Another crucial difference between the prediction of natural disasters and financial markets is the very nature of what is being predicted and hence the possibility hereof. Earthquakes, volcanic eruptions, and other such disasters are natural phenomena. They are all related to how our physical world is put together, to the physical laws describing it. They are not something we humans have created. The financial markets, on the other hand, are a human construct. It may be affected by natural phenomena—disasters, wars, etc.[16]—but is to a much greater extent driven by human psychology. Thus, although fiendishly difficult to predict as it stands right now, earthquakes, volcanic eruptions, etc. may turn out to be predictable due to them being natural, physical phenomena. Financial markets, as something created—and very much affected—by humans, seem more likely to be something that is actually unpredictable.

Lastly, we remark that there is an interesting comparison between natural disasters (earthquakes, etc.) and "disasters" on the stock market (crashes). In this regard there are actually examples of scientists in e.g. geophysics that attempt to predict stock market crashes. For example, inspired by the swings and quakes of natural systems, [72], [73] employs a logarithmic-periodic power-law model for describing the characteristic behaviour of a speculative bubble and predicting its subsequent crash. Mathematically, the log-periodic power-law proposes that the price p of an instrument (stock, index, etc.) evolves at time t according to

$$p(t) = A + B\,(t_c - t)^z + C\,(t_c - t)^z \cos\left(\omega \log\left(t_c - t\right) + \Phi\right), \tag{3.1}$$

where t_c is the most probable time of crash, z is exponential growth, ω is oscillation amplitude, and A, B, C, Φ are constants. The power-law part describes the price increases (characteristic for bubbles), while the log-periodic part governs the fact that the oscillations become faster and faster (shorter period) and exhibit smaller and smaller amplitude up until the time at which the bubble bursts.

It is beyond the scope of this thesis to go into further detail on this matter. But the literature is rich with examples; see e.g. [74], [75], [76], [77], [78]. Investigating if and how stock market crashes can be predicted using such log-periodic power laws and/or other models inspired by e.g. physics is a very interesting topic of research—and a top candidate for our future work.

[16] For a current example, take e.g. the situation in Russia and Ukraine which has caused the western world to limit or completely shut connections with Russia, consequently causing Russian-related stocks to plummet.

3.3 PDEs in Physics and Finance

Above, we considered several aspects related to physics where prediction and forecasting was both relevant and important. In a comparison of these to financial forecasting we discussed several similarities crucial differences. We now make a slight turn back to partial differential equations (discussed in regards to weather forecasting in Section 3.1) and some of the similarities and shared methodologies in physics and finance.

Finance and physics have many things in common and apply many of the same mathematical methods and models. Earlier, we described how weather forecasting is performed via the numerical solution of a set of partial differential equations. Indeed, PDEs and their numerical solution methods are also used extensively in finance. For example, the price of an option[17] is, under several assumptions, governed by the Black-Scholes PDE[18]

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0, \tag{3.2}$$

where V is the option price (a function of time and the price S of the underlying stock), $sigma$ is the volatility (standard deviation) of the stock's returns, and r is the risk-free interest rate.

As it happens, through a change of variables, this PDE can be transformed to

$$\frac{\partial u}{\partial t} = C\frac{\partial^2 u}{\partial x^2} \tag{3.3}$$

But this is just the one-dimensional *heat equation* from physics! With suitable initial and boundary conditions, this governs the distribution of heat (temperature) in a bar.

In the simple case of European options, the Black-Scholes PDE can be solved analytically. However, for more advanced options, e.g. American options and so-called "exotic" options such as Asian options, analytical solutions no longer exist and one must turn to numerical methods. A popular choice in this regard is Finite-Difference Methods—just as it is for solving the PDEs occurring in weather forecasting (as described in earlier in Section 3.1) and numerous other fields.

Inspired by the differential equation approach in weather forecasting (and countless other fields in physics for that matter) we will try to build a somewhat similar framework for financial forecasting. The governing equations will not be of the heat equation type (or more generally so-called *parabolic* PDEs) encountered in option pricing, nor any other type of partial differential equations. Rather, we will focus on ordinary differential equations, with a specific inspiration drawn from oscillatory systems. We describe our model in detail in the following section.

3.4 Oscillator Model for Financial Forecasting

Motivated by differential equation frameworks in numerous fields, and drawing inspiration from [84], we will attempt to build our own model for the price fluctuations of a financial asset. As with any model, we will be considering a simplification of reality, with one or more accompanying assumptions. These assumptions are actually quite plausible, and we motivate them as follows.

Countless agents trade on the financial markets; private investors, professional analysts and traders, etc. No matter their background, they share a common goal; they want to make money.

[17] In finance, an option is a contract which gives the buyer the right, but not the obligation, to buy sell an underlying asset at a specified price on or before a specified date. The seller has the obligation to fulfil the transaction. If the buyer has the right to buy (sell) something, the option is called a *call* (*put*) option. If the owner is restricted to exercising his right or not at the expiry date, it is a European option. If exercise is allowed on or before expiry, it is an American option.

[18] Derived by Fischer Black and Myron Scholes in their seminal paper from 1973 [48].

And making money on the financial markets requires timing, making the right decisions[19], and managing one's risk. To put it in a simple—but nonetheless true—way, this can be boiled down to buying when the price is low and selling when the price is high. As human beings, driven to a large extent by emotions, it is only natural to want to lock in a profit at some point, to ensure we get our hands on that glittering gold. Both private investors, professional traders, and automated trading systems will eventually lock in a profit if it is present. So, if we own shares in a stock that has risen, we probably want to sell those shares at some point when we—based on emotions, intuition, historical patterns, or complex trading strategies—deem the stock to have risen enough. It is very common for many agents to do this—if not at once, then within some limited time frame—and we may observe a sort of herd-like behaviour where people suddenly begin to hurry away from the stock, selling it. When the majority acts the same, the stock follows accordingly; thus, when investors sell to lock in gains, it is common for the stock price to decrease. A similar thing occurs in the other end of the spectrum when the stock price is low; here, investors eye an opportunity to buy at bargain prices (in order to later sell and hopefully profit). If the herd agrees on this—that the price is sufficiently low and will probably rise back up soon—their buying the stock will often cause the price to increase.

Now, the question remains of *when* to buy and sell. That is, *how much* should the price increase before the herd acts and sells to lock in gains. Similarly, how much should the price decrease for investors to act on the opportunity of buying at bargain prices? Knowing this amount (which probably varies) is certainly craved by many—if it even exists. However, in our simplified model of the real world, we will assume that it does; and we will denote the base "magic" price level by p^*. The price at time t is denoted $p(t)$.

This investment logic and behaviour motivates the following two assumptions:

- As the stock price $p(t)$ **increases** above p^*, investors SELL and the slope of the price **decreases** at a rate proportional to $p(t) - p^*$.
- As the stock price $p(t)$ **decreases** below p^*, investors BUY and the slope of the price **increases** at a rate proportional to $p(t) - p^*$.

Note that we are dealing with *slopes* and *rates of change*, the introduction of which will become evident shortly. The *slope* of the price between the consecutive times $t-1$ and t is just

$$\frac{p(t) - p(t-1)}{t - (t-1)} = \frac{p(t) - p(t-1)}{1} = p(t) - p(t-1). \tag{3.4}$$

This slope, or rate of change, of the price is the financial pendant to the physical notion of velocity, i.e. the rate of change of position. Now, the *change* in slope is then

$$[p(t+1) - p(t)] - [p(t) - p(t-1)], \tag{3.5}$$

which can be understood as the *acceleration* of price.

The assumptions in 3.4 then imply that this change in slope is proportional to $p(t) - p^*$, i.e.

$$[p(t+1) - p(t)] - [p(t) - p(t-1)] = -\omega^2 [p(t) - p^*], \tag{3.6}$$

where ω^2 is some constant of proportionality, and the minus sign ensures that the slope decreases (increases) for $p(t)$ above (below) p^*, in accordance with the assumptions.

We can rearrange terms in Eq. (3.6) to obtain

$$p(t+1) = \left(2 - \omega^2\right) p(t) - p(t-1) + \omega^2 p^* \tag{3.7}$$

[19] Whatever those decisions may be.

Thus, if we find ourselves at time t, this expression gives a prediction of the next stock price $p(t+1)$ in terms of the current price $p(t)$ and previous price $p(t-1)$.

Now, Eq. (3.6) is evidently a difference equation. If we go to the limit of infinitesimal time steps, we can rewrite it as a second order *differential equation*:

$$\frac{d^2p}{dt^2} = -\omega^2 \left[p(t) - p^*\right] \tag{3.8}$$

An expression like this is a well-known phenomenon in physics. Indeed, if p^* is a constant, the solutions $p(t)$ will oscillate sinusoidally about p^* with a period $T = 2\pi/\omega$ (where we identify ω as the angular frequency). Hence, $\omega = 2\pi/T$ and we can rewrite Eq. (3.7) as

$$p(t+1) = \left(2 - (2\pi/T)^2\right) p(t) - p(t-1) + (2\pi/T)^2 p^* \tag{3.9}$$

Here, T should be chosen to be some proper value. A crude way would be to just pick a somewhat sensible value. Another approach is to estimate it using past data in a similar way as one does for e.g. neural networks, etc.; that is by finding the value that yields the best performance (lowest error) on some training set.

The question remains, however, of how to choose the price p^*. To begin with, one may try the average price of the stock over the past n days, i.e. an n-period simple moving average (SMA), or, more advanced, an exponential moving average (EMA). Another option which includes the volume of traded assets is the volume-weighted moving average (VMA).

It should also be clear what the price $p(t)$ is. The obvious choice is the closing price of the asset. Other options include the average of the high and the low, $p(t) = (p^{high} + p^{low})/2$ or the typical price $p(t) = (p^{high} + p^{close} + p^{low})/3$. Yet another option—and perhaps the best one—is a short-period moving average which smooths the data, removing the noise and erratic behaviour that is especially pronounced for daily data. It is important that the period is not too large so as to maintain a close similarity to the unsmoothed prices. Depending on the purpose and application, a good choice is $n = 5$, which, for daily data, is the average price for the past trading week. Note that, when using a moving average for $p(t)$, the price p^* (itself a moving average of $p(t)$) becomes a moving average of a moving average. Like the period T, the best moving average periods may be found by searching parameter space for the combination yielding the best performance on some training set.

The formula in Eq. (3.9) provides a prediction of the next period's price. But the model also allows for multi-step ahead forecasting if we, at some point in time, stop using the actual prices $p(t)$ and $p(t-1)$ and start using the previously predicted prices instead.

3.4.1 Extending the Model to Nonlinear Form

As it is now, the model in Eq. (3.6) or (3.8) governs a linear system. We can introduce nonlinearity by assuming that the rate of change of $p(t) - p(t-1)$, or d^2p/dt^2 in differential form—i.e. the acceleration of price—satisfies

$$\frac{d^2p}{dt^2} = -\beta \left[p(t) - p^*\right] + \alpha \left[p(t) - p^*\right]^3, \tag{3.10}$$

where $\beta, \alpha > 0$ (β corresponds to ω^2 used in the above).

Right now, the acceleration of price, d^2p/dt^2, depends solely on the deviation $p(t) - p^*$. We can make things more interesting by adding a term governing how quickly the price changes, i.e. the velocity dp/dt (or $p(t) - p(t-1)$ in difference form). The term should have the following effect:

- When $p(t) > p^*$ and the price increases, i.e. $p(t) - p(t-1) > 0$, then the price level p^* attracts the price. When the price then begins to decrease, i.e. $p(t) - p(t-1) < 0$, then the attraction wears off.
- When $p(t) < p^*$ and the price decreases, i.e. $p(t) - p(t-1) < 0$, then the price level p^* attracts the price. When the price then begins to increase, i.e. $p(t) - p(t-1) > 0$, then the attraction wears off.

The intuition and reasoning behind this is as follows. First, our previous assumptions already state that the price $p(t)$ seeks back to the base level p^* (like a spring seeking back to its equilibrium). When the price is above this level, and is headed further up with positive velocity (increasing price), then the pull towards equilibrium strengthens, i.e. there is a negative, downwards acceleration. When the price then begins to decrease, but is still above p^*, the pull weakens. Vice versa when the price is below p^*.

Hence, a positive (negative) velocity term, dp/dt or $p(t) - p(t-1)$, should give a negative (positive) contribution to the acceleration. In difference and differential form, respectively, our model thus becomes

$$[p(t+1) - p(t)] - [p(t) - p(t-1)] = -\beta\left[p(t) - p^*\right] + \alpha\left[p(t) - p^*\right]^3 - \delta\left[p(t) - p(t-1)\right], \quad (3.11)$$

$$\frac{d^2p}{dt^2} = -\beta\left[p(t) - p^*\right] + \alpha\left[p(t) - p^*\right]^3 - \delta\frac{dp}{dt}, \quad (3.12)$$

where $\beta, \alpha, \gamma > 0$.

The similarity to a well-known complex, dynamical system in physics may began to appear. Indeed, if we let $x(t) = p(t) - p^*$ and rearrange, then we obtain

$$\frac{d^2x}{dt^2} + \delta\frac{dx}{dt} + \beta x - \alpha x^3 = 0, \quad (3.13)$$

which is the unforced Duffing oscillator!

Now, our current model only includes the current and past prices as well as the base price level. Indeed, the unforced Duffing oscillator is an autonomous system that disregards outside influences. But there are clearly other variables which affect the stock price; e.g. seasonal variations, periodic changes in interest rates, etc. These outside influences can be added via a periodic term:[20]

$$\frac{d^2x}{dt^2} + \delta\frac{dx}{dt} + \beta x - \alpha x^3 = A cos(\omega t), \quad (3.14)$$

which is, of course, the forced Duffing oscillator!

The Duffing equation is a nonlinear second order differential equation used to model certain damped and driven oscillations. With the above notation we obtain the special case of simple harmonic motion when $\delta, \alpha = 0$. Similarly, if the nonlinear term is $\alpha = 0$, then the equation describes a damped and driven simple harmonic oscillator. In the forced case, $A \neq 0$, the system can even exhibit chaotic behaviour. The many parameter combinations and possibilities makes the Duffing oscillator an interesting system to study. Our main use for it is as a basis for our oscillator-inspired model for stock prices, and we refer to the rich literature for further descriptions and detailed analyses; e.g. [79], [80], [81], [82], [83].

[20] The ω appearing in the periodic term on the right-hand side of Eq. (3.14) should be distinguished from the ω used previously.

Now, (3.14) is a second order equation. In order to solve it, we first let $dx/dt = y$ and write it as a set of coupled first order equations:

$$\frac{dx}{dt} = y \tag{3.15}$$

$$\frac{dy}{dt} = -\delta y - \beta x + \alpha x^3 + Acos(\omega t) \tag{3.16}$$

$$\tag{3.17}$$

The problem can now be solved numerically using e.g. the popular fourth order Runge-Kutta (RK4) method[21].

Due to the time and size limitations of this study we will not make practical applications of the Duffing-version of the oscillator model, but rather focus on the slightly simpler case. However, the use of the Duffing equation for modelling stock price fluctuations is a topic for future work.

[21] For more information on numerical methods for differential equations in general, and Runge-Kutta methods in particular, we refer to e.g. [4],[5].

Chapter 4

Traditional Time Series Analysis

This chapter introduces several techniques that are useful for analysing time series data; i.e. data where successive values represent consecutive measurements or observations made at certain time intervals. As the financial markets evolve in time—due to their inherent dependence on changes in worldwide and country-specific economic and political factors, in the performances of individual companies, and in the behaviour and psychology of individual investors, etc.—the data that can be extracted from these markets (e.g. prices, volumes, company earnings, etc.) are inherently of a time series nature. There is a vast amount of various time series methods, each with a large number of different variations and extensions. Covering all of this theory is beyond the scope of this thesis[1], and we shall only consider those methods applied in our analyses and experiments for the purpose of financial forecasting.

We begin by covering some basic concepts of time series analysis that will be used and referred to in the later sections. We then proceed by treating a variety of conditional mean models, beginning with the simple autoregressive (AR) model and moving-average (MA) model. These models will not themselves be used in practice in this study, but their theoretical foundations are used to entertain the combined autoregressive moving-average (ARMA) model, which will be applied in practice. ARMA models are then generalized and extended to include so-called unit roots (ARIMA models) and external/exogenous covariates (ARMAX models). All of these models are used to model the conditional mean of the time series. However, since volatility plays an important role in finance, we are also interested in the variance of the time series. For this reason, we cover an array of conditional variance models, including the autoregressive conditional heteroscedastic (ARCH) model, its generalized version (GARCH), as well as a selection of other extensions.

4.1 Characteristics of Time Series

This section introduces some basic concepts that are essential in time series analysis.

4.1.1 Stationarity

A common assumption in many time series techniques is that the data are stationary. A stationary process has the property that the mean, variance, and autocorrelation (to be introduced in the next subsection 4.1.2) structure do not change over time. Qualitatively, this means a flat

[1] For books and other detailed discussions of the plethora of time series methods, see e.g. [36], [37], [38], [39], [40].

looking series, without trend, constant variance over time, a constant autocorrelation structure over time, and no periodic fluctuations (i.e. so-called seasonality).

In more precise mathematical terms, we say that the a time series $\{r_t\}$ is *strictly stationary* if the joint distribution of $(r_{t_1}, r_{t_2}, \ldots, r_{t_k})$ is identical to that of $(r_{t_1+t}, r_{t_2+t}, \ldots, r_{t_k+t})$ for all t, where k is an arbitrary positive integer and $(t_1, \ldots, t_k)$ is a collection of k positive integers. In words, strict stationarity means that the joint distribution $(r_{t_1}, r_{t_2}, \ldots, r_{t_k})$ is invariant under time shift. This is a very strong condition that is difficult to verify empirically; indeed, for financial data, strict stationarity is a very, very poor assumption.

Therefore, a weaker version of stationarity is often assumed. A time series $\{r_t\}$ is said to be *weakly stationary* if both the mean of r_t and the covariance between r_t and $r_{t-\ell}$ are time-invariant, where ℓ is an arbitrary integer. Put another way, $\{r_t\}$ is weakly stationary if (a) $E(r_t) = \mu$ (i.e. constant) and (b) $\text{Cov}\,(r_t, r_{t-\ell}) = \gamma_\ell$ (i.e. only depends on ℓ and not t). To get a visual picture of this, suppose that our observed series comprises T data points, i.e. $\{r_t\}_{t=1}^{T}$. Weak stationarity then implies that the time plot of the data shows the T values fluctuating with constant variation around a fixed level. In applications, weak stationarity enables one to make inferences about future observations [40]—which is exactly what we will do in this study concerning financial forecasting and prediction.

The covariance $\text{Cov}\,(r_t, r_{t-\ell}) = \gamma_\ell$ mentioned above is actually called the lag-ℓ *autocovariance* of r_t (which will also show up in the following subsection). It has two important properties: $\gamma_0 = \text{Var}(r_t)$ and $\gamma_{-\ell} = \gamma_\ell$. The latter property follows from $\text{Cov}\left(r_t, r_{t-(-\ell)}\right) = \text{Cov}\left(r_{t-(-\ell)}, r_t\right) = \text{Cov}\,(r_{t+\ell}, r_t) = \text{Cov}\,(r_{t_1}, r_{t_1-\ell})$, where $t_1 = t + \ell$.

It is common in the finance literature to assume that an asset return series is weakly stationary. Granted that a sufficiently large amount of historical data is available—which is very often the case (except for data quoted over very long time periods, e.g. yearly data)—this assumption can be checked by visual inspection or empirically by dividing the data into subsamples and checking the consistency of the results across the different subsamples.

Now, if the time series is not stationary—which would be the case for a series of stock prices $\{p_t\}$—it can often be transformed to stationarity via one of the following techniques: (i) differencing the original data to create a new series with elements $d_i = p_i - p_{i-1}$; (ii) fitting a curve to the data (in the presence of a trend) and then modelling the residuals from that fit; (iii) taking the logarithm or square root of the series may stabilize the variance (for non-constant variance). These techniques are intended to generate series with constant location and scale. In this case of financial markets, the immediate, raw data has the form of a series of prices $\{p_t\}$, which is clearly not stationary. The series will be transformed for several reasons, a key one of which is stationarity. The transformation is a sort of differencing as mentioned here, but it is more advanced than the simple differencing in technique (i) above. More specifically, as has been elaborated upon earlier and will be used later on, we will mainly make use of so-called *logarithmic returns*, thus essentially applying both a more advanced version of technique (i) and technique (iii).

4.1.2 Correlation and Autocorrelation

Autocorrelation (or serial correlation) is a very interesting and important aspect of time series analysis—and, indeed, for our case of financial forecasting. Before turning to autocorrelation, however, we briefly recount on the usual concept of correlation.

The correlation coefficient $\rho_{x,y}$ between two random variables X and Y is defined as

$$\rho_{x,y} = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}} = \frac{E\left[(X-\mu_x)(Y-\mu_y)\right]}{E\left[X-\mu_x\right]^2 E\left[Y-\mu_y\right]^2}, \tag{4.1}$$

where μ_x and μ_y are the means of X and Y, respectively, and $E[]$ denotes the expected value. The correlation coefficient measures the strength of linear dependence between X and Y, and satisfies $-1 \leq \rho_{x,y} \leq 1$ and $\rho_{x,y} = \rho y, x$.

Now, in regards to the financial markets, only a finite amount of data is available. We might not even consider all of the available data, but rather select a time frame of a certain length (depending on the frequency of the data and the time horizon of the predictions). This means that we will be considering a *sample*. Letting this be given by $\{(x_t, y_t)\}_{t=1}^{T}$, i.e. T observations of the data (x, y), the correlation can be consistently estimated by the sample correlation

$$\hat{\rho}_{x,y} = \frac{\sum_{t=1}^{T} (x_t - \bar{x})(y_t - \bar{y})}{\sqrt{\sum_{t=1}^{T} (x_t - \bar{x})^2 \sum_{t=1}^{T} (y_t - \bar{y})^2}}, \tag{4.2}$$

where $\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x_t$ and $\bar{y} = \frac{1}{T}\sum_{t=1}^{T} y_t$ are the sample mean of X and Y, respectively.

Equipped with this knowledge, we are now ready to take on the *Autocorrelation Function* (ACF). First, consider a weakly stationary time series r_t. In regards to financial data, this could be a series of the returns of an asset; e.g. the daily return of a stock. It is interesting to investigate whether e.g. the daily return of yesterday (or one or more days further in the past) has an influence on today's return; that is, in general, whether there is any dependence between r_t and its past values r_{t-i}. This is exactly where the concept of correlation is generalized to that of autocorrelation. More specifically, the correlation coefficient between r_t and $r_{t-\ell}$ is called the lag-ℓ *autocorrelation* of r_t and is denoted by ρ_ℓ. It is defined by

$$\rho_\ell = \frac{Cov(r_t, r_{t-\ell})}{\sqrt{Var(r_t)Var(r_{t-\ell})}} = \frac{Cov(r_t, r_{t-\ell})}{Var(r_t)}. \tag{4.3}$$

The second equality assumes that the series is weakly stationary, whereby the property $Var(r_t) = Var(r_{t-\ell})$ holds.

Evidently, as follows from the definition, $\rho_0 = 1$ (the series is, of course, fully positively correlated with itself), $\rho_\ell = \rho_{-\ell}$, and $-1 \leq \rho_\ell \leq 1$. Moreover, we say that the weakly stationary series r_t is *not* serially correlated if and only if $\rho_\ell = 0 \; \forall \; \ell > 0$.

Again, if only a sample $\{r_t\}_{t=1}^{T}$ is considered, then we define the lag-ℓ *sample autocorrelation* of r_t as [2]

$$\hat{\rho}_\ell = \frac{\sum_{t=\ell+1}^{T} (r_t - \bar{r})(r_{t-\ell} - \bar{r})}{\sqrt{\sum_{t=1}^{T} (r_t - \bar{r})^2}}, \qquad 0 \leq \ell < T - 1, \tag{4.4}$$

where $\bar{r} = \frac{1}{T}\sum_{t=1}^{T} r_t$ is the sample mean.

As a whole, the statistics $\hat{\rho}_1, \hat{\rho}_2, \ldots, \hat{\rho}_{T-1}$ are called the *sample autocorrelation function* (ACF) of r_t. This plays an important role in linear time series analysis. In this study, we

[2] For finite samples, the sample autocorrelation $\hat{\rho}_\ell$ is a biased estimator of the true autocorrelation ρ_ℓ, with the bias being of the order $1/T$. [40] This can be significant if T is small. However, financial datasets are generally more than sufficiently large (except, perhaps, if one chooses to consider data quoted on e.g. a yearly basis), and so the bias will be insignificant for our purposes.

will apply it in both the usual way of analysing a given time series, but also for selecting which lagged data might be most relevant as inputs in the methods.

Significance Tests of Autocorrelation

Having calculated, say, the sample autocorrelation $\hat{\rho}_\ell$ of some time series, we are interested in whether the results are statistically significant. The autocorrelation can be tested both individually and jointly.

Testing Individual ACF:

The lag-ℓ autocorrelation ρ_ℓ of the series $\{r_t\}$ can be tested individually for a given positive integer ℓ.[40] The null hypothesis is $H_0 : \rho_\ell = 0$, and the alternative hypothesis is $H_a : \rho_\ell \neq 0$. The test statistic is

$$t = \frac{\hat{\rho}_\ell}{\sqrt{\frac{1+2\sum_{i=1}^{\ell-1}\hat{\rho}_i^2}{T}}}. \tag{4.5}$$

If $\{r_t\}$ is a stationary Gaussian series satisfying $\rho_j = 0$ for $j < \ell$, the test statistic t is asymptotically distributed as a standard normal random variable. Hence, the decision rule of the test is to reject the null hypothesis H_0 if $|t| > Z_{\alpha/2}$, where $Z_{\alpha/2}$ is the $100\,(1-\alpha/2)$th percentile of the standard normal distribution. α is the significance level of the test and is often taken as $0.05 = 5\%$.

When the sample size T is finite, $\hat{\rho}_\ell$ is a biased estimator of ρ_ℓ. The bias is of the order of $1/T$, which can be significant for small T. However, in our case of financial data, the samples will generally be sufficiently large to diminish the effect of the bias.[3]

Box-Pierce Test:

It is often slow and inadequate to test multiple autocorrelations individually, especially for some financial applications. Instead, one can test jointly that several autocorrelations of r_t are zero. More specifically, the null hypothesis $H_0 : \rho_1 = \cdots = \rho_m = 0$ (that the first m autocorrelations are zero) is tested against the alternative $H_a : \rho_i \neq 0$ for some $i \in \{1, \ldots, m\}$. Proposed by Box and Pierce [41], the test statistic is the so-called Portmanteau statistic

$$Q^*(m) = T\sum_{\ell=1}^{m}\hat{\rho}_\ell^2 \tag{4.6}$$

Under the assumption that $\{r_t\}$ is an i.i.d. sequence with certain moment conditions [40], $Q^*(m)$ is asymptotically a χ^2 random variable with m degrees of freedom.

Ljung-Box Q-test:

Ljung and Box [42] modify the above test statistic as follows to increase the power of the test in finite samples:

$$Q(m) = T(T+2)\sum_{\ell=1}^{m}\frac{\hat{\rho}_\ell^2}{T-\ell}. \tag{4.7}$$

[3] Of course, if one considered low-frequency data such as monthly prices for a company that is relatively new on the markets, then the sample size will be rather small, with only 12 observations per year.

The null hypothesis H_0 is rejected if $Q(m) > \chi^2_\alpha$, where χ^2_α denotes the $100(1-\alpha)$th percentile of a χ^2 distribution with m degrees of freedom. The p-value of the statistic $Q(m)$ is often calculated, in which case H_0 is rejected if $p \leq \alpha$, where α is the significance level.

Evidently, the test statistic depends on the number of autocorrelations m, and in practice the selection of m may affect the performance of $Q(m)$. Simulation studies suggest that the choice $m \approx \ln T$ provides better power performance [40]. However, the test is often performed for several values for m, which is also what we have done in the later analysis.

We will mainly make use of the Ljung-Box Q-test for testing the autocorrelation significance in our time series datasets.

4.1.3 White Noise

A term that appears in the time series models treated in the next sections is *white noise*. A time series r_t is called white noise if $\{r_t\}$ is a sequence of independent and identically distributed (i.i.d.) random variables with finite mean and variance. In the special case where r_t is normally distributed with zero mean and variance σ^2, the series is said to be a Gaussian white noise. Now, for a white noise series, all the ACFs are zero; i.e. there are no autocorrelations. This is, of course, the ideal, theoretical case. In practice, if all sample ACFs are close to zero (or more specifically within their two standard error limits, indicating no significant deviation from zero at the 5% level), then the series is deemed a white noise series.

4.2 Autoregressive (AR) Models

Before diving into the general case of autoregressive (AR) models we motivate the theory with a little example. Let r_t denote the return (be it daily, weekly, monthly, etc.) series for some stock, index, or other financial asset. Moreover, assume that there is a statistically significant lag-1 autocorrelation. This indicates that the lagged return r_{t-1} might be useful for predicting r_t. A simple model making use of such predictive power is

$$r_t = \phi_0 + \phi_1 r_{t-1} + a_t, \tag{4.8}$$

where $\{a_t\}$ is a white noise series with mean zero and variance σ_a^2. This model is strikingly similar in form to the classic simple linear regression model where a dependent variable (here r_t) is related to independent, or explanatory, variables (here r_{t-1}), plus an error term (here a_t). In regression jargon, we say that r_t is "regressed" on r_{t-1}. In this particular case, however, the variable r_t is regressed on a previous value of itself; hence, the model is called *auto*regressive. More specifically, the above model in (4.8) is called an autoregressive (AR) model of order 1, or simply an AR(1) model.

This simple AR(1) model implies that, given the past return r_{t-1}, the return r_t is not correlated with observations prior to $t-1$. In more mathematically rigorous terms, we have, conditional on the past return r_{t-1},

$$E\left(r_t | r_{t-1}\right) = \phi_0 + \phi_1 r_{t-1}, \tag{4.9}$$

$$\mathrm{Var}\left(r_t | r_{t-1}\right) = \mathrm{Var}(a_t) = \sigma_a^2. \tag{4.10}$$

This means, given the past return r_{t-1}, the current return r_t is centered around $\phi_0 + \phi_1 r_{t-1}$ with standard deviation σ_a. Hence, we obtain the above conclusion that, conditional on r_{t-1}, r_t is not correlated with r_{t-i} for $i > 1$.[4]

The AR(1) model is a very simple model, and clearly there are situations in which additional past observations are required. Thus, the AR(1) model can readily be generalized to the AR(p) model:

$$r_t = \phi_0 + \phi_1 r_{t-1} + \cdots + \phi_p r_{t-p} + a_t, \tag{4.11}$$

where p is a non-negative integer and $\{a_t\}$ a white noise series as before. This model says that the past p values $r_{t-i}, i = 1, \ldots, p$ jointly determine the conditional expectation $E\left(r_t | \{r_{t-i}\}_{i=1}^{p}\right)$ of r_t given the past data.

A thorough treatment of some basic properties of AR models is given in [40], [37]. The conclusions for the general AR(p) model are as follows. The mean of a stationary series is

$$E(r_t) = \frac{\phi_0}{1 - \phi_1 - \cdots - \phi_p}, \tag{4.12}$$

where it is assumed that the denominator is not zero. The associated polynomial equation of the model is

$$1 - \phi_1 x - \phi_2 x^2 - \cdots - \phi_p x^p = 0. \tag{4.13}$$

This is referred to as the *characteristic equation* of the model. If all the solutions to this equation are greater than one in modulus, then the series r_t is stationary. The inverses of the solutions are called the *characteristic roots* of the model. Thus, stationarity requires that all characteristic roots are less than one in modulus. For a stationary AR(p) series, the autocorrelation function (ACF) ρ_ℓ satisfies the difference equation

$$\left(1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p\right) \rho_\ell = 0, \qquad \text{for} \quad \ell > 0. \tag{4.14}$$

Here, we have introduced the *backward shift operator* B, which shifts the time index by one into the past. It is defined by $Br_t = r_{t-1}$, and hence $B^p r_t = r_{t-p}$. In this regard, the inverse operation is performed by the *forward shift operator* $F = B^{-1}$ given by $Fr_t = r_{t+1}$, and hence $F^p r_t = r_{t+p}$. A plot of the ACF of a stationary AR(p) model would show a mixture of damping sine and cosine patterns and exponential decays depending on the nature of its characteristic roots.

4.2.1 Order Determination for AR Models

Now, the question arises of how AR models are identified and determined in practice. Say we have a time series and want to model the conditional mean via an AR model. Which AR model should we choose? That is, what should we use for the order p of the AR model? The value of p is clearly unknown and must be specified empirically. This is referred to as the *order determination* of AR models. There are two general approaches for determining the value of p. The first approach is to use the so-called *partial autocorrelation function* (PACF), while the second uses some *information criterion* function.

Partial autocorrelation function (PACF):

[4] For a more detailed explanation, see e.g. [40].

The PACF of a stationary time series is a function of its ACF and is a useful tool for determining the order p of an AR model. Denoted $\phi_{j,j}$, the partial lag-j autocorrelation for a time series r_t is the autocorrelation between r_t and r_{t-j} after removing any linear dependence on $r_1, r_2, \ldots, r_{t-j+1}$. The interpretation is entertained by an example as follows. The lag-2 PACF $\phi_{2,2}$ shows the added contribution of r_{t-2} (i.e. the time series value two steps in the past) to r_t over the AR(1) model (which does not include the r_{t-2} term). Similarly, the lag-3 PACF $\phi_{3,3}$ shows the added contribution of r_{t-3} to r_t over the AR(2) model. Generalizing, the interpretation is thus that the lag-j PACF $\phi_{j,j}$ shows the added contribution of r_{t-j} to r_t over the AR($j-1$) model. Therefore, for an AR(p) model, the lag-p sample PACF should not be zero, but $\phi_{j,j}$ should be close to zero (we say that it cuts off) for all $j > p$. This property can be used to determine the order p, and it can be shown [40] that, for an AR(p) series, the sample PACF cuts off after p lags.

Qualitative properties of the ACF and PACF for not only the AR(p) conditional mean model but also the MA(q) and ARMA(p, q) conditional mean models (see sections 4.3 and 4.4, respectively) are given in Table 4.1.

Conditional mean model	**ACF**	**PACF**
AR(p)	Tails off gradually	Cuts off after p lags
MA(q)	Cuts off after q lags	Tails off gradually
ARMA(p,q)	Tails off gradually	Tails off gradually

Table 4.1: Qualitative properties of autocorrelation function (ACF) and partial autocorrelation function (PACF) for the conditional mean models AR(p), MA(q), and ARMA(p, q).

In conclusion, if we want to model the conditional mean of a time series with an AR(p) model, we can determine the order p by plotting the sample PACF of the time series and choosing p as the lag value at which the PACF cuts off.

Information criteria:

Information criteria are likelihood-based measures of model fit that include a penalty for complexity[5]. Different information criteria are distinguished by the form of the penalty, and can prefer different models.

Let $\log L(\theta)$ denote the value of the maximized loglikelihood objective function for a model with k parameters fit to N data points. Two commonly used information criteria are:

Akaike Information Criterion (AIC): The AIC compares models from the perspective of information entropy. The AIC for a given model is

$$\text{AIC} = -2 \log L(\theta) + 2k \tag{4.15}$$

Bayesian Information Criterion (BIC): Also known as the Schwarz information criterion, the BIC compares models from the perspective of decision theory, as measured by expected loss. The BIC for a given model is

$$\text{BIC} = -2 \log L(\theta) + k \log(N) \tag{4.16}$$

[5]As we seek the most simple, parsimonious model, complexity (specifically, the number of parameters) is penalized.

When comparing AIC or BIC values for multiple models, smaller values of the criterion are better.

As mentioned earlier, the criteria functions penalize model complexity. The penalty for each parameter used is 2 for AIC and $\ln(N)$ for BIC, as evident from the right terms in the above functions. Thus, BIC tends to select a lower AR model when the sample size N is moderate or large.

In this case of order determination for AR models, AIC or BIC values are first computed for a number of different models with different values of p. The order p is then chosen as the one giving the model with the lowest AIC or BIC.[6]

Since the above treatment of AR models mainly serves the purpose of laying the groundwork for the later ARMA and ARIMA models, we end the section here. For a more detailed description and discussion of AR models, we refer to e.g. [36], [37], [38], [39], [40].

4.3 Moving Average (MA) Models

Having laid the groundwork of AR models, we now turn to another class of simple models useful in modelling financial return series; namely, *moving-average* (MA) models. This class of models can be introduced in several ways. One approach is to treat the model as a simple extension of white noise series. In this case, however, we follow the approach in [40], where the model is treated as an infinite-order AR model with some parameter constraints.

So, consider first the AR model of infinite order:

$$r_t = \phi_0 + \phi_1 r_{t-1} + \phi_2 r_{t-2} + \cdots + a_t. \tag{4.17}$$

Such an AR model is not realistic, though, due to its infinite number of parameters. One way to make the model practical is to assume that the coefficients ϕ_i satisfy some constraints so that they are determined by a finite number of parameters. As an example of a simple special case, we could have them depend on the single parameter θ_1 via $\phi_i = -\theta_1^i$ for $i \geq 1$. This leads to the model

$$r_t = \phi_0 - \theta_1 r_{t-1} - \theta_1^2 r_{t-2} - \theta_1^3 r_{t-3} - \cdots + a_t. \tag{4.18}$$

For this model to be stationary, θ_1 must be less than one in absolute value, i.e. $|\theta_1| < 1$. If this requirement is not fulfilled, then θ_1^i will increase without bounds and the series will explode. Instead, since $|\theta_1| < 1$, we have $\theta_1^i \to 0$ as $i \to \infty$. This means that the contribution of r_{t-i} to r_t decays exponentially as i increases; i.e. that past observations play smaller and smaller roles the older they are. This is both reasonable and intuitive as the dependence of a stationary series r_t on its lagged value r_{t-i} should decay over time.

We can rewrite model (4.18) in a rather simple form. To see this, we first rewrite it as

$$r_t + \theta_1 r_{t-1} + \theta_1^2 r_{t-2} + \cdots = \phi_0 + a_t. \tag{4.19}$$

Substituting $t \to t-1$, the model for r_{t-1} is then

$$r_{t-1} + \theta_1 r_{t-2} + \theta_1^2 r_{t-3} + \cdots = \phi_0 + a_{t-1}. \tag{4.20}$$

Multiplying this equation by θ_1 and subtracting the result from the previous equation yields

[6] Sometimes the AIC and BIC formulas are scaled by a factor of the sample size N.

$$r_t + \theta_1 r_{t-1} + \theta_1^2 r_{t-2} + \cdots - \theta_1 \left(r_{t-1} + \theta_1 r_{t-2} + \theta_1^2 r_{t-3} + \cdots \right) = \phi_0 + a_t - \theta_1 \left(\phi_0 + a_{t-1} \right) \quad (4.21)$$

$$r_t = \phi_0 \left(1 - \theta_1 \right) + a_t - \theta_1 a_{t-1} \quad (4.22)$$

This result says that, apart from the constant term with ϕ_0, r_t is a weighted average of shocks a_t and a_{t-1}. For this reason—that it is a weighted *average* and that it only goes back *one* time step to $t-1$—the model is called a moving-average MA model of order 1, or simply an MA(1) model for short. Collecting $\phi_0 \left(1 - \theta_1 \right)$ into the constant c_0, the general form of an MA(1) model is

$$r_t = c_0 + a_t - \theta_1 a_{t-1}, \quad (4.23)$$

or, making use of the backward shift operator,

$$r_t = c_0 + \left(1 - \theta_1 B \right) a_t. \quad (4.24)$$

Again, $\{a_t\}$ is a white noise series. Similarly, an MA(2) model is in the form

$$r_t = c_0 + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} \quad (4.25)$$

$$= c_0 + \left(1 - \theta_1 B - \theta_2 B^2 \right) a_t. \quad (4.26)$$

Finally, we generalize to an MA(q) model

$$r_t = c_0 + a_t - \theta_1 a_{t-1} - \cdots - \theta_q a_{t-q} \quad (4.27)$$

$$= c_0 + \left(1 - \theta_1 B - \cdot - \theta_q B^q \right) a_t, \quad (4.28)$$

where $q > 0$.

We now review some properties of MA models. First, the constant term c_0 is the unconditional mean of the series, i.e.

$$E(r_t) = c_0 \quad (4.29)$$

Second, by taking the variance of (4.28) above and using the fact that a_t and a_{t-1} are uncorrelated, the variance of an MA(q) model can be shown to be

$$\mathrm{VAR}(r_t) = \left(1 + \theta_1^2 + \theta_2^2 + \cdots + + \theta_q^2 \right) \sigma_a^2. \quad (4.30)$$

Evidently, both the mean and variance are time-invariant, and hence MA models are always weakly stationary.

Moreover, regarding autocorrelation, it can be shown that the ACF for an MA(q) model cuts off after lag q. That is, the lag-q ACF is not zero, but $\rho_\ell = 0$ for all $\ell > q$. This makes the ACF a useful tool for identifying the order of an MA model, just as the PACF is useful for order determination of AR models, as described in 4.2.1. Recall that the order p of an AR model can be determined as the lag at which the PACF cuts off. Similarly, the order q of an MA model is determined as the lag at which the ACF cuts off; or, more precisely, as the lag q for which $\rho_q \neq= 0$ but $\rho_\ell = 0$ for $\ell > q$. So, order determination is performed qualitatively via the PACF for AR models and the ACF for MA models. Again, the earlier Table 4.1 gives a good overview.

The above treatment of MA models mainly serves the purpose of laying the theoretical foundation for the more advanced ARMA models (and their extensions). For a further examination of MA models, we refer to e.g. [36], [37], [38], [39], [40].

4.4 Autoregressive Moving Average (ARMA) Models

So far, we have considered the simple AR and MA models. In some applications, these become too cumbersome because a complex, high-order model with many parameters may be needed to adequately describe the dynamic structure of the data. Indeed, in our case, these models will not be able to encompass the complexity of financial data which we consider here. For this reason, we will not make practical use of AR and MA models individually. But the theoretical treatment above is far from in vain; to overcome the methods' individual difficulties and shortcomings, they are combined into the so-called *autoregressive moving-average* (ARMA) models which we introduce here. By combining the ideas of AR and MA models into a compact form, ARMA models keep the number of parameters small. Still, however, the chance is low of using ARMA models for financial return series [40], but their theory lays some important groundwork for even more advanced methods.[7]

We begin by studying the simplest case of a combination comprising an AR(1) and MA(1) model, which is denoted an ARMA$(1,1)$ model. A time series r_t follows an ARMA$(1,1)$ model if it satisfies

$$r_t - \phi_1 r_{t-1} = \phi_0 + a_t - \theta_1 a_{t-1}, \tag{4.31}$$

where $\{a_t\}$ is a white noise series. The left-hand and right-hand sides are the AR and MA components, respectively. The constant term is ϕ_0. It should be noted that, for the model to be meaningful, we require $\phi_1 \neq \theta_1$; otherwise, it can be shown [40] that there is a cancellation in the equation and the process reduces to a white noise series. In addition, regarding autocorrelation, it can be shown that neither the PACF nor the ACF of an ARMA(1,1) model cut off at any finite lag—as opposed to the PACF for the individual AR model and the ACF for the individual MA model, as mentioned in section 4.2 and 4.3, respectively. Instead, both the ACF and PACF exhibit exponential decay starting at lag 2.

Now, moving on to the general case, an ARMA(p,q) model has the form

$$r_t = \phi_0 + \sum_{i=1}^{p} \phi_i r_{t-i} + a_t - \sum_{i=1}^{q} \theta_i a_{t-i}, \tag{4.32}$$

where $\{a_t\}$ is a white noise series, and p and q are non-negative integers. Of course, the individual AR and MA models are special cases of the ARMA(p,q) model. Using the backward shift operator defined above, the model can be written

$$(1 - \phi_1 B - \cdots \phi_p B^p)\, r_t = \phi_0 + (1 - \theta_1 B - \cdots \theta_p B^p)\, a_t, \tag{4.33}$$

The left-hand side polynomial $(1 - \phi_1 B - \cdots \phi_p B^p)$ is the AR polynomial of the model, while $(1 - \theta_1 B - \cdots \theta_p B^p)$ is the MA polynomial. We require there to be no common factors between the AR and MA polynomials; otherwise the order $(p, q$ of the model can be reduced [40]. Like the pure AR model, the AR polynomial introduces the characteristic equation of the ARMA model. If all of the solutions of the characteristic equation are less than one in absolute value, then the ARMA model is weakly stationary. In this case, the unconditional mean of the model is

$$E(r_t) = \frac{\phi_0}{1 - \phi_1 - \cdots - \phi_p} \tag{4.34}$$

[7] In this regard, the concept of ARMA models is highly relevant in volatility modelling [40].

4.4.1 Order Determination for ARMA models

In sections 4.2 and 4.3, respectively, we discussed how the PACF could be used to determine the order p of an AR model and how the ACF could do so for MA(q) models. However, this is not the case for ARMA models; since the ACF and PACF for an ARMA(p, q) model decay (and do not cut off at any finite lag), neither are informative in order determination for ARMA models.

How, then, do we go about the problem of order determination? One approach is to use the extended autocorrelation function (EACF) [40]. Although the basic idea is relatively simple, the derivation and theory is quite involved. We will not pursue this approach here, but rather turn to the information criteria-based approach presented in section 4.2.1. This section introduced two information criteria commonly used for model selection and order determination; namely, the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC). These criteria functions penalize model complexity; the penalty for each parameter used is 2 for AIC and $\ln(N)$ for BIC, as evident from the formulas in section 4.2.1.

In this case of order determination for ARMA models, one computes, for some pre-specified positive integers P and Q, AIC or BIC values for a number of ARMA(p, q) models, where $0 \leq p \leq P$ and $0 \leq q \leq Q$. Since lower AIC or BIC is better (indicating a simpler, more parsimonious model, which is what we seek), we select the model with the minimum AIC or BIC.

Then, once an ARMA(p, q) model is specified, i.e. its order has been determined, its parameters can be estimated by either the conditional or exact likelihood method. Going into further theoretical detail on this matter is beyond the scope of this study, and we refer to e.g. [36], [37], [38], [39], [40]. In addition, residuals can be used to perform model diagnostics and assess the adequacy of the fitted model, as is described and done in the practical applications in the later analysis.

For now, we will proceed with the question of how a specified ARMA model is actually used to provide forecasts of future values of the time series.

4.4.2 Forecasting with ARMA Models

In some applications, we are only interested in predictions one period, or time step, ahead. In other cases, forecasts multiple periods ahead are very interesting and useful. We begin with the former case and later generalize to the latter.

Denote the forecast origin (the point beyond which we have no actual information and from which we must therefore forecast) by h and the available information by F_h. The 1-step ahead forecast of r_{h+1} can easily be obtained from the ARMA(p, q) model as

$$\hat{r}_h(1) = E\left(r_{h+1}|F_h\right) = \phi_0 + \sum_{i=1}^{p} \phi_i r_{h+1-i} - \sum_{i=1}^{q} \theta_i a_{h+1-i}. \tag{4.35}$$

The "1" in parenthesis in $\hat{r}_h(1)$ indicates that it is a 1-step ahead forecast, and the expression reads that this forecast is given by the expectation of r_{h+1} conditioned on the available information F_h. The associated forecast error is

$$e_h(1) = r_{h+1} - \hat{r}_h(1) = a_{h+1} \tag{4.36}$$

The variance of the 1-step ahead forecast error is

$$\mathrm{Var}\left[e_h(1)\right] = \sigma_a^2. \tag{4.37}$$

Now, generalizing to the case of predictions multiple periods in the future, the ℓ-step ahead forecast is

$$\hat{r}_h(\ell) = E\left(r_{h+\ell}|F_h\right) = \phi_0 + \sum_{i=1}^{p} \phi_i \hat{r}_h(\ell - i) - \sum_{i=1}^{q} \theta_i a_h(\ell - i), \tag{4.38}$$

where $\hat{r}_h(\ell - i) = r_{h+\ell-i}$ if $\ell - i \leq 0$, and $a_h(\ell - i) = 0$ if $\ell - i > 0$, and $a_h(\ell - i) = a_{h+\ell-i}$ if $\ell - i \leq 0$. Thus, the multi-step ahead forecasts of an ARMA model can be computed recursively. The associated ℓ-step ahead forecast error is

$$e_h(\ell) = r_{h+\ell} - \hat{r}_h(\ell), \tag{4.39}$$

which can be computed easily via the so-called *impulse response function* $\{\psi_i\}$ of the ARMA model, which is introduced in the following section on different ARMA model representations.

4.4.3 Three Representations for ARMA Models

The three representations of an ARMA model that will be introduced below serve three different purposes. Knowing these representations can lead to a better understanding of the model.

The first representation is the ARMA(p, q) model in (4.33), i.e.

$$\left(1 - \phi_1 B - \cdots \phi_p B^p\right) r_t = \phi_0 + \left(1 - \theta_1 B - \cdots \theta_p B^p\right) a_t, \tag{4.40}$$

which can be written as

$$\phi(B) r_t = \phi_0 + \theta(B) a_t. \tag{4.41}$$

The merits of this representation is its compactness and usefulness in both parameter estimation and recursive computation of multi-step ahead forecasts of r_t.

For the other two representations, we will use long division of the two polynomials. Given the two polynomials $\phi(B) = 1 - \sum_{i=1}^{p} \phi_i B^i$ and $\theta(B) = 1 - \sum_{i=1}^{q} \theta_i B^i$, we obtain by long division that

$$\frac{\theta(B)}{\phi(B)} = 1 + \psi_1 B + \psi_2 B^2 + \cdots \equiv \psi(B), \tag{4.42}$$

where the impulse response function mentioned in the previous section appears, and

$$\frac{\phi(B)}{\theta(B)} = 1 + \pi_1 B + \pi_2 B^2 + \cdots \equiv \pi(B). \tag{4.43}$$

As an example, for $p = q = 1$, we have $\phi(B) = 1 - \phi_1 B$ and $\theta(B) = 1 - \theta_1 B$, giving

$$\psi(B) = \frac{1 - \theta_1 B}{1 - \phi_1 B} = 1 + \left(\phi_1 - \theta_1\right) B + \phi_1 \left(\phi_1 - \theta_1\right) B^2 + \phi_1^2 \left(\phi_1 - \theta_1\right) B^3 + \cdots, \tag{4.44}$$

$$\pi(B) = \frac{1 - \phi_1 B}{1 - \theta_1 B} = 1 - \left(\phi_1 - \theta_1\right) B - \phi_1 \left(\phi_1 - \theta_1\right) B^2 - \phi_1^2 \left(\phi_1 - \theta_1\right) B^3 - \cdots. \tag{4.45}$$

From the definition we see that $\psi(B)\pi(B) = 1$. Then, making use of the fact that the backshift operator B satisfies $Bc = c$ for any constant c (because the value of a constant is time-invariant), we have

$$\frac{\phi_0}{\theta(1)} = \frac{\phi_0}{1-\theta_1-\cdots-\theta_q}, \tag{4.46}$$

$$\frac{\phi_0}{\phi(1)} = \frac{\phi_0}{1-\phi_1-\cdots-\phi_p}. \tag{4.47}$$

AR Representation

Now, using the second result of long division above, the ARMA(p,q) model can be written

$$r_t = \frac{\phi_0}{1-\theta_1-\cdots-\theta_q} + pi_1 r_{t-1} + \pi_2 r_{t-2} + \pi_3 r_{t-3} + \cdots + a_t \tag{4.48}$$

This representation shows the dependence of the current return r_t on the past returns r_{t-i}, where $i > 0$. This is also why it is called an AR representation. The coefficients $\{\pi_i\}$ are referred to as the π-weights of an ARMA model. To show that the contribution of the lagged value r_{t-i} to r_t is diminishing as i increases, the π_i coefficient should decay to zero as i increases. An ARMA model having this property is said to be *invertible*. For a pure AR model (i.e. $q = 0$), $\theta(B) = 1$ so that $\pi(B) = \phi(B)$, which is a finite-degree polynomial. Thus, $\pi_i = 0$ for $i > p$, and the model is invertible. For other ARMA models ($q \neq 0$), a sufficient condition for invertibility is that all the zeros of the polynomial $\phi(B)$ are greater than unity in modulus. For example, consider the simple MA(1) model (an ARMA model with $p = 0$ and $q = 1$) $r_t = (1-\theta_1 B)\, a_t$. The zero of $\phi(B) = 1-\phi_1 B$ is $B = 1/\theta_1$. Therefore, an MA(1) model is invertible if $|1/\theta_1| > 1$, which is equivalent to $|\theta_1 < 1$.

This AR representation tells us that an invertible ARMA(p,q) series r_t is a linear combination of the current shock a_t and a weighted average of the past values. The weights decay exponentially for more remote past values.

MA Representation

Considering the original representation $\phi(B)r_t = \phi_0 + \theta(B)a_t \Longleftrightarrow r_t = \frac{\phi_0}{\phi(B)} + \frac{\theta(B)}{\phi(B)}a_t$ and using now the first result of long division above, the ARMA(p,q) model can be written

$$r_t = \mu + a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + \cdots = \mu + \psi(B)a_t, \tag{4.49}$$

where $\mu = E(r_t) = \dfrac{\phi_0}{1-\phi_1-\cdots-\phi_p}$ is the unconditional mean. This representation explicitly shows the impact of the past shocks a_{t-i} for $i > 0$ on the current return r_t, which is why it is called the MA representation. The coefficients $\{\psi_i\}$ are the aforementioned *impulse response function* of the ARMA model. For a weakly stationary series, the ψ_i coefficients decay exponentially as i increases. This is also intuitive and understandable as the effect of the shock a_{t-i} on the return r_t should diminish over time. Thus, for a stationary ARMA model, the shock a_{t-i} does not have a permanent impact on the series.

As mentioned earlier, the ℓ-step ahead forecast and its error variance can be expressed via the impulse response function appearing in the MA representation. At the forecast origin h, we have the shocks $a_t, a_{t-1}, \ldots$. Therefore, the ℓ-step ahead forecast is

$$\hat{r}_h(\ell) = \mu + \psi_\ell a_h + \psi_{\ell+1} a_{h-1} + \cdots, \tag{4.50}$$

with associated forecast error

$$e_h(\ell) = a_{h+\ell} + \psi_1 a_{h+\ell-1} + \cdots + \psi_{\ell-1} a_{h+1}. \tag{4.51}$$

Consequently, the variance of the ℓ-step ahead forecast error is

$$\mathrm{Var}\left[e_h(\ell)\right] = \left(1 + \psi_1^2 + \cdots \psi_{\ell-1}^2\right) \sigma_a^2. \tag{4.52}$$

This is a nondecreasing function of the forecast horizon ℓ; i.e., the further into the future we forecast, the larger becomes the uncertainty on our forecasts.

Finally, the MA representation shows another feature of a stationary time series. As mentioned above, the stationarity implies that ψ_i decays exponentially and thus approaches zero as $i \Leftarrow \infty$. Hence, by the forecast formula (4.50), we have $\hat{r}_h(\ell) \Leftarrow \mu$ as $\ell \Leftarrow \infty$. Since $\hat{r}_h(\ell)$ is the conditional expectation of $r_{h+\ell}$ at the forecast origin h, the result says that, in the long term, the return series is expected to approach its mean. We say that the series is mean-reverting. This is also empirically evident from our practical applications of ARMA models to be shown later.

4.5 Autoregressive Integrated Moving Average (ARIMA) Models

Above, we considered ARMA models. For such models to be weakly stationary, we required the solutions of their characteristic equation to be less than one in absolute value. Now, if we allow the polynomial to have 1 as a characteristic root, then the model becomes an *autoregressive integrated moving-average* (ARIMA) model. An ARIMA model is said to be unit-root nonstationary because its AR polynomial has a unit root.

Now, consider a time series y_t. If the change, or difference series $c_t = y_t - y_{t-1} = (1 - B)\, y_t$ follows a stationary and invertible ARMA(p, q) model, then the series y_t is said to be an ARIMA$(p, 1, q)$ process. As mentioned earlier, financial price series are commonly believed to be nonstationary. However, the logarithmic return series $r_t = \log p_t - \log p_{t-1}$ is stationary. In this case, we difference two logarithms, whereby the logarithmic price series $\log p_t$ is unit-root nonstationary and hence can be treated as an ARIMA process.

This idea of transforming a nonstationary series to a stationary one is called *differencing*, a concept we also touched upon in a previous section. The change series $c_t = y_t - y_{t-1} = (1 - B)\, y_t$ is called the first differenced series of y_t. But a time series y_t may contain multiple roots and needs to be differenced multiple times to become stationary. For instance, if both y_t and its first differenced series c_t are unit-root nonstationary, but $s_t = c_t - c_{t-1} = y_t - 2y_{t-1} - yt - 2$ is weakly stationary, then y_t has double roots, and s_t is the second differenced series of y_t. If s_t follows an ARMA(p, q) model, then y_t is an ARIMA$(p, 2, q)$ process. Evidently, if y_t and its first $d - 1$ differenced series are unit-root nonstationary, and its d-th differenced series is weakly stationary and follows an ARMA(p, q) model, then y_t is an ARIMA(p, d, q) process.

4.6 Autoregressive Moving Average Models with Exogenous Variables (ARMAX)

So far, we have considered cases where the current value r_t is modelled via its own past values and shocks. However, it might be that other variables, or covariates, have an influence and thus possess potential predictive power in the modelling of r_t. Such covariates are called external or

exogenous variables, and their inclusion extends the ARMA (ARIMA) model to the ARMAX (ARIMAX) model.

Recall the general ARMA(p, q) model

$$r_t = \phi_0 + \sum_{i=1}^{p} \phi_i r_{t-i} + a_t - \sum_{i=1}^{q} \theta_i a_{t-i}. \tag{4.53}$$

Letting the exogenous series be x_t (we consider the general case of k such series), the ARMAX(p, q) model is

$$r_t = \phi_0 + \sum_{i=1}^{p} \phi_i r_{t-i} + \sum_{h=1}^{k} \beta_k x_{tk} + a_t - \sum_{i=1}^{q} \theta_i a_{t-i}. \tag{4.54}$$

The coefficient β_k measures the linear effect of the kth exogenous series x_t on the original series r_t. x_{tk} is the value of the kth exogenous, time-varying predictor at time t. Of course, it would also be possible to include lagged values of the exogenous series.

As a practical example, consider the modelling of the return r_t of a stock. We could model this return with an ARMA model, using solely lagged returns and shocks. This may or may not lead to good results and an adequate model. Now, if the company was, say, an exporter of oil, then the oil price would be a plausible exogenous covariate, potentially improving the model and its power to better forecast future stock prices (or, more precisely, future returns). Moreover, if the company exported to other countries, then exchange rates between currencies would likewise be plausible exogenous covariates.

4.7 Conditional Heteroscedastic Models

The previous sections concerned the modelling of the return of an asset. In this section and onwards we turn to study some statistical and econometric methods for modelling the *volatility* of an asset return. These are referred to as conditional heteroscedastic[8] models.

Volatility plays a crucial role in finance and trading. Referring to the standard deviation of an asset's return, volatility is a measure of how much the return varies. For this reason, volatility is ever so closely connected to the concept of *risk*: Are we gamblers who want to risk large losses at the prospect of large potential gains? Or do we play it safe, keeping the risk low and settling for smaller returns? But volatility also has many other financial applications than risk management; for example, modelling the volatility of a time series can improve the efficiency in parameter estimation and the accuracy in interval forecasting [40]. In addition, the volatility of a market has become a financial instrument itself, enabling people to trade and speculate in it like a normal stock or index.

Volatility has a number of characteristics commonly seen in asset returns [40]. First, volatility tends to cluster and may be high for certain time periods and low for other periods. Second, volatility evolves over time in a continuous manner and seldom "jumps." Third, it does not diverge to infinity but varies within some fixed range, which, in statistical terms, means that it is often stationary. Fourth and finally, volatility seems to react differently to a large drop in price

[8]Heteroscedasticity, the absence of homoscedasticity, is a statistical term describing a collection of random variables that contains sub-populations with different variabilities (e.g. variances or other measures of statistical dispersion). As an example, take the variable in question here: the volatility/variance of asset returns. Assuming homoscedasticity entails that the variation of the returns is constant in time. But, as we show in 6.1.4, this is far from the truth; not only does the variability of returns (volatility) change in time, it also gathers in clusters (the "sub-populations").

than a large increase—the so-called *leverage* effect. These properties play an important role in volatility modelling. The following subsections explore different models of increasing detail and complexity.

4.7.1 ARCH Models

Developed by Robert Engle [43], autoregressive conditional heteroscedastic (ARCH) models are used in the modelling of observed time series whenever there is reason to believe that the error terms will have a characteristic size, or variance. In particular, ARCH models assume the variance of the current shock, or innovation[9], to be a function of the actual sizes of the previous time periods' shocks.

The basic idea of ARCH models is that (a) the shock a_t of an asset return is serially uncorrelated (i.e. no autocorrelation), but dependent, and (b) the dependence of a_t can be described by a simple quadratic function of its lagged values. More specifically, an ARCH(m) assumes that the shocks a_t are split into a stochastic piece ϵ_t and a time-dependent standard deviation σ_t characterizing the typical size of the terms so that

$$a_t = \sigma_t \epsilon_t, \tag{4.55}$$

where $\{\epsilon_t\}$ is a sequence of independent and identically distributed (i.i.d.) random variables with mean zero and unit variance. The variance (squared standard deviation) series σ_t^2 is then modelled by

$$\sigma_t^2 = \alpha_0 + \alpha_1 a_{t-1}^2 + \cdots + \alpha_m a_{t-m}^2 = \alpha_0 + \sum_{i=1}^{m} \alpha_i a_{t-i}^2, \tag{4.56}$$

where $\alpha_0 > 0$ and $\alpha_i \leq 0$ for $i > 0$. The coefficients α_i must satisfy some regularity conditions to ensure that the unconditional variance of a_t, i.e. σ_t^2, is finite. In practice, ϵ_t is often assumed to follow the standard normal distribution, a standardized Student-t distribution, or a generalized error distribution.

Now, from the structure of the above model, it is seen that large past squared shocks $\left\{a_{t-i}^2\right\}_{i=1}^{m}$ imply a large conditional variance σ_t^2 for the current shock, or so-called innovation, a_t. Consequently, a_t tends to assume a large value (in modulus). Hence, under the ARCH framework, large shocks tend to be followed by another large shock. (Of course, this is not for certain; a large variance does not necessarily produce a large realization, but only says that the probability of obtaining a large value is greater than for a smaller variance.) This theoretical feature agrees very well with the volatility clusterings observed in financial asset returns.[40]

4.7.2 GARCH Models

In the case of ARCH models above, we assumed an MA-type model for the error variance σ_t^2. If instead an autoregressive moving-average (ARMA) model is assumed for σ_t^2, then the model is a *generalized autoregressive conditional heteroscedastic* (GARCH) model [45].

For a logarithmic return series r_t, let $a_t = r_t - \mu_t$ be the innovation at time t. Then a_t follows a GARCH(m, s) model if

[9] In time series analysis, the shock, or innovation, is the difference between the observed value of a variable at time t and the optimal forecast of that value based on information available prior to time t. Thus, for an asset return r_t, the shock is defined by $a_t = r_t - \mu$, where μ is the mean return conditioned on all observations prior to time t.

$$a_t = \sigma_t \epsilon_t, \tag{4.57}$$

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{m} \alpha_i a_{t-i}^2 + \sum_{j=1}^{s} \beta_j \sigma_{t-j}^2. \tag{4.58}$$

Again, $\{\epsilon_t\}$ is a sequence of independent and identically distributed (i.i.d.) random variables with mean zero and unit variance, and $\alpha_0 > 0$, $\alpha_i \leq 0$, and $\beta_j \leq 0$, where it is furthermore understood that $\alpha_i = 0$ for $i > m$ and $\beta_j = 0$ for $j > s$. Moreover, $\sum_{i=1}^{\max(m,s)} (\alpha_i + \beta_j) < 1$. This constraint implies that the unconditional variance of a_t is finite, whereas its conditional variance σ_t^2 evolves over time. The α_i and β_j are the ARCH and GARCH parameters, respectively. m and s are the corresponding parameters denoting the order(s) of the model. Of course, if $s = 0$, then the above GARCH model (4.58) reduces to an ARCH(m) model.

The strengths and weaknesses of GARCH models can be seen by focusing on the simplest case and then generalizing. The simple GARCH(1,1) model has $\sigma_t^2 = \alpha_0 + \alpha_1 a_{t-1}^2 + \beta_1 \sigma_{t-1}^2$, $0 \leq \alpha_1, \beta_1 \leq 1$, $(\alpha_1 + \beta_1) < 1$. First, a large a_{t-1}^2 or σ_{t-1}^2 evidently gives rise to a large σ_t^2. In turn, via (4.57), this large σ_t^2 results in a large a_t^2. Thus, a large past shock a_{t-1}^2 tends to be followed by another large shock a_t^2, generating the well-known behaviour of volatility clustering observed in financial time series. Second, it can be shown [40] that, similar to ARCH models, the GARCH(1,1) process has heavier tails than the normal distribution. This makes it a better candidate for modelling financial return series because these have been shown empirically to exhibit heavy-tailed distributions [46]. Third, the GARCH model provides a simple parametric function that can be used to describe the volatility evolution. As for weaknesses, the GARCH model encounters the same problems as the ARCH model. For instance, it responds equally to positive and negative shocks.[10] In addition, empirical studies of high-frequency financial data indicate that the tail behaviour of GARCH models—although heavy as it is—is still too short even with shocks modelled by standardized Student-t [40].

4.7.3 IGARCH Models

Just as ARMA models became ARIMA models in the case of a unit root in the AR polynomial, so do GARCH models become *integrated generalized autoregressive conditional heteroscedastic* (IGARCH) models. Similar to ARIMA models, a key feature of IGARCH models is that the impact of past squared shocks a_{t-i} for $i > 0$ on a_t is persistent.

The IGARCH model is just a restricted GARCH model, the only difference being the unit-root condition

$$\sum_{i=1}^{m} \alpha_i + \sum_{j=1}^{s} \beta_j = 1. \tag{4.59}$$

In the simplest case of $m = s = 1$, this condition becomes $\alpha_1 + \beta_1 = 1$. The IGARCH(1,1) model can thus be written

$$a_t = \sigma_t \epsilon_t, \tag{4.60}$$

$$\sigma_t^2 = \alpha_0 + (1 - \beta_1)\, a_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \tag{4.61}$$

[10] An equal response to positive and negative shocks might sound plausible at first. However, negative returns have been observed to increase the future volatility by a larger amount than positive returns of the same magnitude [44]. This is called the *leverage effect* and is included in extensions to the GARCH model such as the NGARCH model or the EGARCH model.

where $\{\epsilon_t\}$ are i.i.d. random variables with mean zero and unit variance as before, and $0 < \beta_1 < 1$.

4.7.4 EGARCH Models

The conditional variance models described assume an equal response to positive and negative shocks. Thus, they do not take into account the aforementioned leverage effect observed in many financial time series. A model that does account for this effect is the exponential GARCH (EGARCH) model. It is a variant that models the logarithm of the conditional variance process whilst also including additional leverage terms to capture asymmetry in volatility clustering.

The simple case of an EGARCH(1,1) model can be written

$$r_t = \mu + a_t \tag{4.62}$$

$$a_t = \sigma_t \epsilon_t \tag{4.63}$$

$$\log \sigma_t^2 = \alpha_0 + \gamma_1 \log \sigma_{t-1}^2 + \alpha_1 \left[\frac{|a_{t-1}|}{\sigma_{t-1}} - E\left[\frac{|a_{t-1}|}{\sigma_{t-1}} \right] \right] + \xi_1 \frac{|a_{t-1}|}{\sigma_{t-1}}. \tag{4.64}$$

The form of the expected value in the expression depends on the distribution of the innovation process ϵ_t. Typically, either a Gaussian or Student's t is chosen.

In our analysis in Chapter 6, we will mainly make use of the GARCH and EGARCH models.

4.7.5 Conditional Heteroscedasticity Tests

A time series can be tested for conditional heteroscedasticity (autocorrelation in the squared series, or so-called volatility clustering) in a more quantitative way than just inspecting the ACF and PACF plots of the squared series. Below, we present two ways to do it.

Box-Ljung Q-test:

First, conditional heteroscedasticity can be tested for by means of the Ljung-Box Q-test introduced in section 4.1.2. There, we considered the autocorrelation of the return series. In this case, when testing for ARCH effects[11], the Ljung-Box Q-test is conducted on the squared series. See section 4.1.2 for the formulae, hypotheses, decision rules, etc.

Engle's ARCH test:

An alternative test for conditional heteroscedasticity is Engle's ARCH test [43], a Lagrange multiplier test for assessing the significance of ARCH effects.

We consider the residual series $e_t = r_t - \hat{\mu}_t$, where $\hat{\mu}_t$ is the mean. If all autocorrelation in the original series r_t is accounted for in the conditional mean model (e.g. an ARIMA model), then the residuals are uncorrelated with mean zero. However, the residuals can still be serially dependent.

The alternative hypothesis in Engle's ARCH test is autocorrelaton in the squared residuals, as given by the regression

$$H_a : e_t^2 = \alpha_0 + \alpha_1 e_{t-1}^2 + \cdots + \alpha_m e_{t-m}^2 + u_t, \tag{4.65}$$

[11] A time series exhibiting conditional heteroscedasticity (autocorrelation in the squared series) is said to have *autoregressive conditional heteroscedastic* (ARCH) effects.

where u_t is white noise. The null hypothesis H_0 is that of zero autocorrelation in the squared series (no ARCH effects), i.e.

$$H_0 : \alpha_0 = \alpha_1 = \cdots = \alpha_m = 0. \tag{4.66}$$

The test requires the specification of the lag m. One way to choose m is to compare loglikelihood values for difference choices of m, using e.g. the Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC).

The test can be generalized to GARCH effects by noting that a GARCH(p, q) model is locally equivalent to and ARCH$(p + q)$ model. This suggests also considering values $m = p + q$ for reasonable choices of p and q.

The test statistic is the Lagrange multiplier statistic TR^2, where T is the sample size and R^2 the coefficient of determination from fitting the ARCH(m) model for m lags via regression.

Chapter 5

Machine Learning Methods

This chapter gives a detailed description and discussion of the theoretical framework behind the selected machine learning methods applied in this study. The particular choice of methods is based mainly on the historical and contemporary research in the literature as well as our own academic experience with the methods and our ideas for their applicability to financial forecasting.

5.1 Introduction

Generally, machine learning methods are used to obtain some adaptive model of a data set—be it spatial or a time series as well as continuous or discrete. The approach is to tune the parameters of the model by use of a subset, called the *training set*, of the available data. The data comprise a certain number of observations, each having the same finite dimension. This is referred to as the input data. Depending on the problem and the method used for solving it, the data sometimes also include a desired value, the *target*, for each observation. The result of the machine learning algorithm can then be expressed as a function that takes as argument the input data and then generates an *output* of the same type as the potential target data. The precise form of this function, as defined by the parameters governing it, is determined during the *training* phase, or *learning* phase, on the basis of the training data.

The model parameters are tuned so as to minimize a certain measure of the deviation, or error, between the output and the target—granted, of course, that target data are available for the given problem. Here, one distinguishes between *supervised learning* and *unsupervised learning*; the data in the former type include target values, whereas the data in the latter type does not. Thus, the goal in unsupervised learning is either to discover groups of similar examples within the data (called *clustering*), or to determine the distribution of data within the input space (called *density estimation*), or to project the data from a high-dimensional space down to two or three dimensions (*visualization*). We are concerned with supervised learning, since target data (stock prices) are available.

After the model has been trained, it can then be used on new, previously unseen data—referred to as the *test* set—for the purpose of either categorizing/classifying these new points, or—highly relevant for our study—forecasting their future target values. The ability of the model to correctly categorize or predict new examples is known as *generalization*, and this is of the utmost importance. There is also another distinction present here; cases in which the aim is to assign each input to one of a finite number of discrete categories are called *classification* problems; on the other hand, if the desired output consists of one or more continuous variables,

then the task is called *regression*. We will consider both classification and regression problems. The latter type might be the most obvious, since the ultimate goal is to predict continuous stock prices. However, we will also consider some of the problems as classification tasks; this approach can be pursued e.g. by slightly readjusting our expectations for the forecasts in the sense that we wish to predict not the value of the price but instead whether it will move up or down. Another application of classification could be to the problem of categorizing stocks into e.g. "good" and "bad" investments based on a number of criteria defined in the input data.

5.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a class of machine learning algorithms that draw inspiration from biological neural systems. Of course, the human brain is far more complex than what our current level of computing power allows us to program on computers, and ANNs merely attempt to replicate some of the most basic behavioral and adaptive features of biological neural networks.

To lay the groundwork, this section begins with a description of the methodology behind feedforward networks and the steepest descent backpropagation algorithm used for network training. We also discuss the merits and shortcomings of several other training algorithms, comparing them to what is required for our particular task in order to choose the most suitable algorithm. Next, we proceed with a description of the actual type of neural network used in this study, namely the nonlinear autoregressive network with exogenous (external) inputs (NARX). Finally, we discuss the general strengths, weaknesses, and problems of artificial neural networks.

5.2.1 Feedforward Networks

An Artificial Neural Network (ANN) is a layered network of interconnected neurons, or nodes. The network consists of an input layer, a number of hidden layers, and an output layer. Each layer consists of a number of neurons, each of which is connected to the other neurons in the subsequent layer, as shown in Figure 5.1 below. In biological terms, these connections are called synapses.

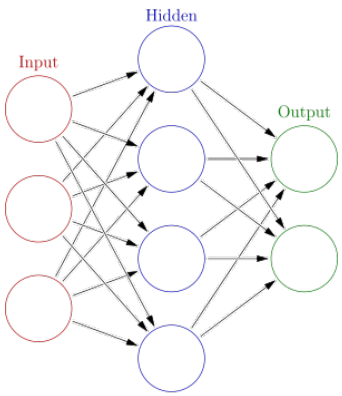

Figure 5.1: Artificial Neural Network structure. The example shown has three input neurons, one hidden layer with five neurons, and two output neurons.

To understand how neural networks work we must consider the case that some input, or

stimulus, is presented to it. Let the ith input signal be denoted by x_i. The connections then transfer these signals into the neurons in the hidden layer. Each connection is ascribed a certain weight w_{ij}, representing the importance of the particular input.

We now find ourselves in the hidden layer. Each neuron in this layer has received some signals from all of the input nodes. The sum of weighted inputs $\sum_i w_i x_{ij}$ is then calculated in each hidden neuron and passed on as argument to the activation function ϕ, which yields a signal y_j for the jth hidden neuron. Now, a variety of different activation functions can be used, however, most of them limits the range of the neuron's output to the interval $[0, 1]$ or $[-1, 1]$. Among these we find e.g. the threshold or step function, which is either 1 or 0, and the piecewise linear function, which is also 1 or 0 but with an added linear segment in between. Sometimes, a pure linear function is used, which does not limit the output range. Common choices for activation functions are the hyperbolic tangent and the sigmoid, as defined, respectively, by:

$$\phi(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{5.1}$$

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{5.2}$$

These squash the range into the intervals $[-1, 1]$ and $[0, 1]$, respectively. The choice of activation function naturally depends on which function makes most sense to use with respect to the particular problem and the corresponding data.

The output of the jth hidden neuron is then given by $y_j = \phi(s_j)$, where s_j denotes the sum of weighted inputs coming into the jth hidden neuron. Now, if there is more than one hidden layer,[1] the signals y_j from the first hidden layer are then transferred along similar connections, with weights ascribed to them, to the neurons in the subsequent hidden layers. Again, the sums of weighted signals are calculated and used as arguments in whichever activation function is chosen. Finally, the signals out of the neurons in the last hidden layer are then transferred along connections to a number of output neurons. For each output neuron, k, the sum of weighted signals from the neurons in the last hidden layer is calculated. Letting the signals from the hidden neurons be the y_j from before, the output z_k from the kth output neuron is computed as $z_k = \phi(s_k)$, where s_k denotes the mentioned sum (with the proper weights between hidden and output neurons). In this way, the neural network processes the given information by transferring the initial input signals through the layers of the network until finally giving a number of outputs. This direction of flow is also the reason that such networks are classified as *feed-forward* networks.

5.2.2 Network Training and Learning Algorithms

So far, we have described how the network processes some input and finally yields some output. However, the question arises: "What now?"

We have a large amount of input data, often denoted the *patterns*, as well as some desired, target output. It is these input patterns that are fed to the network, after which we obtain some computed output. This output will often be somewhat different from the desired, target output. But our goal, what we would like to accomplish, is to get the computed output to be as close as possible to the target output. For this purpose we must *train* the network; that is, adjust the connection weights between neurons to minimize a measure of the error. There are several ways to do this, and under one name they are called *training* or *learning algorithms*. The most common one of these—standard Steepest Descent Backpropagaton—is considered in

[1] There could also be zero hidden layers, in which case the network would be a single-layer network. The network described, i.e. with one or more hidden layers, is a multi-layer network.

the following subsection. After that, we give an overview and discussion of a wide variety of extensions and other training algorithms.

Backpropagation (Steepest Descent)

The simplest and most common learning algorithm is *backpropagation*. As the name suggests, the basic idea is to propagate output error signals backwards through the network and adjust the weights.[2] Qualitatively, the process can be summarized as follows:

1. Feed an input pattern from the training data into the ANN.
2. Run it through the layers of the network to obtain the computed output.
3. Calculate the error between the resulting, computed output and the actual, desired output.
4. Propagate the errors backwards through the network and tweak the weights of all the connections in the direction (positive or negative) that minimizes the error of the output.

This process is repeated over a set of training data. The tweaking, or adjustment, of the weights is performed using a formula called the *Delta Rule*, which is based on the principle of gradient descent. As for the error measure to be minimized, this is often taken as the sum (over the patterns h) of squared errors:

$$SSE = \frac{1}{2}\sum_{h}^{M} E_h = \frac{1}{2}\sum_{h}^{M} \left(z_{t_h} - z_{p_h}\right)^2 , \tag{5.3}$$

where z_{t_h} and z_{p_h} are the desired/target output (associated to the input data) and the predicted output (assuming there is just one output neuron), respectively, for the hth pattern. For the exact formulae for the error signals and weight adjustments—and the Delta Rle in general—we refer to e.g. [10].

Related to backpropagation and the weight updates are also the so-called *learning rate* and *momentum* parameters, denoted by η and α, respectively. Moreover, we also need some criteria for when the network is deemed sufficiently learned and training should be stopped. These points are listed and described below.

Learning Rate: The *learning rate* η determines the size of the weight changes during the backpropagation. Small values of the learning rate parameter make the weight corrections small, thus making the weights change only by a little bit, which results in increased learning times (and consequently longer computation times). Although small learning rates do tend to decrease the chance of overshooting the optimal solution to the problem, they also increase the risk of getting stuck at local minima of the error function. Large values of the learning rate, on the other hand, may train the network faster, but with the possibility that no learning is achieved at all.

Momentum: Several factors contribute when it comes to updating the weights. Whereas the learning rate affects the current correction term applied to the weight, as described above, the *momentum* α is a parameter that determines the extent to which the previous correction term should affect the current weight change. Including momentum can help reduce oscillations in the error curve.

[2] The amount of adjustment depends on the output error signal, the activation function and some parameters, the learning rate and the momentum, which will be described shortly. For the precise formulae, see e.g. [10].

Stopping Criteria: There are different ways to determine when the ANN algorithm should stop iterating. In neural network jargon, one iteration corresponds to one *epoch*. Setting a maximum value for the number of epochs can serve as a stopping criterion for the algorithm—whether we have achieved convergence to a solution or not. Another stopping criterion deals with the error, i.e. the difference between the known, target output and the predicted output produced by the network. Specifying when an appropriate measure of the error—e.g. the sum of the squared errors (SSE)—is sufficiently low for one's wishes also serves as a stopping criterion. This *error tolerance*, or *accuracy*, is then the value at which the neural network is deemed to be fully—or, rather, sufficiently—trained.

Overview and Discussion of Training Algorithms

This section lists and discusses the majority of available algorithms for network training. Many of them draw from and include the same basic ideas as the backpropagation algorithm described above. It is beyond the scope of this project to delve into further detail on the theory behind neural network training algorithms. Rather, we will discuss the merits and shortcomings of each algorithm, and argue on this basis as to why they will either be considered or discarded in our particular case.

Gradient Descent Backpropagation: This is the basic backpropagation algorithm described in Section 5.2.2. The network weights and biases are updated in the direction of the negative gradient of the error function, with the steps determined by the constant learning rate parameter. As one of the very first and most common learning algorithms, it sports several extensions. Among these we find the inclusion of the momentum, which defines the sensitivity of the correction term to the previous values. This was also discussed in Section 5.2.2 above. A more advanced extension includes both momentum and an adaptive learning rate; no longer held constant, the learning rate is now adapted during training to continuously find the best balance between large values (which can result in oscillations and instability) and small values (which can make the algorithm take too long to converge) and thus to obtain a lower error and better performance. This most advanced extension is a potential candidate for use in our study, but only comes in as a second choice due to its often slow convergence.

Resilient Backpropagation: This algorithm takes into account only the *sign* of the partial derivative (rather than the magnitude, as we saw for the normal backpropagation algorithm in Section 5.2.2), and adjusts each weight independently. This is also one of the reasons for the algorithm being one of the fastest for pattern recognition problems. However, it does not perform well on function approximation (regression) problems—our main focus—which is why we discard it in favor of some of the other algorithms. For a complete description of resilient backpropagation, see e.g. [20].

Conjugate Gradient: This class spans several algorithms, all of which start out by searching in the steepest descent direction (i.e. normal backpropagation). A search is then performed to determine the optimal distance to move along the current search direction. Then, the next search direction is determined so that it is conjugate to previous search directions. The various conjugate gradient algorithms are distinguished by the manner in which they combine the new steepest descent direction with the previous search direction to obtain the new search direction. Although it varies depending on the problem and application, the conjugate gradient algorithms are generally much faster than adaptive learning rate backpropagation, and sometimes faster than resilient backpropagation. Requiring only

slightly more storage space than simpler algorithms, the conjugate gradient algorithms are good for networks many weights. They are almost as fast as Levenberg-Marquardt on function approximation problems (faster for large networks) and almost as fast as resilient backpropagation on pattern recognition problems. The latter is not that relevant for our case, but the former—their regression performance—make them a plausible contender as the method of choice for us. They will not be our primary choice here, but we will keep them as a second choice. For a detailed discussion, see e.g. [21].

Quasi-Newton Backpropagation: This is an alternative to conjugate gradient algorithms, often with faster convergence. Unfortunately, it requires the equivalent of matrix inverse (an often computationally demanding task) to be computed at each iteration. For large networks, the algorithm is not as adequate as resilient backpropagation and the conjugate gradient methods. It can be an efficient training function for smaller networks, though. We will keep it as a second choice. For more a detailed discussion, see e.g. [22, p. 119].

Levenberg-Marquardt: This algorithm performs an approximation to the matrix computation mentioned above in a much faster and less complex way. A type of mix between the two methods above, it introduces a parameter μ that balances the weight adjustment process between Newton's method (small μ) and a gradient descent method with small step size (large μ). Since Newton's method is faster and more accurate near an error minimum, the aim is to shift towards this methods as quickly as possible by decreasing μ after each successful step (i.e. reduction in the error). μ is only increased when a tentative step would increase the error. In this way, error is always reduced at each iteration of the algorithm.

In general, on function approximation (regression) problems, for moderately-sized networks (up to several hundred weights), the Levenberg-Marquardt algorithm will have the fastest convergence. This advantage is especially noticeable if very accurate training is required. In many cases, the algorithm is able to obtain lower mean square errors (MSE) than any of the other algorithms. However, as the number of weights in the network increases, the advantage of the algorithm decreases. In addition, the algorithm performs relatively poorly on pattern recognition problems. This is not so relevant for our case, though, since we will mainly be concerned with regression. The method's merits for this type of problem make it an outstanding candidate for the training algorithm of choice. For a thorough discussion of the Levenberg-Marquardt method as a neural network training algorithm, we refer to e.g. [23].

Since our main focus in this thesis is on forecasting prices and trends in the financial markets—inherently a regression problem—and since the networks used will not contain too many inputs and hidden neurons (and thus weights), the Levenberg-Marquardt algorithm is a good choice of learning algorithm for our particular application. Indeed, this is the algorithm we will mainly use in the later analysis in Section 6.1.5. Some of the other algorithms also have appealing features, and as second choices we will keep in mind backpropagation with momentum and adaptive learning rate as well as the conjugate gradient.

5.2.3 Recurrent Networks

The feedforward networks discussed above are very efficient at learning *static* patterns. However, when it comes to patterns that change in time—as encountered in some[3] time series—such networks usually underperform and are thus inadequate. The main reason is that feedforward

[3]Not all time series exhibit patterns.

networks do not take past input patterns into account when analysing a pattern presented to them. To deal with this issue one could feed a feedforward network with lagged data, but this approach can drastically increase the number of inputs and thus introduce unwarranted model complexity.

A better solution to the problem of time series data is the use of recurrent, or feedback, networks. These are networks with neurons that have feedback connections; that is, the neurons have connections to themselves or send their outputs to a previous layer. For example, a neuron may send its output to itself, allowing it to incorporate this information in the analysis of the next pattern, or the output layer may send its final answer as an input for the next pattern (see e.g. the bottom panel in Fig. 5.2).

This class spans many different subtypes of networks.[4] In this case, we will make use of a specific type of dynamic, recurrent network; namely, the *nonlinear autoregressive network with exogenous (external) inputs* (NARX).

The NARX network is used to predict future values of a time series $\{y_t\}$ using past values of that time series (the autoregressive part) and past values of a second (or more) time series $\{x_t\}$ (the exogenous inputs). The model can thus be written

$$y(t) = f\left(y_{t-1}, \ldots, y_{t-d}, x_{t-1}, \ldots, x_{t-d}\right), \tag{5.4}$$

where f is some unknown function that the network is trying to approximate, and d is the number of delays, or lags, determining how many past values that are used.

This makes NARX networks particularly appealing for our case, since we would like to predict future values or trends in the price of a financial asset using historical/past information of (i) the price itself (which would correspond to y_t) and (ii) other data such as e.g. technical indicators, other stocks or indexes, etc. (which would be the exogenous inputs $x(t)$).

There exists two different types of NARX network architectures; the series-parallel (open loop) configuration and the parallel (closed loop) configuration. Visual examples of these are presented in Fig. 5.2. In both images, $y(t)$ and $x(t)$ denote the main time series and exogenous inputs, respectively. The large box named "Hidden" shows the hidden layer of the network. Here, we see that delays/lags of $1, 2$ of the target y and inputs x are used (i.e. $d = 2$ according the above notation). Moreover, the hidden layer employs a sigmoid activation function (the curve in the rectangular box), and the number of hidden neurons is 10. The boxes with "w" and "b" indicate the weights and biases, respectively. The large box named "Output" shows the output layer, where we see that there is just a single output neuron with a linear activation function. The difference between the two configurations is the loop (long green line) in the bottom image, which indicates the feedback connection between the output and the input. In the open-loop configuration the true outputs $y(t)$ are sent into the network, while in the closed-loop configuration the estimated outputs are fed back into the network.

[4] It is beyond the scope of this thesis to consider all of them. For this, we refer to e.g. [1], [2].

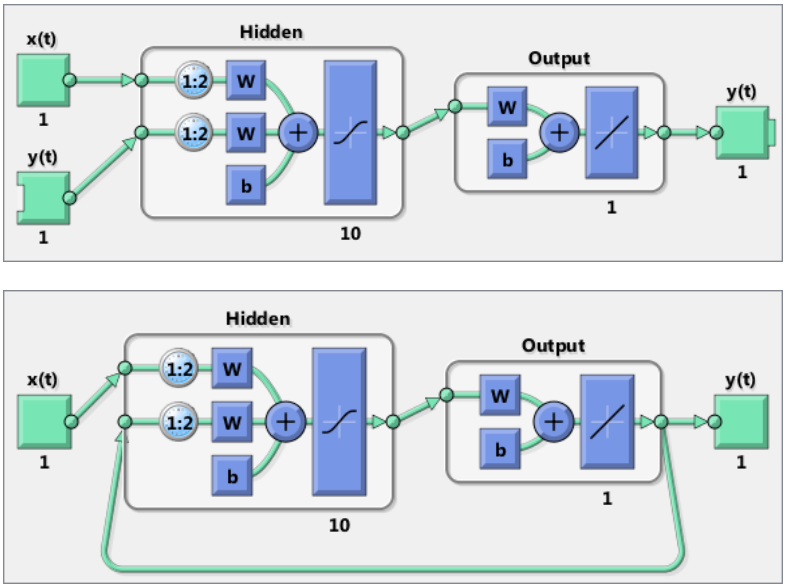

Figure 5.2: NARX network architectures. Top: series-parallel (open loop) configuration. Bottom: parallel (closed loop) configuration.

For network training, the series-parallel configuration is most useful and efficient. The reason is that the true output (e.g. the actual, observed stock prices) is available during the training process, allowing us to use this output instead of feeding back the estimated output through the loop. This has two advantages: (i) the input to the network is more accurate (not using estimated values); and (ii), the resulting network has a purely feedforward architecture, allowing more efficient algorithms to be used for training. The series-parallel (open loop) architecture is also suitable for one-step ahead prediction.

However, for multi-step ahead prediction[5], the series-parallel (closed loop) architecture is the configuration of choice. For example, for predicting the value y_{t+1} one step ahead in time, we can use the known values $y_t, y_{t-1}, \ldots$ and $x_t, x_{t-1}, \ldots$. But, for predicting the value two steps ahead, i.e. y_{t+2}, we do not know y_{t+1} and x_{t+1}. The useful feature of the closed-loop configuration is that it allows us to use the prediction $\tilde{y}_{t+1}$ as an estimate for y_{t+1}. It does this by feeding back, along the closed loop, the estimated output $\tilde{y}_{t+1}$ to be used as an input for prediction at the next time step.

5.2.4 Strengths, Weaknesses and Problems

Here, we recount and discuss the strengths, weaknesses, and problems with ANNs.

In regards to network training, one needs to be aware of one of the main problems with ANNs; namely the problem of *overtraining*, or *overfitting*. As the name suggests, this is a problem that can occur if the neural network is trained too much, i.e. for too many epochs, or by having many hidden neurons.

Now, one of the main advantages of neural networks is their ability to learn from data, and thus their potential to generalize, i.e. to produce a proper output for previously unseen data. This generalization feature is vital for prediction tasks such as ours. However, when a network suffers

[5]In this thesis we mainly consider one-step ahead prediction. Using recurrent networks for multi-step ahead prediction is a topic for future work.

from overtraining, it memorizes patterns in the input and thereby loses the important ability to generalize. We are very interested in avoiding this, and one of the ways to prevent overtraining is to perform so-called train-and-test procedures, in which the network is first trained on a large part of the data sample and then tested on the remaining, smaller part to see if it has the ability to generalize what it learned during training when met with unknown patterns.[6] If the network performs poorly on the test data, then the network configuration (i.e. the number of hidden layers and/or neurons) and parameters (the learning rate and momentum) can be changed and the network re-trained and re-tested.

We also find some drawbacks and shortcomings among the various training algorithms. For instance, backpropagation, the most common learning algorithm, adjusts the weights via the gradient descent method, following a negative slope along the "error landscape." While it does minimize error, there is a chance that it only finds a local minimum, and not a global one, thus yielding a network that is not properly optimized.

We now move on to the network architecture, or structure, which refers to the amount of input, hidden, and output layers, and the number of neurons in each. In this regard, we do not necessarily meet some direct strengths or weaknesses. However, we do face some ambiguity and freedom of choice, as there is no generally optimal way of constructing a neural network.

Now, as for the numbers of input and output neurons, these are determined by the problem at hand, and more specifically the data (i.e. the amount of input features and outputs). As for the hidden neurons, there is no agreed-upon number for the amount of these. There are, however, several rules of thumb suggested in the literature by ANN researchers.[9][10] One is the geometric pyramid rule suggesting that, for a three-layer network with n_i input neurons and n_o output neurons, the hidden layer should have $n_h = \sqrt{n_i n_o}$ neurons (or, depending on the problem, somewhere between one-half to two times this value). Another rule suggests that n_h should be about 75% of n_i, while yet another one says that $n_h = 2n_i + 1$. All of these rules incorporating the number of input neurons implicitly assume that the training set is sufficiently large [10], otherwise the network is prone to suffer from overtraining, as discussed above. This is not much of an issue in our case, though, seeing that our data sets are very large.

In reality, selecting the best number of hidden neurons involves a great deal of experimentation. It is common to apply a so-called fixed method in which the number of hidden neurons is varied—while keeping all other parameters constant—for the purpose of finding the optimal network structure–in the sense that the SSE is minimized, local minima are avoided, and overtraining is prevented. Other, more advanced methods of finding the optimal network structure include optimization techniques such as e.g. Genetic Algorithms, Particle Swarm Optimization, etc. [7], [8], [11]

In a later section we compare ANNs to another popular machine learning method: Support Vector Machines.

5.3 Support Vector Machines

Support Vector Machines (SVMs) are a type of supervised learning method; that is, they use a desired, target output for learning patterns in data. On the fundamental level, SVMs work by establishing an optimal hyperplane that separates the input data. Depending on the type of output data—whether it is discretely labeled (often binary) or continuous—we distinguish between SVM classification and regression, respectively. In the following subsections we will go

[6] From our review of the literature in Section 1.3, the training data often makes up between 80% and 90% of the total data, while the test data makes up the remaining 10-20%.

into much further detail on all of these points and more.[7]

5.3.1 SVM Classification

SVM classification is a generalization of linear decision boundaries for classification. Simpler methods for statistical learning can only obtain optimal separating hyperplanes (the decision boundary) when the classes are linearly separable; in two dimensions, this can be visualized by two clusters of points completely separated by a straight line, i.e. without any points on the wrong side (as in the left panel in Fig. 5.3). SVMs, however, is an advanced method that covers even the nonseparable case where the classes overlap. It produces nonlinear boundaries (visualized in two dimensions by curves and other nonlinear functions) by constructing a linear boundary in a large, transformed version of the feature space.

In going through the theoretical framework of SVMs, we begin by briefly reviewing the simpler problem of constructing an optimal separating hyperplane between two perfectly separable classes. Following [**?**], we then proceed by generalizing to the nonseparable case.

As with all machine learning methods, we must have some training data to feed to the computer in order for it to learn patterns and (hopefully) be able to properly generalize to the unseen test data.

Let the training data consist of N observations $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$. Here, $x_i \in \mathbb{R}^p$, which means that each input pattern x_i is a p-dimensional vector of p features. As for the output, $y_i \in \{-1, 1\}$, meaning that the two classes are labeled by -1 and 1. Then, define a hyperplane by

$$\left\{x : f(x) = x^T\beta + \beta_0 = 0\right\}, \tag{5.5}$$

where β is a unit vector with $\|\beta\| = 1$. Classifying the points into the group $G \in \{-1, 1\}$ is done by

$$G(x) = \text{sign}\left[x^T\beta + \beta_0\right]. \tag{5.6}$$

It can be shown[8] that $f(x)$ gives the signed distance from a point x to the hyperplane $f(x) = x^T\beta + \beta_0 = 0$. Since the classes are separable, we can find a function $f(x) = x^T\beta + \beta_0$ for which $y_i f(x_i) > 0 \forall i$. Consequently, we can find the hyperplane that creates the largest *margin* between the training points for the two classes 1 and -1. This margin, along with the hyperplane and some examples of training points, are visualized in the two-dimensional case in Figure 5.3.

[7] For the basic implementation of SVMs we have made use of the LIBSVM package [33].
[8] See e.g. [2].

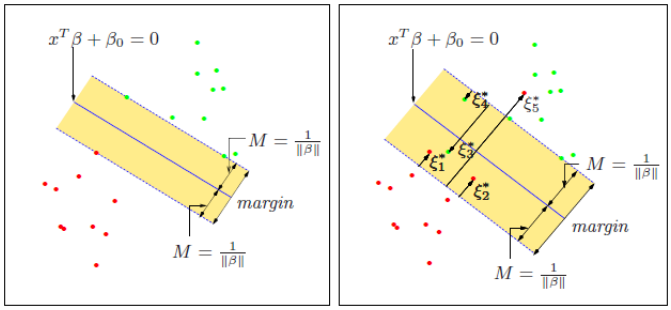

Figure 5.3: Support vector classifiers. The left part shows the separable case. The green and red dots are points, or observations, the colors of which dictate their class. The solid line is the decision boundary, and the dotted lines bound the yellow maximal margin area. The points on the dotted lines are the *support vectors*. The right part shows the nonseparable case where overlap is allowed. Points labeled ξ_j^* are on the wrong side of their margin by an amount $\xi_j^* = M\xi_j$, where M is the width of the margin. Defining $\xi_j = 0$ for points on the correct side, the margin is maximized subject to the constraint $\sum \xi_i \leq$ constant. Hence, $\sum \xi_i \leq C$ is the total distance of points on the wrong side of the margin.

This concept—finding the *largest* margin—is evidently connected to optimization, and more specifically maximization. Indeed, it can be stated as the optimization problem

$$\max_{\beta,\beta_0,\|\beta\|=1} M \tag{5.7}$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq M,\ 1,\ldots,N. \tag{5.8}$$

The set of conditions in the latter equation above ensure that all the points are at least a signed distance M away from the decision boundary (as shown in Figure 5.3) defined by the parameters β and β_0. (Note that β is a p-dimensional vector like x.) In the maximization problem, we thus seek the largest such M and associated parameters. The constraint $\|\beta\| = 1$ can be dropped by replacing the conditions with

$$\frac{1}{\|\beta\|} y_i(x_i^T\beta + \beta_0 \geq M \tag{5.9}$$

$$\Leftrightarrow \quad y_i(x_i^T\beta + \beta_0 \geq \|\beta\| M \tag{5.10}$$

For any β and β_0 satisfying these inequalities, any positively scaled multiple satisfies them, too. Hence, we can set $\|\beta\| = 1/M$. The optimization problem can thus be rephrased as

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 \tag{5.11}$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1,\ 1,\ldots,N. \tag{5.12}$$

Note that the previous maximization problem is now a minimization problem.[9] This is an example of a *quadratic programming* problem where we must minimize a quadratic function subject to a set of linear inequality constraints.

[9] Maximizing $\|\beta\|^{-1}$ is equivalent to minimizing $\|\beta\|^2$. [1, p. 328] The factor of 1/2 is included for later convenience.

Solving this constrained optimization problem can be done using Lagrange multipliers. However, due to the size and scope limitations of this thesis, we will not go into further detail in this regard.[10]

Solving the optimization problem yields the solution vector β and bias $beta_0$, i.e. the parameters that make up the model. It can be shown that these parameters solely depend on those data points x_i that lie *on* the boundary of the maximum margin hyperplane (data points *not* on the hyperplane contribute nothing). These points on the hyperplane are called *support vectors*.

The optimal separating (maximum margin) hyperplane produces a function $\hat{f}(x) = x^T\hat{\beta} + \hat{\beta}_0$, where $\hat{\beta}$ and $\hat{\beta}_0$ are the determined optimal parameters. This function—namely its sign—is used to classify new points, or observations, via

$$\hat{G}(x) = \text{sign}\hat{f}(x) \tag{5.13}$$

Overlapping Classes

We now proceed by considering the case where the classes overlap in feature space, as shown in the right part of Figure 5.3. One way to deal with this problem is to still maximize M, but allow for some points to be on the wrong side of the margin. Thus, we introduce the appropriately named *slack* variables $\xi = (\xi_1, \xi_2, \ldots, \xi_N)$. The constraints in (5.8) of the previous maximization problem can then be modified to

$$y_i(x_i^T\beta + \beta_0) \geq M(1 - \xi_i), \tag{5.14}$$

for all i, $\xi_i \geq 0$, and $\sum_{i=1}^{N} \xi_i \leq$ constant. Recalling that M is the width of the margin, the above equation indicates that the overlap is measured in relative distance (which changes with M).

The value ξ_i in the constraint above is thus the proportional amount by which the point falls on the wrong side of its margin. Points for which $0 < \xi_i \leq 1$ lie inside the margin, but on the correct side of the decision boundary. Points for which $\xi_i > 1$ lie on the wrong side of the decision boundary and are thus misclassified. By bounding the sum $\sum_{i=1}^{N} \xi_i$, we bound the total proportional amount by which points fall on the wrong side. Since misclassification occurred when $\xi_i > 1$, bounding the sum at the constant, say, K, i.e. $\sum_{i=1}^{N} \xi_i \leq K$, bounds the total number of misclassified training points at K.

As before, we want to maximize the margin. But we also want to softly penalize points that lie on the wrong side of the margin boundary. We therefore minimize

$$C\sum_{i=1}^{N} \xi_i + \frac{1}{2}\|\beta\|^2 \tag{5.15}$$

under the constraints $\xi_i \leq 0$ and $y_i(x_i^T\beta + \beta_0) \geq M(1 - \xi_i)$. Thus, the problem is again quadratic with linear inequality constraints—i.e. a convex optimization problem. Written more formally we have

[10] For a complete treatment, we refer to e.g. [1], [2].

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i \tag{5.16}$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \tag{5.17}$$

$$\text{and } \xi_i \leq 0 \tag{5.18}$$

Here, the constant $C > 0$ is the *cost* parameter controlling the trade-off between the slack variable penalty and the margin. The separable case considered earlier corresponds to the limit $C \to \infty$.

As before, the problem can be solved using Lagrange multipliers.[11]

Solving the optimization problems yields the parameters β and β_0 which define the optimal separating hyperplane. As before, these parameters depend solely on the *support vectors*. Before, the support vectors were those points that lay *on* the hyperplane. In this case of overlapping classes, the support vectors are those points that lie on the edge of the margin or inside it. The slack variables ξ_i determine whether the points are correctly classified; if $0 < \xi_i \leq 1$, then the point x_i is on the correct side of the boundary and thus correctly classified; if $\xi_i > 1$), then it is misclassified.

Letting our parameter estimates be denoted $\hat{\beta}$ and $\hat{\beta}_0$, the decision function for classifying new observations is given by

$$\hat{G}(x) = \text{sign}\left[\hat{f}(x)\right] \tag{5.19}$$

$$= \text{sign}\left[x^t\hat{\beta} + \hat{\beta}_0\right] \tag{5.20}$$

ν-SVC

There exists an alternative, equivalent formulation of the support vector machine, known as the ν-SVM. This is very similar to the above case (called C-SVC). The main difference is that the cost parameter C has been replaced by ν.[12] This parameter can be interpreted as both an upper bound on the fraction of *margin errors* (i.e., points for which $\xi_i > 0$ and which thus lie on the wrong side of the margin boundary) and a lower bound on the fraction of support vectors.

As mentioned earlier, only the support vectors contribute to the parameters defining the optimal separating hyperplane—that is, the decision boundary—and thus only the support vectors are used for making predictions for new inputs.

The theory covered so far deals with support vector classifiers that find *linear*[13] boundaries in the input feature space. This linear method can be made more flexible by enlarging the feature space via basis expansions such as polynomials or splines; that is, we transform the input x via some function. With this basis function extension we will then be finding linear boundaries in the enlarged space—which generally achieves better separation of the training data [2]. These linear boundaries in the enlarged space translate to *nonlinear* boundaries in the original space. All that needs to be done is selecting the basis functions $h_m(x), m = 1, \ldots, M$, after which the procedure is the same as before. We then use the transformed input features

[11] Again, to keep this thesis from getting too long, we will not go into further details in this regard. For a full treatment, we refer to e.g. [**?**], [1].

[12] Here, in the case of classification, the parameter ν in ν-SVC replaces the parameter C in C-SVC. For regression, which we consider shortly, the parameter ν in ν-SVR replaces the parameter ϵ in ϵ-SVR.

[13] As seen from the linearity in $x^T\beta$ in (5.5).

$h(x_i) = (h_1(x_i), h_2(x_i), \ldots, h_M(x_i)), i = 1, \ldots, N$, and produce the nonlinear function $\hat{f}(x) = h(x)^T\hat{\beta} + \hat{\beta}_0$. Points are then classified according to the sign of this function, i.e. by $\hat{G}(x) = \text{sign}(\hat{f}(x))$ as before.

It can be shown that the transformation $h(x)$ only appears through inner products. In fact, $h(x)$ does not need to be specified at all; a sufficient requirement is knowledge of the *kernel* function

$$K(x, x') = \langle h(x), h(x') \rangle , \tag{5.21}$$

which computes inner products in the transformed feature space.[14] The solution $\hat{f}(x)$ can now be written

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{\alpha}_i y_i K(x, x_i) + \hat{\beta}_0 \tag{5.22}$$

In the SVM literature there are three popular choices for the kernel function K: the d-th degree polynomial, the radial basis, and the neural network, respectively:

$$K(x, x') = (1 + \langle x, x' \rangle)^d \tag{5.23}$$

$$K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right) \tag{5.24}$$

$$K(x, x') = \tanh\left(\kappa_1 \langle x, x' \rangle + \kappa_2\right) \tag{5.25}$$

As a final remark, let us briefly return to the cost parameter C. A large value of C will discourage any positive ξ_i, i.e. points that lie far away from their margin, and thus lead to a decision boundary (in the original space) that wiggles around the points in order to correctly classify them. Such a wiggly decision boundary can be related to an overfit of the problem, causing bad generalization performance on unseen (test) data. A small value of C will encourage a small value of $\|\beta\|$, which in turn causes $f(x)$ and hence the decision boundary to be smoother. As always, it is a balance between accuracy on the training data and generalization ability on the test data. The optimal parameter value is often found via *cross-validation*, an important and very usable concept often encountered in machine learning problems.

In the above we have considered SVM classification for the two-class problem. Indeed, SVMs are fundamentally a two-class classifier, but it is possible to extend the framework to also tackle multi-class problems with more than two classes. However, since the financial classification problems posed and solved in this study are exactly of the two-class nature, we will not go into further detail on the multi-class case.[15] Instead, we will proceed with SVMs for regression tasks.

5.3.2 SVM Regression

We now extend the SVM framework to regression tasks with a continuous response. First, we consider the linear regression model

$$f(x) = x^T\beta + \beta_0, \tag{5.26}$$

after which we will generalize to the nonlinear case. To estimate β, we minimize

[14] K should be a symmetric positive (semi-)definite function [2].

[15] For a treatment of multi-class SVMs, see e.g. [1].

$$H(\beta, \beta_0) = \sum_{i=1}^{N} V_\epsilon \left(y_i - f(x_i)\right) + \frac{\lambda}{2}\|\beta\|^2, \tag{5.27}$$

where

$$V_\epsilon \left(y_i - f(x_i)\right) = \begin{cases} 0, & \text{if } |y_i - f(x_i)| < \epsilon, \\ |y_i - f(x_i)| - \epsilon, & \text{otherwise.} \end{cases} \tag{5.28}$$

V_ϵ is an *ϵ-insensitive error function.* As the expression shows, it gives zero error if the absolute deviation between the prediction $f(x)$ and the target y is less than ϵ, where $\epsilon > 0$. Thus, it basically ignores errors less than ϵ. (This is roughly analogous to the part in the SVM classification above, where points on the correct side of the decision boundary—but far away from it—were irrelevant and ignored in the optimization.)

Letting $\hat{\beta}$ and $\hat{\beta}_0$ be the parameters that minimize H, the solution function can be shown[16] to have the form

$$\hat{\beta} = \sum_{i=1}^{N} \left(\hat{\alpha}_i^* - \hat{\alpha}_i\right) x_i \tag{5.29}$$

$$\hat{f}(x) = \sum_{i=1}^{N} \left(\hat{\alpha}_i^* - \hat{\alpha}_i\right) \langle x, x_i \rangle + \beta_0, \tag{5.30}$$

where $\hat{\alpha}_i, \hat{\alpha}_i^* > 0$ solve the quadratic programming problem

$$\min_{\alpha_i, \alpha_i^*} \epsilon \sum_{i=1}^{N} \left(\alpha_i^* + \alpha_i\right) - y_i \sum_{i=1}^{N} \left(\alpha_i^* - \alpha_i\right) + \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \left(\alpha_i^* - \alpha_i\right)\left(\alpha_k^* - \alpha_k\right) \langle x_i, x_k \rangle \tag{5.31}$$

subject to the constraints

$$0 \leq \alpha_i,\ \alpha_i^* \leq 1/\lambda, \tag{5.32}$$

$$\sum_{i=1}^{N} \left(\alpha_i^* - \alpha_i\right) = 0, \tag{5.33}$$

$$\alpha_i \alpha_i^* = 0. \tag{5.34}$$

The nature of these constraints typically leads to only a subset of the solution values $(\hat{\alpha}_i^* - \hat{\alpha}_i)$ being nonzero. The data points x_i corresponding to these nonzero solution values are exactly the support vectors. Now, as in the above case of classification, the solution depends on the input values x_i only through the inner products $\langle x_i, x_k \rangle$. Hence, we can generalize this framework to larger spaces by defining an appropriate inner product; that is, as we did in the end of the previous section, by specifying the kernel function K that computes inner products in the transformed feature space.

In this way, the framework is extended to the nonlinear case, since linear decision boundaries in the transformed, larger feature space translates to nonlinear decision boundaries in the original

[16] See e.g. [2].

space. Replacing the inner product in (5.30) with the kernel function K for inner products in the larger space, the new decision function becomes

$$\hat{f}(x) = \sum_{i=1}^{N} (\hat{\alpha}_i^* - \hat{\alpha}_i) K(x, x_i) + \beta_0 \tag{5.35}$$

from which predictions for new inputs can be made.

ν-SVR

Earlier, for classification, we mentioned that an alternative, equivalent formulation existed; namely, the ν-SVM. The same is true for this case of regression.[17] The benefit, or advantage, in the ν-SVM formulation is that the parameter governing the model complexity has a more intuitive interpretation. Instead of fixing the width ϵ (the parameter worked with in the above) of the insensitive region, we fix instead a parameter ν that bounds the fraction of points lying outside the region. This gives us much better control of the model. It can be shown that there are at most νN data points falling outside the insensitive region, while at least νN points are support vectors lying either on the region or outside it [1].

We end our treatment here of the theoretical, methodological framework behind SVMs.[18] In the next section we highlight the strengths, weaknesses, and problems.

5.3.3 Strengths, Weaknesses, and Problems with SVMs

In our earlier treatment of SVM classification we highlighted that SVM classifiers are fundamentally geared towards two-class problems. Although extensions have been made to multi-class problems [1], these still present some problems and are in need of an optimal design [30]. However, since the financial classification problems posed and solved in this study are exactly of the two-class nature, we need not worry about these issues.

Another limitation, perhaps the biggest one, lies in the choice of the kernel function K used to define inner products in the larger, transformed space. Once the kernel is fixed, there really remains just one user-specified parameter; the error penalty C. However, "the kernel is a very big rug under which to sweep parameters ... [and] the best choice of kernel for a give problem is still a research issue" [30].

[30] also lists some additional limitations of SVMs. Among these are the problems with discrete data as well as the limitations in regards to speed and size, both in the training and testing phase. Training for very large datasets (with millions of support vectors) is still an unsolved problem. However, this will not be a problem for our case, since the size of the financial datasets considered here will only reach the hundreds at the very maximum. The reason is that stock data sufficiently far back in the past is deemed too old to have any significantly predictive power and influence on the present and near future.

Despite these limitations—many of which will, in fact, not influence us significantly in this study—SVMs are a class of very good and popular methods, with a number of advantages over some other machine learning methods. The key features are the use of kernels, the absence of local minima, the sparseness of the solution, and the capacity control obtained by optimising the margin.

In regards to kernels, we highlighted a limitation above; however, the use of kernels, as mentioned in the previous theoretical sections, is also a good thing in that it allows for extensions

[17] Here, for regression, the parameter ν in ν-SVR replaces the parameter ϵ in ϵ-SVR.
[18] For a more rigorous treatment with additional extensions, we refer to e.g. [1], [2].

to larger, higher-dimensional spaces and thus for nonlinear decision boundaries in the original space.

The absence of local minima is a very interesting and extremely attractive feature that sets SVMs apart from e.g. Artificial Neural Networks (ANNs). This comparison is actually very interesting. Whereas the development of ANNs followed a heuristic path, with applications and extensive experimentation preceding theory, the development of SVMs began with sound theory followed by implementation and experiments. A significant advantage of SVMs is that, whilst ANNs can get stuck in local minima, the solution to an SVM is global and unique. This is related to the fact that SVMs use so-called *structural risk minimization*, while ANNs use *empirical risk minimization*.[19] SVMs also have a simple geometric interpretation and give a sparse solution, as opposed to ANNs. Furthermore, unlike ANNs, the computational complexity of SVMs does not depend on the dimensionality of the input space; that is, while ANN models become very complex and thus take very long to train when met by sufficiently many input features, SVM models do not suffer from this limitation. Finally, the reason that SVMs often outperform ANNs in practice is that they deal with the biggest problem with ANNs; SVMs are less prone to overfitting and thus often show better generalization ability and performance when met with unseen test data. [34]

5.4 Random Forests

Random Forests (RFs) [35] are a type of *ensemble learning* method using a divide-and-conquer approach to improve performance. The main principle behind ensemble methods as a whole is that a group of "weak learners" (individual classifiers) can come together to form a "strong learner." Generally, the weak learners are individual classifiers, while the strong learner is obtained when considering them collectively.

More specifically for Random Forests, the weak learners are so-called *decision trees*. The Random Forest then averages the conclusions made by a number of these. For classification problems, the ensemble of simple trees (called a *committee*) vote for the most popular class. In regression problems, the Random Forest's estimate of the dependent variable is obtained by averaging the continuous response from each tree. Evidently, in order to understand Random Forests, we must first understand the trees comprising them.

Tree-based models are simple but powerful, and moreover interpretable by humans. Generally, they partition the feature space into rectangular regions and a assign a simple model—usually a constant value or label—to each region. Several tree-based methods exist, with odd names such as C4.5, ID3, etc., but we will use as a starting point the so-called Classification and Regression Trees (CARTs), which are binary trees. Afterwards, we will then proceed with our main focus, Random Forests, which solve a number of shortcomings and limitations of normal tree-based models.

5.4.1 Classification and Regression Trees

To motivate the understanding of Classification and Regression Trees (CARTs), we consider an example of a regression problem with a continuous response Y and inputs X_1 and X_2, each taking values in the unit interval. The feature space is partitioned by lines parallel to the coordinate axes. To avoid problems, we are restricted to recursive binary partitions, which means that

[19] It is beyond the scope of this study to go into further detail on these concepts. For a treatment of structural and empirical risk minimization, we refer to e.g. [31], [34].

the feature space is not only partitioned by such parallel lines, but indeed partitioned into non-overlapping rectangles. This is shown in the left part of Figure 5.4.

The approach is as follows. We first split the space into two regions, and model the response by the mean of Y in each region. The variable and split-point are chosen to achieve the best fit. Then one or both of these regions are split into two more regions, either of which may in turn be split themselves, and so on. This process is continued until some stopping rule is applied. In the left part of Figure 5.4 we first split the space by a vertical line at $X_1 = t_1$. The left region $X_1 \leq t_1$ is then split by a horizontal line at $X_2 = t_2$; and the right region $X_1 > t_1$ is ended by a vertical line at $X_1 = t_3$. Finally, the rightmost region $X_1 > t_3$ is split horizontally at $X_2 = t_4$. This partitioning creates the five regions $R_1, R_2, \ldots, R_5$, as shown in the figure. The corresponding regression model predicts the response Y with a constant c_m in region m, i.e,

$$\hat{f}(X) = \sum_{m=1}^{5} c_m I\left\{(X_1, X_2) \in R_m\right\}. \tag{5.36}$$

This model can also be represented by the tree structure in the right part of Figure 5.4. It is a binary tree since each node branches into two new ones. The full dataset sits at the top of the tree. Observations that satisfy the condition at each node, or junction, are assigned to the left branch, while the others are assigned to the right branch. The terminal nodes (also called *leaves*) in the bottom of the tree correspond to the regions $R_1, R_2, \ldots, R_5$.

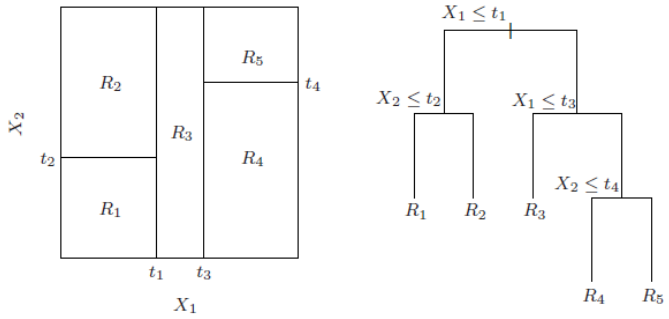

Figure 5.4: Example of CART partitioning. The left part shows a partitioning of a two-dimensional input feature space by recursive binary splitting. The right part shows the corresponding tree structure.

As mentioned earlier, one of the key advantages of the recursive binary tree is its interpretability; the feature space partition is fully described by a single tree. Of course, higher-dimensional problems with more than two input features make it difficult to draw partitions such as that in the left part of Figure **??**. However, the binary tree representation in the right part of the figure works in the same way. This representation is also very intuitive, mimicking a simple "if-then" decision process.

Regression Trees

We now turn to the more general case of regression trees and how to "grow" them. Let our data consist of N observations, each of which comprises p inputs (the features) and an output (the response). Thus, the data can be described by $\{(x_i, y_i)\}_{i=1}^{N}$, with $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. The algorithm must automatically decide on the splitting variables and split points (the X_i's and t_i's

in the above example), and also what topology, or shape, the tree should have. Suppose first that we deal with a partitioning into M regions $R_1, R_2, \ldots, R_M$, and the response is modelled as a constant c_m in each region. This gives the model

$$f(x) = \sum_{m=1}^{M} c_m I\left(x \in R_m\right). \tag{5.37}$$

Now, for a given observation, the difference between the true response and the result from our model is clearly $y_i - f(x_i)$. If we, as a criterion for the automatic decision of the tree topology, adopt minimization of the sum of squared errors (SSE) $\sum_i \left(y_i - f(x_i)\right)^2$, then the best constant $\hat{c}_m$ is just the average of y_i in region R_m,

$$\hat{c}_m = \text{avg}\left(y_i | x_i \in R_m\right), \tag{5.38}$$

which, in words, naturally reads the average of y_i over all input observations x_i falling in the region R_m.

Now, finding the best binary partition in terms of minimum SSE is generally computationally infeasible. Following [2], we therefore proceed with a "greedy" algorithm. Starting with all of the data, we consider a splitting variable j and split point s (in our previous example, these could correspond to e.g. X_1 and t_1), and define the pair of half-planes

$$R_1(j,s) = \left\{X | X_j \leq s\right\}, \tag{5.39}$$
$$R_2(j,s) = \left\{X | X_j > s\right\}. \tag{5.40}$$

Evidently, if we look at the simple two-dimensional case as in the previous example, R_1 would be the left region in feature space and R_2 the right region (granted, of course, that the splitting variable was the one along the horizontal axis, i.e. X_1). In any case, we now have two-half planes, the separation of which is determined by j and s. Since we want a partitioning that minimizes the sum of squared errors, we seek the splitting variable j and split point s that solve

$$\min_{j,s} \left\{ \min_{c_1} \sum_{x_i \in R_1(j,s)} \left(y_i - c_1\right)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} \left(y_i - c_2\right)^2 \right\} \tag{5.41}$$

The inner part is a minimization of the SSE in the two regions R_1 and R_2, respectively; that is, for all observations x_i in a particular region, we choose the constant c that minimizes the sum of squared differences. The outer part is just a minimization over j and s, which then gives us the optimal (in the sense of lowest total SSE) splitting variable and point. For any choice of j and s, the inner minimization is solved by

$$\hat{c}_1 = \text{avg}\left(y_i | x_i \in R_1(j,s)\right), \tag{5.42}$$
$$\hat{c}_2 = \text{avg}\left(y_i | x_i \in R_2(j,s)\right). \tag{5.43}$$

For each splitting variable j, the determination of the split point s can be done very quickly, and thus by scanning through all of the inputs, determination of the best pair (j,s) is now feasible.

We have now found the best initial split. The next step is then to partition the data into the resulting regions R_1 and R_2, and then repeat the splitting process on each of these two regions. This process is repeated on all of the resulting regions.

However, since we cannot go on indefinitely, a question arises: For how long should we repeat? Each split is like two new branches, so how large should we grow the tree? Clearly, in some limit of sufficiently many splits, the regions will shrink, each holding none to very few points. Hence, the model will fit the training data perfectly, but it comes at the cost of being able to generalize on unseen test data. A large tree might therefore lead to overfitting. A small tree, on the other hand, might not capture the important structure. Again, as with so many other methods, we face a trade-off between a model that is too complex and prone to overfitting and one that is too simple to uncover any important patterns in the data.

Tree size is a tuning parameter governing the complexity of the model, and the optimal tree size should be adaptively chosen from the data. One approach would be to introduce some threshold, errors below which are deemed insignificant; splits would then only be performed if the decrease in SSE due to that split exceeds the threshold. Such a strategy is too short-sighted, however, since a seemingly worthless split (small decrease in SSE) might lead to a good, valuable split (large SSE decrease) below it. A better strategy is to grow a large tree and then perform so-called *cost-complexity pruning*.[20]

Classification Trees

Above, we considered the regression problem where the response is continuous. We now proceed with the case of classification where the target could be categorical and generally take on values $1, 2, \ldots, K$. In fact, the only changes needed in the tree algorithm are related to the criteria for splitting nodes and pruning the tree. Above, we used as a node impurity measure the squared error $Q_m(T)$ defined by (**??**). However, since the output y_i is no longer continuous, this measure is not adequate for classification. Instead, we do as follows. For the (terminal) node m representing the region R_m with N_m observations in it, let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I\left(y_i = k\right). \tag{5.44}$$

Evidently, $\hat{p}_{mk}$ denotes the proportion of class k observations in node m. In this node there will probably be observations from a few different classes. But our model must for each node classify observations into just one class. Thus, observations in node m are classified to the most common class in that node; that is, the class in node m with the largest proportion of observation. In formulaic terms, observations in node m are classified to class $k(m) = \arg\max_k \hat{p}_{mk}$. For a measure $Q_m(T)$ of node impurity, it is then common to choose one of the following:

Misclassification Error:	$\frac{1}{N_m} \sum_{x_i \in R_m} I\left(y_i \neq k(m)\right) = 1 - \hat{p}_{mk(m)}$
Gini Index:	$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}\left(1 - \hat{p}_{mk}\right)$
Cross-entropy or Deviance:	$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$

In the case of two classes, and if p is the proportion of observations in the second class, these three measures are $1 - \max\left(p, 1-p\right)$, $2p(1-p)$, and $-p \log p - (1-p)\log(1-p)$, respectively. All three are quite similar in shape. However, whereas the misclassification error evidently has a triangular, pyramid-like shape, the latter two are smooth and differentiable and thus more

[20] It is beyond the scope of this thesis to go into further detail. For a more rigorous treatment we refer to e.g. [2], [35].

suitable for numerical optimization. Looking back at (**??**) and (**??**), we see that the node impurity measure $Q_m(T)$ by the number N_{m_L} and N_{m_R} of observations in the two child nodes created by splitting node m.

Finally, for growing the tree, it is recommended to use either the Gini index or cross-entropy. Guiding the cost-complexity pruning, on the other hand, can be done by any of the three measures, though the misclassification rate is typically chosen [2].

Strengths and Issues of Decision Trees

Decision trees in general, including CARTs as we have considered here, have a number of advantages and disadvantages.

The strengths include their intuitive structure and interpretability. Another advantage is that they are applicable to both numerical and categorical data (i.e. for regression and classification tasks, respectively), and that the frameworks are very similar. Finally, the methods are applicable to large datasets.

Regarding disadvantages, one of the main problems with tree-based models is that they suffer from instability and high variance. Often a small change in the data can result in very different series of splits, making interpretation somewhat uncertain. The main reason for this instability is the hierarchical nature of the tree-growing process; namely, the effect of an error in the top split is propagated down to all of the consequent splits below it.[21] Moreover, they have non-smooth prediction surfaces/boundaries (as opposed to e.g. SVMs that we considered earlier), which is especially a problem in regression tasks and can be seen by the constants c_m that are clearly not smooth (unless they are equal) across different regions R_m. Finally, when compared to other machine learning methods, tree-based models often have little theoretical understanding in terms of statistical learning theory.

For a further discussion of tree-based methods, we refer to e.g. [2], [1]. Now we proceed with the method we will actually employ in the practical part of this study—namely Random Forests, which solve many of the problems associated with decision trees such as CARTs.

5.4.2 Random Forests

Random Forests [35] belongs to the class of so-called *bagging*, or *bootstrap aggregation* techniques that reduce the variance of an estimated prediction function. In fact, it is a substantial modification of bagging that builds a large collection of *de-correlated* trees, and then averages them, thereby alleviating the main shortcomings—instability and high variance and lack of smooth decision boundaries—of single decision trees. For regression, we fit the same regression tree many times on bootstrap-sampled versions of the training data, and then average the result. For classification, a *committee* of trees each cast a vote for the predicted class.

The essential idea in bagging is to average many noisy but approximately unbiased models, thus reducing the variance. If grown sufficiently deep, decision trees have relatively low bias, while always being notoriously noisy. This makes them ideal candidates for bagging. Moreover, since each tree generated in bagging is identically distributed (i.d.) [**?**], the expectation of an average of B of them is the same as the expectation of any one of them. Consequently, the bias of bagged trees is the same as that of the individual trees, and the only hope of improvement is through variance reduction.

Now, to start off, consider a number B of independent and identically distributed (i.i.d.) random variables, each with variance σ^2. The variance of their mean is given by $\frac{1}{B}\sigma^2$. If, on the

[21] The ensemble learning method of Random Forests alleviates this problem as we shall see shortly.

other hand, the variables are only identically distributed (i.d.) and have pairwise correlation ρ, then the variance of the average is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2. \tag{5.45}$$

Evidently, as B increases—corresponding to growing more trees—the second term vanishes. But the first remains, causing the correlation ρ to limit the benefit of averaging. The idea in Random Forests is then to reduce the variance through a reduction in the pairwise correlation. Such a reduction in the correlation is achieved in the tree-growing process through random selection of the input variables. More specifically, the approach when growing a tree is, before each split, to randomly choose $m \leq p$ of the input variables as candidates for splitting. Typical values for m are $\sqrt{p}$ or even as low as 1.

Then, after B such trees have been grown—denoted by the set $\{T(x;\Theta_b)\}_{b=1}^{B}$—the random forest predictor (for regression) is

$$\hat{f}_{RF}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T(x;\Theta_b). \tag{5.46}$$

Here, Θ_b is a variable characterizing the bth random forest in regards to split variables, cut points at each node, and terminal-node values. $T(x;\Theta_b)$ is the prediction of the bth tree at the point x.

Now, recall that m of the input variables were randomly chosen as candidates for splitting. Larger m will therefore make it more probable that the same input variables are chosen for splitting, hence making the trees more alike—and more correlated. Conversely, reducing m will thus reduce the correlation ρ between pairs of trees, and thereby reduce the variance of the average as seen by (5.45).

Above, we have mainly considered regression. Regarding classification, each individual tree makes a prediction in the form of a class. Each such prediction can be considered a vote for that particular class. The ensemble of trees—the Random Forest—then collects the votes and classifies the point into whichever class that received the most votes.

The Random Forest approach for classification and regression can be summarized by the following algorithm [2]:

Random Forest Algorithm for Classification and Regression:

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $Z*$ of size N from the training data.
   (b) Grow a tree T_b to the bootstrapped data by recursively repeating the steps below for each terminal node (leaf) of the tree until the minimum node size n_{min} is reached.
      i. Select m variables at random from the available p input features.
      ii. Pick the best variable/split-point among the m variables.
      iii. Split the node into two child nodes.
2. Output the ensemble of trees $\{T_b\}_{b=1}^{B}$.

A prediction for a new point x is made by:

- Regression: $\hat{f}_{RF}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$

- Classification: $\hat{C}^B_{RF}(x) = \text{majority vote} \left\{ \hat{C}_b(x) \right\}_{b=1}^{B}$, where $\hat{C}_b(x)$ is the class predicted by the bth tree.

In [35], the inventors of Random Forests make some recommendations for parameters:

- For classification, the default value for m is $\sqrt{p}$ and the minimum node size is one.
- For regression, the default value for m is $\frac{p}{3}$ and the minimum node size is five.

In practice, the best parameter values will depend on the problem, and they should be tuned.

As for the number of trees, B, bigger is generally better. It is common to start with e.g. $B = 100$ and then increase it. Of course, as seen by (5.45), B should of course be sufficiently large so as to make the second term in the variance of the average vanish.

Now, as mentioned above, the parameter m should be tuned, and as in many methods, this is often achieved via cross-validation. However, Random Forests make another approach possible, where the model is fit in just one sequence (as opposed to other methods requiring a model fit followed by a separate cross-validation sequence). This is related to the method's use of *out-of-bag* (OOB) samples: For each observation x_i in the training set, a random forest predictor is built by averaging over all trees constructed *not* using x_i. As it happens [2], this OOB error estimate is almost identical to that obtained by n-fold cross-validation. Consequently, this OOB error can be used instead of cross-validation to determine m, making it possible for Random Forests to be fit in just one sequence with the cross-validation-alternative being performed along the way.

Finally, as mentioned before, Random Forests addresses the main problems with CARTs listed in section 5.4.1. This means that Random Forests really have very few shortcomings. Perhaps more like a general lack of knowledge than a shortcoming per se, the method still has relatively little theoretical understanding in terms of statistical learning theory, a feature it has inherited from CARTs. As for advantages, however, there are plenty; these include a good performance without much tuning, applicability to both numerical and categorical data (i.e. both regression and classification tasks, respectively), simple parallelization, applicability to large datasets, and optional probabilistic output.

The above treatment of decision trees and Random Forests is sufficient for our use of the methods in this study, and we shall end the section here. For a more detailed analysis and discussion of Random Forests, we refer to e.g. [2], [1], [35].

5.5 Application to Financial Forecasting

In this section we describe and discuss how the above machine learning methods can be applied to the task of financial forecasting. Of course, the only limitation is imagination, and there will be numerous such applications—some more plausible than others. We focus on the applications that have been pursued in this study.

Neural Networks, Support Vector Machines, and Random Forests can all be used for both classification and regression. In regards to classification, some methods generalize easier to multi-class classification (e.g. RFs) than others (e.g. SVMs). But the two-class problem is solidly supported by all of them.

Fortunately, the type of problems we encounter with the financial data in this study can exactly be posed as two-class classification problem. To elaborate on this, consider a time series of the price of a stock. Let us say that the prices are quoted on a daily basis. Then, since we are considering classification, we label each data point; if the price closes at a higher level on the

current day than the previous day (i.e. the price has increased), the data point for the current day is labelled "up"; similarly, if the price has decreased, the data point is labeled "down." To make it easier to program on a computer, these "up" and "down" labels can, without loss of understanding, be denoted 1 and -1, or the binary 1 and 0.

We now have two groups, or classes. Next, we are interested in some *features* that might have caused the stock price to go up or down. Of course, as mentioned earlier, there is clearly a large degree of human psychology and other complex factors influencing this. So, we can either assume that everything—including the sometimes erratic, hard-to-predict human behaviour—is included in the raw and derived data; or we can put human psychology aside and assume that the stock price movements are to some extent caused by certain features in the data. Regardless of which way we choose, these features could be anything from the stock prices of the previous days, the trading volume, the volatility, technical indicators, as well as fundamental data, etc. In theory, these features can be anything that one can imagine to have the slightest influential and predictive power. Indeed, if one considered the stock of, say, a company that specialized in removing water from flooded houses, then weather data such as the daily precipitation (or, more likely, the precipitation forecast) would be highly relevant as a potential feature to the input data.

In any case, for each daily observation we now have a number of input features as well as the label or class of that point. Our task is then to predict, or classify, tomorrow's stock price movement using an observation of input data. If the previous stock prices were deemed influential for future stock prices, then one of the input features would be today's stock price. Or, in our weather company case, another feature could perhaps be today's forecast for tomorrow's precipitation rate. Finally, the observation's label—i.e. to which class the data point of tomorrow's stock price belongs—is determined by the NN, SVM, or RF method.

Regarding regression, things are much the same as the classification case described above. The main difference is that, instead of trying to predict the movement—i.e. "up" or "down"—of the stock price, we attempt to predict a continuous value. Still, we do this using a number of input features such as those described above.

Chapter 6

Analysis

This chapter contains our analyses, experiments, and results. We begin by analysing the distributional properties and time series structure of the daily returns of the NASDAQ Index. Next, we construct, select, and test ARMA-GARCH models for predicting the future prices and trends in the NASDAQ Index. Then we turn our attention to the machine learning methods, where we explore the workings of the models and analyse the effects of changing parameters. This gives a better feel for the methods and also showcases one of the most important problems in machine learning—overfitting. After this, we proceed with numerous experiments. Using mainly the machine learning methods, we investigate how well these perform in one-day ahead prediction of stock prices. The experiments also examine the effects of varying different properties of the data and predictors, etc.

6.1 Analysis of the NASDAQ Index and Initial Experiments

In this section we analyse the NASDAQ Index[1] for the 4-year time period from Jan 1 2010 to Jan 1 2014. Using daily data, this gives us 1006 observations of prices (1005 for the logarithmic return and up/down direction data).

First, we consider some statistical properties and explore selected aspects of the wide array of raw and derived data. We then initiate the time series analysis by investigating both quantitatively and qualitatively the potential presence of autocorrelation and conditional heteroscedasticity. Using the results from this analysis, we proceed by constructing and diagnosing conditional a mean and variance composite model, and then it them for forecasting. Finally, we evaluate the forecasting performances of numerous different models.

After this, we turn to the machine learning methods of Artificial Neural Networks, Support Vector Machines, and Random Forests. For each method (and sub-methods: regression and classification) we construct numerous models and evaluate their forecasting performances on both training and test sets. Our main focus in these initial experiments is to analyse the applied methods and investigate the effects of changing various parameters. The reasons for this are multiple: to familiarize ourselves with the models and their behaviour; to see if their features and results are consistent with what is expected from theory; to get a sense of the best parameter values; and to investigate the models' strengths and weaknesses, including e.g. the problems of overfitting and underfitting.

[1] The ticker symbol for the NASDAQ Index is "^IXIC"

6.1.1 Descriptive Statistics

Our written MATLAB program computes some statistical properties for any given asset. Table 6.1 shows the results for the daily NASDAQ returns.

Range	Mean	Median	Variance	Skewness	Kurtosis
$[-0.071489; 0.051592]$	0.000590	0.000899	0.000139	-0.409176	6.538752

Table 6.1: Statistical properties of the continuously compounded daily return (logreturn) of the NASDAQ index for the 4-year period from Jan 1 2010 to Jan 1 2014. The unbiased skewness and kurtosis are -0.409788 and 6.562415, respectively.

We see that the mean is smaller than the median, which implies left skewness. This is also indicated by the negative skewness value. The kurtosis is 3 for a Gaussian distribution. The larger value of 6.5 observed here indicates that the logreturn data are not Gaussian, refuting the common assumption of normal distributed logreturns or, equivalently, lognormal distributed returns. The larger kurtosis means that the data exhibit a larger peakedness than the normal distribution as well as heavier tails, which we also explore visually later.

6.1.2 Exploratory Data Analysis

In this subsection we visually explore the daily NASDAQ data, focusing primarily on analysing the distributional properties in more detail. First, however, we plot in Fig. 6.1 the closing stock prices and logreturns for the NASDAQ Index.

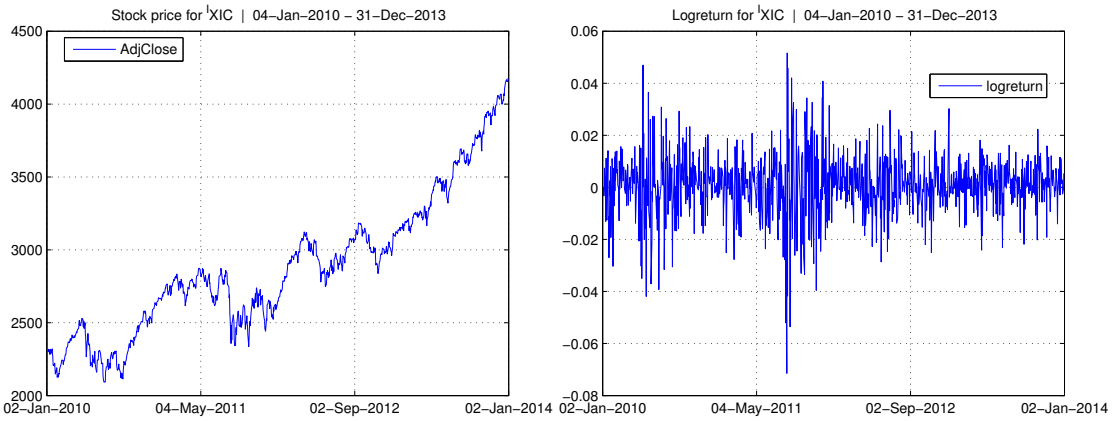

Figure 6.1: Adjusted closing prices (left) and continuously compounded returns (logreturn) (right) for the NASDAQ Index for the 4-year period from 01/01/2010 to 01/01/2014.

The left plot in Fig. 6.1 shows an overall upward trend, especially during the last year. From the right plot we see that the logreturns appear to fluctuate around a constant level (which is the mean $\bar{r} = 0.000590$ shown back in Table 6.1). We also observe some large spikes, or deviations, which correspond to the extreme events—large rises and drops in the price. As can be seen, these volatile periods often gather in groups (a few of which are very conspicuous in the plot), hinting

at *volatility clustering* in the data. Section 6.1.3 gives a more detailed, quantitative analysis of this phenomenon.

Now we turn to the qualitative distribution analysis. Fig. 6.2 shows a probability (P-P) plot of the empirical data, along with some theoretical distributions. Plotting the cumulative distribution functions against each other, a probability plot works as a qualitative test for a particular distribution.

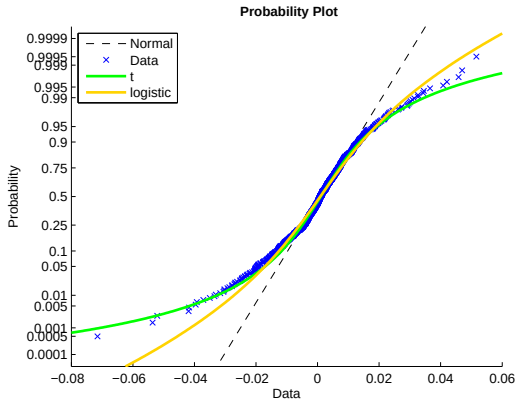

Figure 6.2: Probability plots including the normal distribution, t location-scale distribution, and the logistic distribution.

Fig. 6.2 shows that the Normal distribution (straight dashed line) is a fine model for the *body* (the range $[0.1; 0.9]$ on the vertical axis) of the empirical distribution of the logreturn data. But it fails to model the tails, lending much too low probabilities to extreme events (large gains or losses). This is also what was indicated by the large kurtosis back in Table 6.1; there are simply too many and too large fluctuations observed on the stock markets than expected under the Gaussian distribution. This can have dire consequences in risk management; assuming a Gaussian model significantly underestimates the potential risk associated with an investment.

The logistic distribution (gold curve) appears to be a better model than the Gaussian, following the data further along the tails. Still, the agreement is not fully satisfactory—especially in the bottom/left tail. This is actually the most important part since it reflects the extreme negative events—large drops and, potentially, market crashes. The inconsistencies thus render the logistic distribution inadequate.

Lastly, we have included the t distribution (green curve) which turns out to provide a truly remarkable model. With its heavier tails, it neatly accounts for the extreme events observed in the NASDAQ data; even some of the largest price drops (see the three points in the bottom left of Fig. 6.2) are very modelled.

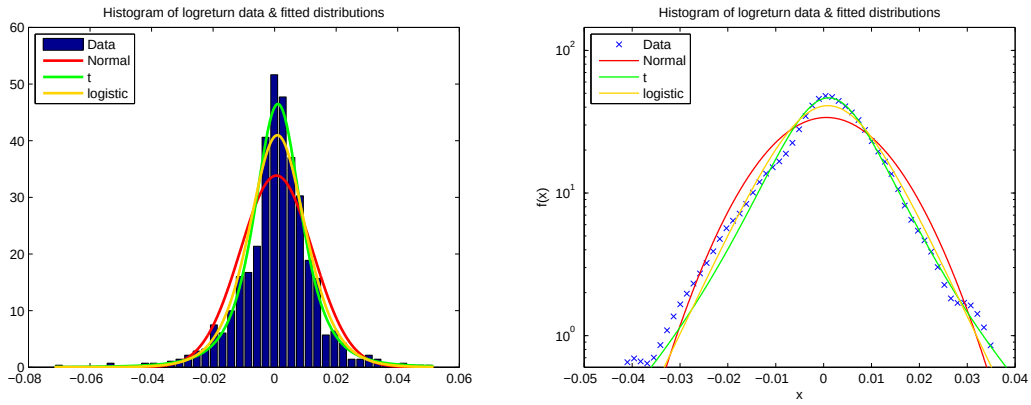

Figure 6.3: Histogram (left) and empirical probability density function (right) of logreturn data with fitted distributions. Left: normal scale. Right: log scale. In the right plot, the "squiggles" at the bottom of each tail of the logreturn graph are due to the kernel smoothing method used to obtain the empirical PDF and therefore not inherent in the real, observed data.

We continue the distribution analysis with Fig. 6.3. The left part shows a histogram of the logreturn data along with fitted normal, t, and logistic distributions. The right part shows the estimated empirical probability density function (PDF), also with the PDFs of the fitted distributions. Both plots clearly show that the t distribution provides the best model for the logreturn data; it better encompasses the peakedness of the data as well as the heavier tails. Indeed, the t distribution fits the data very well down the right tail (representing large gains). But the fit is not that perfect along the left tail, where we see the t distribution (still better than the normal and logistic distribution, though) giving slightly lower probability densities than actually observed; i.e. the risk of large losses is not perfectly, or sufficiently, modelled by the t distribution. Related to the well-known and intuitive phenomenon of people acting and reacting differently to large losses than large gains, this points in the direction of an asymmetrical empirical distribution.

In conclusion, our quantitative and qualitative analyses provide empirical evidence that, in contrast to common assumptions—the Gaussian is an inadequate model for the distribution of stock returns. The t distribution is a better model that lends more realistic (higher) probabilities to extreme events.[2]

6.1.3 Time Series Analysis

In this section we perform a traditional time series analysis of the daily NASDAQ logreturns, specifically with a focus on autocorrelation and conditional heteroscedasticity (volatility clustering). The results and conclusions will be of valuable use for model construction and prediction in the following sections.

Autocorrelation

Figure 6.4 shows the sample autocorrelation function (ACF) and partial autocorrelation function (PACF) for lags up to 100. Error bounds are plotted at $2\sigma = \pm 0.0631$ to aid in determining

[2] An even better model yet might be given by the family of α-stable distributions, as explored in [46].

the significance. Both the ACF and PACF show several autocorrelation lags that stand out. As it can be difficult to visually ascertain the exact lags, our program prints these to the screen. The significant ACF lags (with absolute values greater than 2σ) are: 3, 5, 10, 22, 25, 39, 47, 48, 67, 88. The significant PACF lags are identical, but with two fewer values: 3, 5, 10, 22, 25, 39, 48, 88. Of course, one can argue whether data as far in the past as e.g. 88 days has any sort of impact on the present values; however, these results could also point in the direction of *seasonality* in the time series, and more specifically with a period of approximately 20.

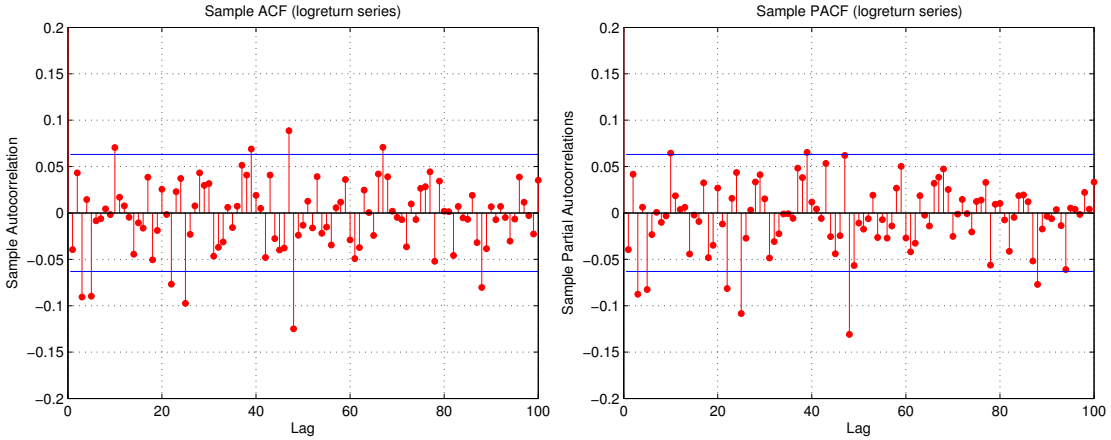

Figure 6.4: Sample Autocorrelation Function (ACF) (left) and Partial Autocorrelation Function (PACF) (right) of the NASDAQ logreturn data. Error bounds (blue) at 2 standard deviations. Lags up to 100 are included. The lag-0 autocorrelation is 1 by definition. The y-axes have been re-scaled to increase emphasis on the interesting (non-zero) lags.

The sample ACF and PACF mainly provides visual, qualitative support for the claim that the logreturn data exhibits significant autocorrelation. We can analyse this in more detail by performing a *Ljung-Box Q-test*. The Ljung-Box Q-test is a quantitative test for autocorrelation, assessing the significance of autocorrelation at multiple lags jointly. The null hypothesis H_0 is that the first m autocorrelations are zero. [40] suggests a default value of $m = \log N$, which gives $m = \log(1005) \approx 7$ in our case. We have performed the test at the 5% significance level for several different values of m (though not too large, since the test will otherwise lose power). The results are shown in Table 6.2. Evidently, the Ljung-Box Q-tests all yield $h = 1$, whereby the null hypothesis is rejected at the 5% level at all four lags. The conclusion is that there is significant autocorrelation in the logreturn series. This is support against a fully random market, and motivates the use of past values for predicting future values.

m	χ^2	Q	p	h
5	11.0705	20.1084	0.0012	1
10	18.3070	25.3054	0.0048	1
15	24.9958	27.8140	0.0228	1
25	37.6525	51.1039	0.0016	1

Table 6.2: Ljung-Box Q-test for autocorrelation. H_0: First m autocorrelations are zero. Tests are performed at the 5% significance level ($\alpha = 0.05$) and for several different m. χ^2 is the χ^2 critical value, Q is the test statistic, p is the p-value. $h = 1$ rejects the null hypothesis, while $h = 0$ indicates that H_0 cannot be rejected.

Conditional Heteroscedasticity

We now turn to the investigation of conditional heteroscedasticity (ARCH effects), i.e. volatility clustering. An indication of this feature is autocorrelation in the *squared* series, and so we plot in Fig. 6.5 the sample ACF and PACF of the squared logreturn.

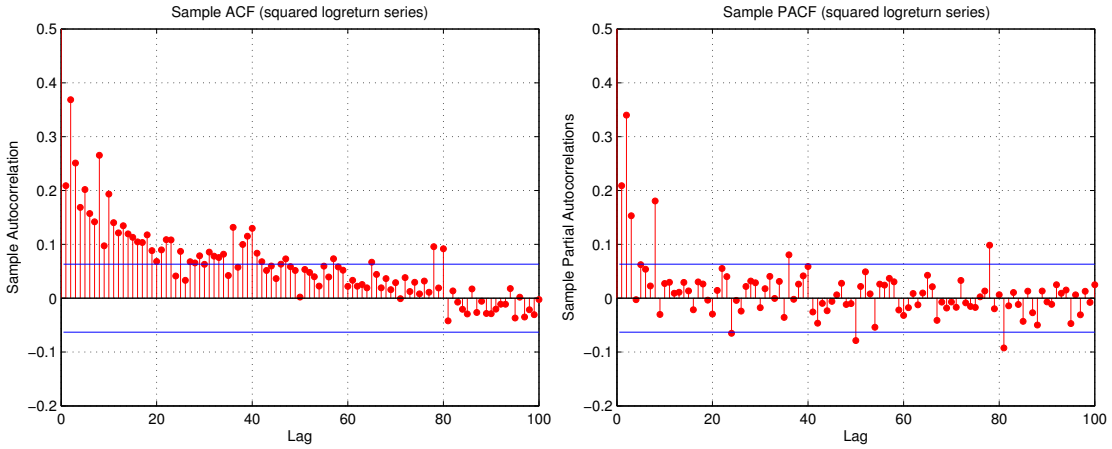

Figure 6.5: Sample ACF (left) and PACF (right) of the *squared* logreturn data. Error bounds (blue) at $2\sigma = 0.0631$.

As indicated by the left plot in Fig. 6.5, and specified exactly by the output of the written program, the significant lags of the sample ACF count almost all lags from 1 to about 50, as well as a few after this. The significant lags for the PACF are fewer. In any case, though, the plots of the sample ACF and PACF provide clear visual support for significant autocorrelation in the squared series, and thus volatility clustering.

But, let us again assess this more quantitatively. This can be done via the Ljung-Box Q-test from before, only now we naturally apply it to the squared logreturn data. The results are shown in Table 6.3. The tests for the different values of m give p-values of zero (and hence $h = 0$), indicating that the null hypothesis of no autocorrelation in the squared series (i.e. of no ARCH effects, or volatility clustering, in the logreturn series) is rejected at the 5% level at all four lags. The conclusion is that there is significant autocorrelation in the *squared* series, and hence that the observed logreturn data exhibits conditional heteroscedasticity, i.e. volatility clustering.

Conditional heteroscedasticity (volatility clustering) can also be checked via *Engle's ARCH*

m	χ^2	Q	p	h
5	11.0705	314.7891	0	1
10	18.3070	479.4108	0	1
15	24.9958	560.6133	0	1
25	37.6525	651.8233	0	1

Table 6.3: Ljung-Box Q-test for autocorrelation in *squared* series, i.e. for ARCH effects (volatility clustering) in normal series. H_0: First m autocorrelations are zero. Significance level: $\alpha = 0.05 = 5\%$. χ^2 is the χ^2 critical value, Q is the test statistic, p is the p-value. $h = 1$ rejects the null hypothesis, while $h = 0$ indicates that H_0 cannot be rejected.

test. This tests the null hypothesis H_0 of no conditional heteroscedasticity against the alternative of an ARCH(n) model with n lags (which is equivalent to a GARCH($n/2$,$n/2$) model for even n). The test results are shown in Table 6.4. In all three cases of $n = 2, 4, 6$, the null hypothesis is soundly rejected ($h = 1$, $p = 0$) in favor of the respective ARCH(2), ARCH(4), and ARCH(6) alternatives. The F-statistics for the tests are much larger than the critical values from the χ^2 distribution with n degrees of freedom. Of course, since tests were conducted for different n, and since all tests rejected the null hypothesis, the question arises of which of the three alternatives to choose. Since we are interested in simple, parsimonious models (to avoid unwarranted complexity), we keep with the ARCH(2) model, which is equivalent to a GARCH(1,1) model. In any case, though, the conducted ARCH tests conclude that there is significant volatility clustering in the logreturn series.

n	χ^2	F	p	h
2	5.9915	157.6105	0	1
4	9.4877	175.0208	0	1
6	12.5916	181.0270	0	1

Table 6.4: Engle's ARCH test for conditional heteroscedasticity (volatility clustering). H_0: No ARCH effects. H_a: ARCH(n) model with n lags. Significance level: $\alpha = 0.05$. χ^2 is the χ^2 critical value, F is the test statistic, p is the p-value. $h = 1$ rejects the null hypothesis H_0 in favor of the alternative H_a, while $h = 0$ indicates that H_0 cannot be rejected.

This concludes our analysis of autocorrelation and conditional heteroscedasticity in the logreturn series of the NASDAQ Index. We have found qualitative and quantitative evidence of both significant autocorrelation and significant volatility clustering. The presence of these phenomena motivates the use of past data to predict future data, as is done in the many sections to come. We begin in the following section with conditional mean and variance models.

6.1.4 ARMA-GARCH Models

In the previous section we saw that the logreturns of the NASDAQ Index exhibited both significant autocorrelation and volatility clustering. This makes it plausible to try to model the mean return with conditional mean models (e.g. ARMA) and the volatility, or variance, with conditional variance models (e.g. ARCH or GARCH).

Modelling the Conditional Mean

We begin by modelling the conditional mean, for which we choose the autoregressive moving-average (ARMA) model (the AR and MA models are both subclasses of this more general model). It should be noted that an ARMA model for the logreturn data (which is stationary) is equivalent to an ARIMA model for the non-stationary price data. For now, we will not include exogenous covariates (leading to e.g. ARMAX and ARIMAX models) but rather stick with the ARMA/ARIMA models that use lagged response data.

In order to identify the best lags for an ARMA model for the logreturn data, we can consult the sample ACF and PACF plots in Fig. 6.4 in the previous section. The specific significant lags can be hard to read off from the plots, but the written program outputs these values, which were also presented in the discussion of the plots. However, there appears to be an awful lot of seemingly significant lags, and it is not certain that we will obtain an optimal model by just choosing some or all of these lags (indeed, we would surely obtain a bad, complex, highly non-parsimonious model by choosing all of them). This does not render the ACF and PACF plots useless—far from it—but just means that they are used for the general assessment of autocorrelation, and not the specific identification of the lags to be used in the ARMA model.

Instead, we pursue another approach for identifying the best lags p and q in the ARMA(p, q) model. Namely, we fit several models with different lag choices, thus searching the lag parameter space for the combination that gives yields the optimal model. There is some ambiguity inherent the word 'optimal'; we need a measure to assess the adequacy, or goodness, of the model. As discussed in Section 4.4.1, we choose as measures the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). We then fit all combinations of lags $p = 1, \dots, 10$ and $q = 1, \dots, 10$ (a total of 100 models), compute the AIC and BIC for each, and finally select the optimal model as the one yielding the lowest AIC and BIC[3]. This is easily done via the attached MATLAB program, which outputs matrices (the rows and columns of which indicate the lags p and q) of AIC and BIC values, and then finds the minimum value and its index (which is the optimal combination (p, q)). The results are:

Minimum AIC $= -5430.4$ obtained for $p = 6$, $q = 3$.

Minimum BIC $= -5392.2$ obtained for $p = 1$, $q = 1$.

Evidently, the two information criteria point at different optimal models; the AIC chooses an ARMA$(6, 3)$ model, while the BIC chooses an ARMA$(1, 1)$ model. The ARMA$(1, 1)$ model is the more parsimonious choice, and we will mainly focus on this one. However, for the sake of curiosity, we will not entirely dismiss the ARMA$(6, 3)$ model suggested by the AIC. In fact, the BIC value for this model is -5382.3, which is one of the lowest values next after the minimum listed above.

Modelling the Conditional Variance

We now turn to modelling the conditional variance using the ARCH/GARCH-type of models discussed in Section 4.7. The sample ACF and PACF in Fig. 6.5 show very significant volatility clustering. This was further supported by the Ljung-Box Q-test and Engle's ARCH test. The conditional heteroscedastic nature of the data makes it crucial to properly model the variance rather than just take it as being constant.

Regarding identification of the best parameters for the GARCH(u, v) model, we can turn back to the ARCH tests conducted in the previous section. The test results concluded that the null

[3]If the AIC and BIC choose different models we may select the more parsimonious choice.

hypothesis was rejected in favor of a variety of ARCH models, the simplest, most parsimonious one of which was an ARCH(2) model, equivalent to a GARCH(1,1) model. This gives some initial incentive for choosing this as our conditional variance model.

However, let us still pursue the same approach as for the ARMA model above, searching the parameter space for the best combination u, v that minimizes the AIC and BIC. Using all combinations of $u = 1, \ldots, 4$ and $v = 1, \ldots, 4$, our MATLAB program yields the following results:

Minimum AIC $= -5622.1$ obtained for $u = 1$, $v = 2$.

Minimum BIC $= -5602.8$ obtained for $u = 1$, $v = 2$.

The AIC and BIC agree that the optimal model is a GARCH$(1, 2)$. This is very close to the GARCH(1,1) model argued for above, only with one additional GARCH lag. Even though the AIC/BIC-based parameter space search yielded GARCH(1,2) as the best model, let us investigate whether we can be without this extra GARCH lag and thus make do with a GARCH(1,1) model. For this purpose, we conduct a likelihood ratio test to compare the restricted, nested GARCH(1,1) model to the unrestricted, extended GARCH(1,2) model. The degrees of freedom for the test are 1, due to a difference of 1 in the number of parameters. The test, performed at the 5% significance level, yields a very low p-value of 0.00078126. The null GARCH(1,1) model is therefore rejected in favor of the GARCH(1,2) alternative.

In conclusion, a GARCH(1,2) appears to be the optimal choice for a conditional variance model.

However, as discussed in Section 4.7, there are conditional heteroscedastic models that are even more advanced than GARCH. Among these we find the E-GARCH, which, as opposed to GARCH, accounts for the so-called *leverage effect*[4] often observed in financial time series[5]. As done in the attached program, we fit both an E-GARCH(1,1) and E-GARCH(1,2) model and perform an AIC/BIC-based comparison of these models to each other as well as the hitherto optimal GARCH(1,2) model. The results are:

GARCH(1,2): AIC $= -5622.1$, BIC $= -5602.1$

E-GARCH(1,1): AIC $= -5669.4$, BIC $= -5650.2$

E-GARCH(1,2): AIC $= -5665.4$, BIC $= -5636.6$

Both E-GARCH models yield lower AIC and BIC values than the GARCH(1,2) model. The lowest values are obtained by E-GARCH(1,1), which will therefore be our choice.

In summary, the optimal conditional mean and variance models are ARMA(1,1) and E-GARCH(1,1).

Building the Composite Model

We now build the ARMA(1,1) and E-GARCH(1,1) composite model and fit it to the data. The resulting parameter estimates are shown in Table 6.5.

For an E-GARCH(1,1) model, the GARCH and ARCH coefficients are expected to be positive, while the leverage coefficient is expected to be negative (due to the fact that large unanticipated shocks should increase the variance). The parameter values obtained here are in excellent agreement with these expectations. Generally, everything looks good for the E-GARCH model, with

[4]The leverage effect covers the feature of the asset reacting differently to large negative changes than positive changes.

[5]We also observed this for the NASDAQ data when we back in Fig. 6.3 found evidence for an asymmetric distribution.

Model	**Parameter**	**Value**	**Std. Error**	t **statistic**
ARMA(1,1)	Constant	0.000862368	0.000601128	1.43458
	AR{1}	-0.774204	0.388892	-1.99079
	MA{1}	0.75321	0.406287	1.85389
E-GARCH(1,1)	Constant	-0.515226	0.0833525	-6.18129
	GARCH{1}	0.943026	0.00925724	101.869
	ARCH{1}	0.127586	0.0308727	4.13264
	Leverage{1}	-0.191869	0.0232118	-8.26601

Table 6.5: ARMA(1,1) and E-GARCH(1,1) composite model parameters. The notation is such that AR{1} denotes the coefficient to the variable with lag 1. Leverage is the new parameter introduced with the E-GARCH model.

low relative standard errors and hence large t-statistics indicating statistical significant parameters. As for the ARMA(1,1) model, the t-statistics very close to 2 place the parameters just on the border of statistical significance. However, this also has to do with MATLAB's internal way of handling ARMA and ARIMA models. As mentioned earlier, there are two ways to go about the model construction process; (1) manually difference the non-stationary price data to obtain e.g. the stationary logreturn data and then use an ARMA(p, q) model, or (2) use an ARIMA(p, d, q) model with a differencing parameter of $d = 1$ on the original price data. MATLAB's internal programming is such that the maximum likelihood parameter estimates obtained using the second approach are more robust than those obtained using the first approach. We should try it just to make sure. This has been done in the attached MATLAB program, where we fit an ARIMA(1,1,1) (with differencing parameter $d = 1$) and E-GARCH(1,1) to the original closing price data. Indeed, as outputted by the program, the t-statistics for the AR1 and MA1 coefficients are -2.58177 and 2.39499, respectively, both of them now greater than 2 (as opposed to -1.99 and 1.85 as seen in Table 6.5) and thus statistically significant. As for the other parameters, the ARMA/ARIMA Constant remains only borderline statistically significant, while all the E-GARCH(1,1) parameters still have very large t-statistics and are thus highly significant.

Model Diagnostics

It is now time to scrutinize more thoroughly the constructed composite model. For this purpose, we infer the residuals and perform diagnostic checks. The residuals r are standardized to $\hat{r} = r/V$, where V is the inferred conditional variance. Several diagnostics plots are shown in Fig. 6.6.

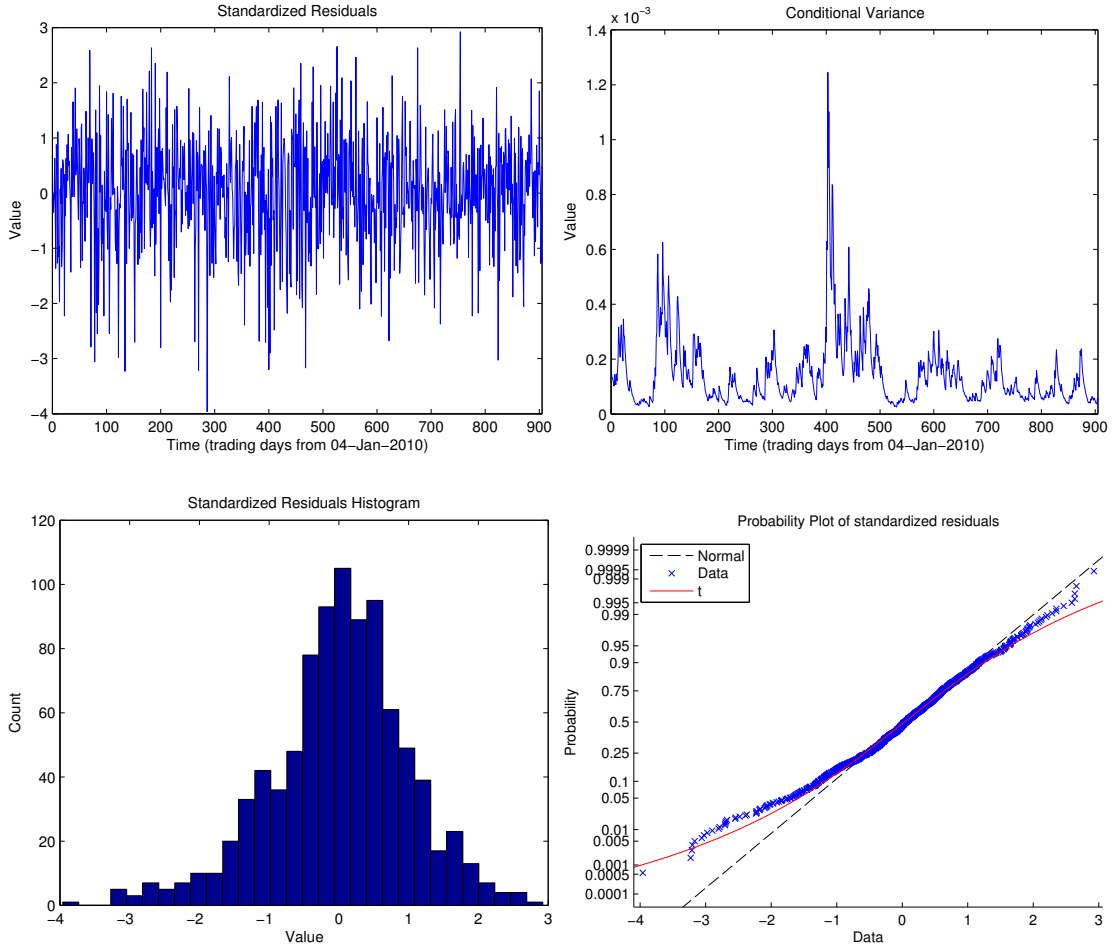

Figure 6.6: Diagnostic checks for ARMA(1,1) and E-GARCH(1,1) composite model. Top left: Time plot of standardized residuals. Top right: Time plot of conditional variance. Bottom left: Histogram of standardized residuals. Bottom right: Probability plot of standardized residuals.

The top right plot in Fig. 6.6 shows a conditional variance that is far from being constant. We observe several peaks, the most dominant one of which occurs after about 400 trading days. This corresponds to the large and sudden drop shortly after May 2011 that we also saw in the earlier Fig. 6.1. These increases in variance indicate increases in volatility, and it is important to note their durations and positions; namely, we observe periods of large variance followed by periods of low variance. The positions, or occurrences in time, of these periods agree with the corresponding periods of high and low volatility observed in the logreturn plot in the earlier Fig. 6.1.

The top left plot in Fig. 6.6 shows a time plot of the standardized residuals. The values range from about -4 to 3. The crucial thing to notice is that there are quite a lot of these extreme event (especially negative values, corresponding to large price drops). This is also seen by the histogram in the bottom left of Fig. 6.6, where we observe a somewhat large amount of $\pm 2\sigma$ events. In our model, these residuals, or innovations/shocks, are by default modelled by the

Normal distribution. However, the amount of extreme events significantly exceeds the amount expected under a standard Normal distribution. This indicates that the Gaussian might not be the best model for the innovation distribution, a claim that is further supported as we turn to look at the bottom right plot in Fig. 6.6. This shows a probability plot of the standardized residuals (blue) along with the Normal distribution (black) and t distribution (red). Along the right tail of extreme positive events (large increases), the data points are placed just between the Normal and t distributions, making either model somewhat plausible in this region. Both distributions model the data perfectly in the "body", with the t distribution performing better as we slowly move down and to the left. As we move further along the left tail, the Normal distribution is way off, significantly underestimating the probability of extreme negative changes and thus, in financial jargon, the risk of large losses! This is very crucial in risk management, and the Normal distribution's dramatic shortcoming in this regard renders it wholly implausible for practical use. The t-distribution, however, performs remarkably well, closely following the data points along the left tail—and even almost correctly modelling the most extreme -4σ event (it slightly overshoots and ascribes a larger probability than observed, but better an overestimation of risk than an underestimation!)

In conclusion, we find that the Normal distribution fails to properly model the large values of the standardized residuals, while the t distribution does a very good job.

We therefore fit an ARMA(1,1) and E-GARCH(1,1) composite model with a t distribution for the innovation process. The resulting model parameters are shown in Table 6.6.

Model	**Parameter**	**Value**	**Std. Error**	t **statistic**
ARMA(1,1)	Constant	0.0014165	0.000645252	2.19527
	AR{1}	-0.711422	0.468072	-1.5199
	MA{1}	0.68691	0.484904	1.41659
	DoF	6.90726	1.99892	3.45551
E-GARCH(1,1)	Constant	-0.504189	0.104268	-4.83551
	GARCH{1}	0.945376	0.0115045	82.1741
	ARCH{1}	0.132051	0.0425107	3.10631
	Leverage{1}	-0.219608	0.0323908	-6.77994
	DoF	6.90726	1.99892	3.45551

Table 6.6: Parameters for ARMA(1,1) and E-GARCH(1,1) composite model with a Student's t innovation distribution. DoF is the number of degrees of freedom of the t distribution. The notation is the same as in Table 6.5

The coefficient estimates change slightly when the t distribution is used for the innovations. The new model fit has one additional parameter; the t distribution degrees of freedom. As can be seen in Table 6.6, the estimated degrees of freedom are relatively small (about 7), indicating significant departure from normality.

Next, we perform an AIC/BIC-based comparison of the new model with a t innovation process to the old model with a Gaussian innovation process. The results are:

ARMA(1,1) and E-GARCH(1,1) model with

- Gaussian innovation: AIC $= -5666.3$, BIC $= -5632.7$
- t innovation: AIC $= -5685.6$, BIC $= -5647.2$

The second model achieves lower (more negative) AIC and BIC values than the first, supporting our use of a t innovation process in the ARMA(1,1) and E-GARCH(1,1) composite

model.

Forecasting

Recall from earlier that the NASDAQ dataset analysed and used in all of the above comprised 1006 observations of daily prices (1005 for the logreturns). The data have been split into a training set (90%) and a test/validation set (10%). For constructing the conditional mean and variance models above, we have solely made use of the training set. The test set thus contains wholly unused, unseen data—as if it was future data that we did not know. In this way, the test data can be used to validate and assess the forecasts we are about to make.

The ARMA-GARCH framework provides an easy way of doing **multi-period ahead forecasting**, which we begin with. Our ARMA(1,1) and E-GARCH(1,1) composite model is used to forecast both the price, logreturn, and conditional variance over a period equal to that of the test set (i.e. 100 days). The results are shown in Fig. 6.7.

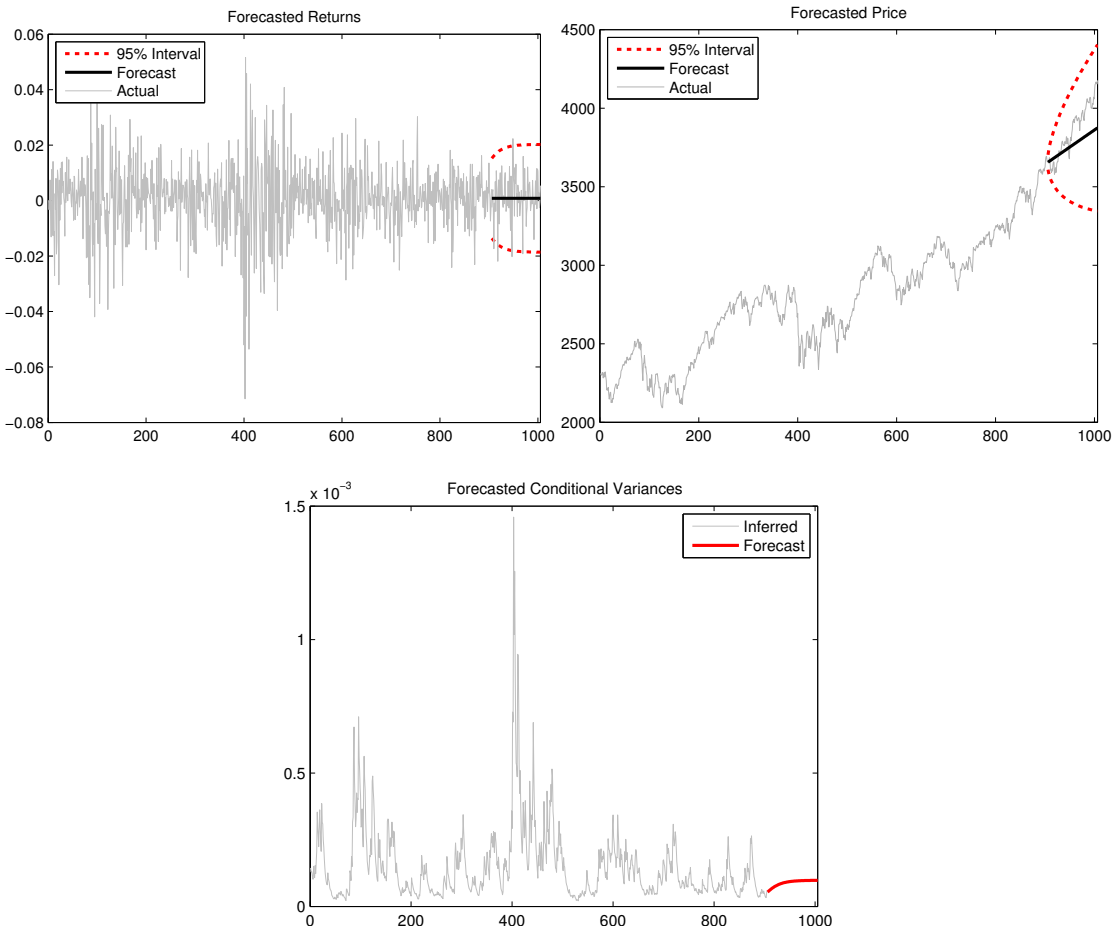

Figure 6.7: Forecasted returns (top left), prices (top right), and conditional variances (bottom) by the ARMA(1,1) and E-GARCH(1,1) composite model. The unused, unseen test data have been included in the top plots to assess forecasting performance.

It should be noted that the ARMA model is a linear time series model. It models the conditional *mean* of the data, which is why its forecasts are linear in nature.[6] Over multiple periods, the model is obviously improper for trying to predict the specific returns or prices and their fluctuating nature. But this does not make it useless at all; indeed, as shown in Fig. 6.7, it provides a prediction of the general *trend*. In addition, we have also calculated error bounds on these forecasts. As can be seen in the top left plot in Fig. 6.7, the actual, observed returns (gray) fit rather well inside the 95% error bounds (red). The top right plot is very interesting, showing the forecast of the conditional mean of the price (the width of the forecast interval grows in time due to the price process being non-stationary, as opposed to the stationary logreturn series). The forecast clearly indicates an upward, increasing trend. Looking at the actual data from the test set, we observe, of course, some fluctuations—but most importantly the presence of an actual upward trend! Trend-wise, the forecast thus fits very well with the actual, observed data.

In addition to this qualitative assessment of the forecasts, we can also evaluate the model's performance in a more quantitative manner. Equipped with the actual data from the test set, denoted y_i for the ith observation (in a total of N observations), and the forecasts from the model, denoted $\tilde{y}_i$, we can compute a variety of error measures. From the numerous choices available, we have chosen the popular mean-squared error (MSE) and the mean-absolute percentage error (MAPE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\tilde{y}_i - y_i)^2 \tag{6.1}$$

$$\text{MAPE} = \frac{100}{N}\sum_{i=1}^{N}\left|\frac{\tilde{y}_i - y_i}{y_i}\right| \tag{6.2}$$

We have chosen both a non-percentage and a percentage-based measure on purpose. The MSE clearly depends on the type and order of magnitude of the data. It is not so adequate if the magnitude of the data varies a lot over the N observations considered (which it might do for stock prices).[7] However, it is a common measure with a good interpretability. Due to its percentage nature, the MAPE is not affected by such scale changes. But this measure may prove inadequate if the data are very small (such that the denominator is close to zero).

However, since we performed multi-step ahead prediction (with linear forecast curves) and mainly focused on trend prediction, such error measures are not proper to use here unless one compares the predicted trend with a quantity measuring the observed trend.

The error measures are more suitable for assessing the performance of **one-step ahead prediction**, which is the type of prediction we are most concerned with in all of the following. We have implemented it by forecasting over a single period (in the test set) at a time, instead of forecasting over the all periods in the test set collectively (e.g. over 100 periods as in the above example).

The results for our ARMA(1,1) and E-GARCH(1,1) composite model are shown in Fig. 6.8.[8]

[6] Oscillations may be included in seasonal versions of the models.

[7] Of course, this can be resolved using scaled, stationary prices. But this also reduces the interpretability.

[8] Note that the price data has been smoothed using a 5-period Exponential Moving Average. We discuss the reason in detail in Section 6.1.6. Smoothing the prices also makes the forecasts a type of trend prediction, but in a more precise, refined way than we saw earlier.

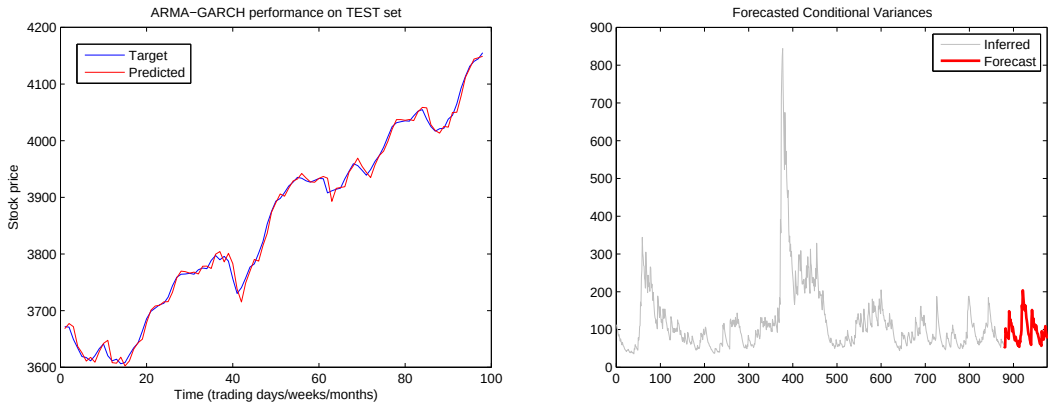

Figure 6.8: One-period ahead forecasting results for the ARMA(1,1) and E-GARCH(1,1) composite model. Left: Stock price (zoomed in on the unseen test set). Right: Conditional variance.

The left plot of the predicted and actual stock prices zooms in on the test set, which is the part of interest. As opposed to the previous Fig. 6.7, the forecast is no longer straight; or, actually it is straight in that it is made up of linear pieces, one for each period. We see that the predictions follow the actual data extremely well, but mainly so when the price is trending and headed in a particular direction. When the price makes a sudden movement in the opposite direction (at e.g. 10 and 40 days on the x-axis), our model stays in its original direction—keeping its momentum, so to speak—resulting in a wrong prediction.

Looking at the right plot, the forecasted conditional variance has also changed shape (compared to Fig. 6.7) now that we predict only one step ahead. It is interesting to see how the forecast (in red) retains the oscillatory behaviour observed in the inferred values between periods 700-850 on the x-axis.

We can evaluate the stock price predictions quantitatively the above error measures. Another, very interesting, error measure that has become available now that we perform one-step ahead forecasting is the Hit Rate accuracy, which measures the percentage of correct movement direction (up/down) predictions. The results in this case are:

$$\text{MSE} \quad = 90.32$$

$$\text{MAPE} \quad = 0.188\ \%$$

$$\text{Hit Rate} = 68.04\ \%\ (66 \text{ out of } 97)$$

The MSE indicates that the squared errors of our model's predictions are, on average, 90 units (US dollars for the NASDAQ Index). So the average error is of the order $\sqrt{90} \approx 9.5$ USD. The MAPE value indicates that, on average, our model shoots wrong by about 0.19 %. The Hit Rate says that our model correctly predicted the movement direction 66 times (out of the 97 test set observations), corresponding to 68 %. These results are actually not bad. Later, in Section 6.2, we shall compare them to the results obtained using other methods.

6.1.5 Neural Network Models

As mentioned in the theoretical section 5.2, the type of neural networks employed in this study are non-linear autoregressive with exogenous (external) input (NARX networks). The reason

is that these networks, compared to other types, are highly adequate for time series prediction because of their ability to more easily utilize the temporal nature of the data and its potential autocorrelations. Moreover, they also allow for the inclusion of other time series as exogenous input.

When it comes to neural networks, there are numerous parameters and options that can be tweaked and adjusted in the attempt to find the optimal model with the very best performance. For example, there are the questions of the type and amount of data to use, including what to choose for the number of delays of the response data and the external inputs, how to split the data into training/testing/validation samples, how to pre-process it, how to construct the network and choose the number of layers and hidden neurons, and then there is the choice of training algorithm, stopping criteria, performance measures, and so on and so forth. With all these aspects to consider, all this freedom of choice, the task of building a neural network can suddenly seem quite overwhelming. It would probably be a vain goal to try to cover all of these aspects, varying each and every one of them in turn in an endless search for the perfect model. Indeed, doing that is beyond the scope of this thesis. We will approach the task in a somewhat more humble manner, starting small and then gradually broadening our reach. In more specific terms, we approach the above points and options as follows:

Data type and amount: To begin with, we will naturally be considering the same NASDAQ data as used and analysed in the previous sections. For details, see Section 6.1.

Inputs: The analysis in Section 6.1.3 concluded that there is significant autocorrelation and that lagged response data may thus be plausible predictors. Regarding exogenous inputs, we have no a priori knowledge of which combinations will work. Here, we consider an arbitrary combination. Later, in Section 6.2, we try different variations.

Data splitting: Drawing inspiration from similar research (see 1.3) and common practice, we manly stick with a 90%/10% split into training and testing data, respectively.[9]

Data pre-processing: Before being fed into the network, the data will be normalized to zero mean and unit variance and scaled into the range $[-1, 1]$.

Network architecture: We will use neural networks just a single hidden layer.[10] The number of hidden neurons will either be varied, or it will be the object of optimization using methods such as e.g. simulated annealing or a simpler search in parameter space. As default values, we can draw inspiration from the different rules of thumb (see Section 5.2.4) stating that the number of hidden neurons should be e.g. $n_h = \sqrt{n_i n_o}$ or $n_h = 0.75 n_i$ or $n_h = 2n_i + 1$, where n_i and n_o is the number of inputs and outputs, respectively.

Training algorithm: Several choices exist for training algorithms, as discussed in Section 5.2. Here, we narrowed down the possibilities through a comparison between each algorithm's merits and shortcomings and the particular application. Since our focus is the regression problem of the NASDAQ data, and since we would like algorithms that are not too slow, we will mainly be concerned with the Levenberg-Marquardt algorithm.[11]

[9] Note that the training data, comprising the first 90% observations of the entire data, is itself split into three parts (training/testing/validation) during the training of each neural network. The remaining 10%, which we call the test data, is wholly unused in the model building process (as if it didn't exist, like future, unknown stock prices), and is solely used for evaluating the forecasting performance of the network.

[10] It has been shown empirically that one, and at most two, hidden layers generally yield the best performance; networks with more than two hidden layers will generally not improve the results. [10],[61].

[11] Since Levenberg-Marquardt may converge too quickly, we will keep the Conjugate Gradient, Bayesian Regularization, and Adaptive Learning Rate algorithms as second options.

Stopping criteria: For designing each network, the training data is split in three parts (training/testing/validation). Training is stopped when the error on the validation set fails to decrease for six iterations.

Performance measures: The network performances and forecasting abilities will primarily be evaluated based on the Hit Rate accuracy (percentage of correct predictions of up and down movements) and the mean-squared error (MSE).

Training the First Neural Network

Now it is time to get to work with training the first neural network. We wish to predict the daily NASDAQ price y_t, and as input we will choose a number of past observations $y_{t-1}, \ldots, t_{t-d}$. We do not know the best value for this number of delays/lags d, but the autocorrelation analysis in 6.1.3 may be of some help. Here, we saw that there were significant autocorrelation at lags 3, 5, 10, etc. Later, we will try to use some of these very specific lags, but for this first experiment we set $d = 3$, using lags from 1 to 3.

We will try to include exogenous inputs straight from the start. However, we have no a priori knowledge of which inputs will work best. Therefore, for this initial experiment, we will just go ahead and choose two technical indicators; Momentum and Relative Strength Index (RSI). We will also use $d = 3$ lagged values of these inputs.

Then comes the question of the number of hidden neurons. This depends on the problem and the amount of inputs and outputs. Here, we use $n_i = 9$ inputs (3 lags for the response data plus 3 lags for each of the two exogenous inputs) and $n_o = 1$ outputs (the response data one step in the future). With these values, the rules of thumb mentioned above say that $n_h = \sqrt{n_i n_o} = \sqrt{9} = 3$, or $n_h = 0.75 n_i = 0.75 * 9 \approx 7$, or $n_h = 2n_i + 1 = 2 * 9 + 1 = 19$. They clearly do not agree. Too few may lead to the network not learning at all, while too many may lead to overfitting and hence poor generalization. Since complex problems (such as ours) require more hidden neurons, we may lean towards one of the larger values. However, instead of merely picking a value, we can also apply optimization techniques to find the value that minimizes some error measure, here the MSE, of the network performance on the training set. We have included a procedure for this in the attached MATLAB program. For this particular case, the lowest MSE was obtained for $n_h = 23$.[12]

We are now ready to train the network, which will be done using the Levenberg-Marquardt algorithm and the training set comprising 90% of the total data. As mentioned earlier, during network training, to ensure good generalization, the training data is itself divided into three parts (training/validation/testing) with a ratio of 70%/15%/15%, respectively. Fig. 6.9 shows several interesting plots that can be used to investigate the training process.

[12] It is important to note that the results from network training—and thus from this optimization—may vary due to the random initialization of the network weights and biases.

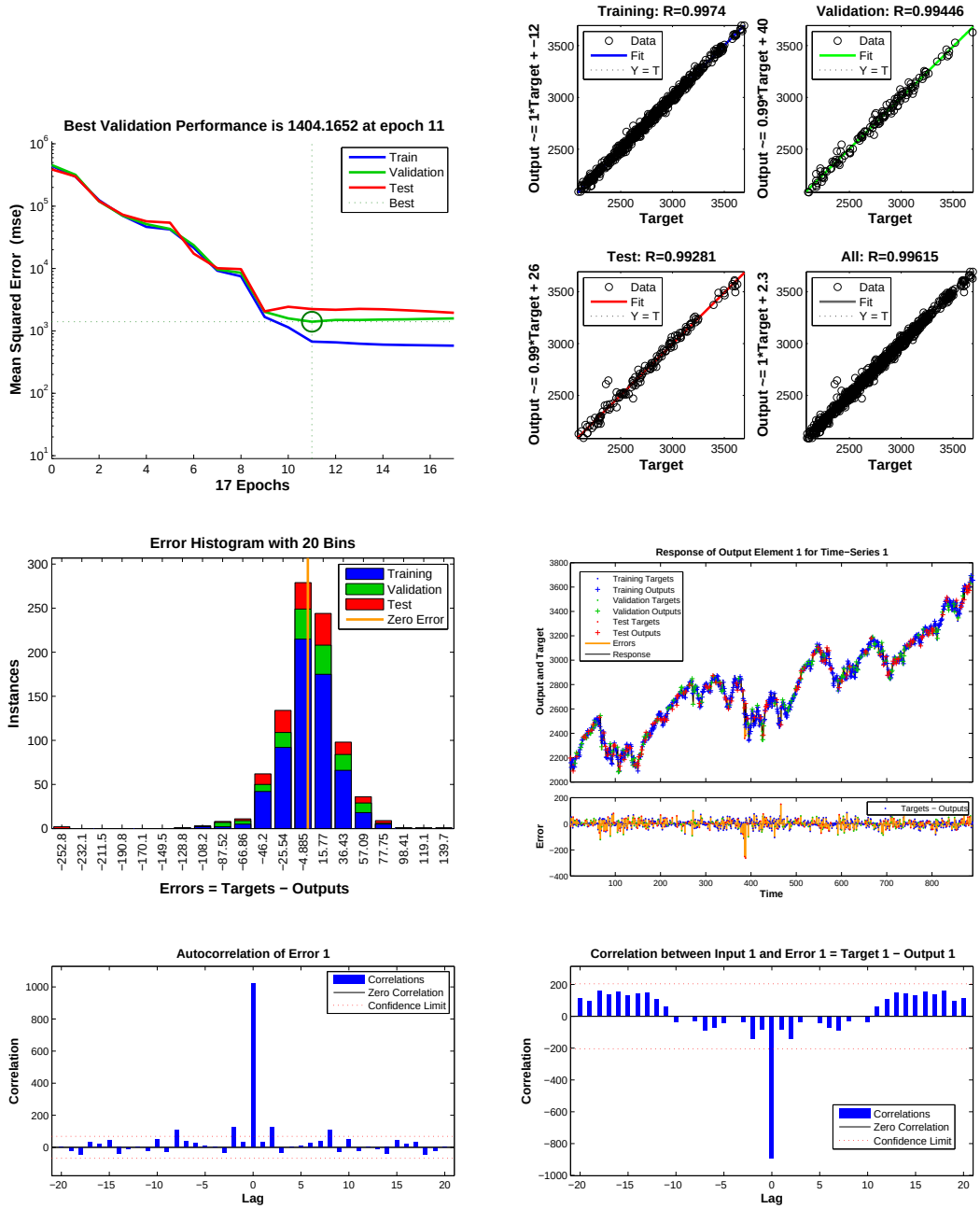

Figure 6.9: Plots for evaluation and diagnostics of network performance in the training process. Top left: Performance plot of MSE vs. epochs/iterations. Top right: Regression plot of output vs. targets. Middle left: Histogram of errors. Middle right: Plot of network response and actual target, including errors. Bottom left: Error correlation. Bottom right: Input-error correlation.

The top-left plot in Fig. 6.9 shows the mean-squared error (MSE) vs. the epoch/iteration for

both the training, validation, and test sets (all three are parts of the main training set comprising 90% of the data). The result is reasonable if the final MSE is relatively small, if the test and validation errors look similar, and if no significant overfitting occurred by the iteration at which the validation error is lowest. Of course, the MSE depends on the scale and type of the data, but we see that a minimum (possible local, though) has been found. Moreover, the validation (green) and test (red) curves follow each other almost identically up until epoch 9, after which there is only slight deviation. But the test curve does not increase significantly (only very little) before the validation curve, which indicates that no significant overfitting has occurred. We also see that the early-stopping criteria was met; training stopped after 17 epochs/iterations when the validation error (obtaining its minimum at epoch 11) failed to decrease for 6 iterations.

The top-right plot in Fig. 6.9 shows a linear regression between the outputs and the targets. This can be used to determine how well the network outputs match the actual targets. A perfect fit is when the data (black circles) fall along the 45-degree line, giving a correlation coefficient of $R = 1$. In this case, we have $R \geq 0.99$ for both the training, validation, and test subdivisions, indicating excellent agreement between estimated outputs and actual targets.

The middle-left plot in Fig. 6.9 is a histogram of the error $e_t = y_t - \tilde{y}_t$, where y and $\tilde{y}$ denote the target and output, respectively, and t is the time. The error histogram can give an indication of outliers, i.e. data points where the fit is significantly worse than for the majority of the data. Here, we see that there are about six cases in total where the network missed the actual target by roughly $e = 100$ or more. And then there is one extreme case (furthest to the left in the plot) where the network overshot by about 250. This extreme case corresponds to the sudden drop in the NASDAQ index around September 2011 (about 400 trading days after our starting date of January 1st 2010), which we also saw and discussed earlier in Fig. 6.1.

The middle-right plot in Fig. 6.9 shows the outputs, targets, and errors, and colors the data depending on their subpart in the main training set. It can be difficult to distinguish what happens in the top part, but the bottom part captures all of the essential things; the errors fluctuate up and down (corresponding to the network underestimating and overestimating the price, respectively). A large negative error stands out just before a time of 400; this is exactly the extreme case described in the previous paragraph.

The bottom-left plot in Fig. 6.9 shows the autocorrelation[13] of the error. This describes how the errors are related in time. For a perfect model there would only be one non-zero value occurring at lag 0. In this case, all lags but four—lags $-8, -2, 2, 8$ (effectively only two lags because they are symmetric and negative lags bear no meaning in that they refer to the future)—have correlations that fall within the confidence interval. This indicates an adequate model, but possibly with room for improvement via e.g. adding more inputs, increasing the number of delays of the inputs, and/or increasing the number of hidden neurons.

The bottom-right plot in Fig. 6.9 shows the input-error cross-correlation. This describes how the errors are correlated with the exogenous input. For a perfect model, all correlations are zero. In this case, all correlations fall within the confidence limits, indicating an adequate model. Again, however, there seems to be room for improvement.

With the network trained, i.e. the weights and biases determined, we now feed the entire training data into the network to calculate the estimated outputs. These outputs are then compared to the actual, observed data. For performance measures we consider the mean-squared error (MSE) and the percentage of correct up/down movement predictions (Hit Rate). The results are:

Network performance on training set (890 observations):

[13] See Section 4.1.2 for more on autocorrelation.

MSE = 1019.6

Hit Rate = 57.3% (510 out of 890)

This performance on the training set can be assessed more qualitatively by plotting the predicted outputs and the actual targets (stock prices), as done in Fig. 6.10.

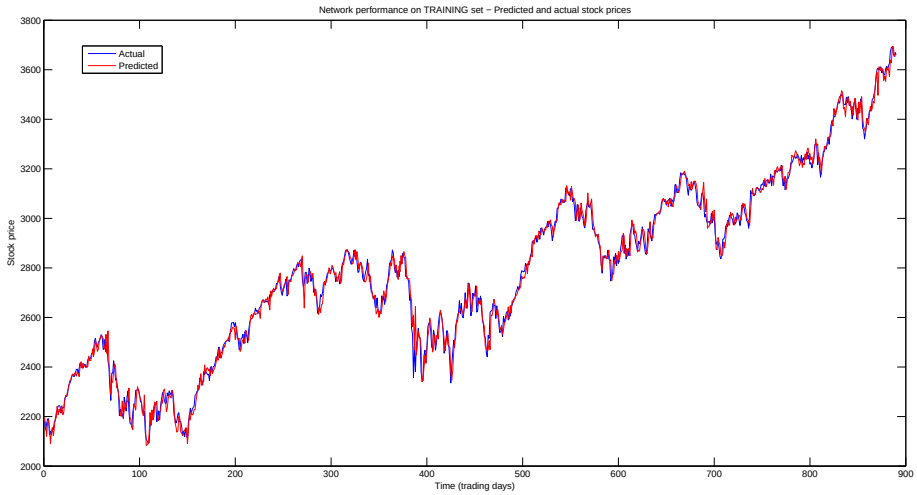

Figure 6.10: Neural network performance on TRAINING set. Vertical axis: Stock price. Horizontal axis: Time (trading days). Red curve: Predicted Prices. Blue curve: Actual prices.

The network seems to very accurately predict the data, which was, of course, also the data used to train it. But this also has to do with the viewpoint—that we are considering a frame of 890 points in time. When zooming in to take a closer look, the small deviation become more apparent. Overall, though, the network predictions follow the actual training data very closely. However, these results can only be used to evaluate the training, not the network's ability to generalize. For this purpose, we must turn to the test set.

The test set that will now be used is the one comprising 10% of the total data. This test data has not been used whatsoever for purposes of training the network and building the prediction model. It is entirely unused and unseen—effectively unknown, just as future data is. Of course, we know it here, but for the sole reason of being able to evaluate the network's predictive performance on unseen data (otherwise we would have had to wait for the days and weeks to pass, and for the future data to slowly become available). Lastly, it is important to note that we are performing predictions one period ahead. The results for the test set are:

Network performance on test set (99 observations):

MSE = 18015.4

Hit Rate = 48.5% (48 out of 99)

The MSE is extremely high and the Hit Rate a disappointing 48.5%. Thus, the network predicted slightly less than half of the up/down movement directions. From a practical point of

view, these results are very unsatisfactory. But from a research point of view, they are interesting. However, this is just one, simple neural network model, and we must wait to draw any conclusions as to the predictability of the markets, the efficient market hypothesis, etc. For now, we turn to Fig. 6.11 which visualizes the performance on the test set.

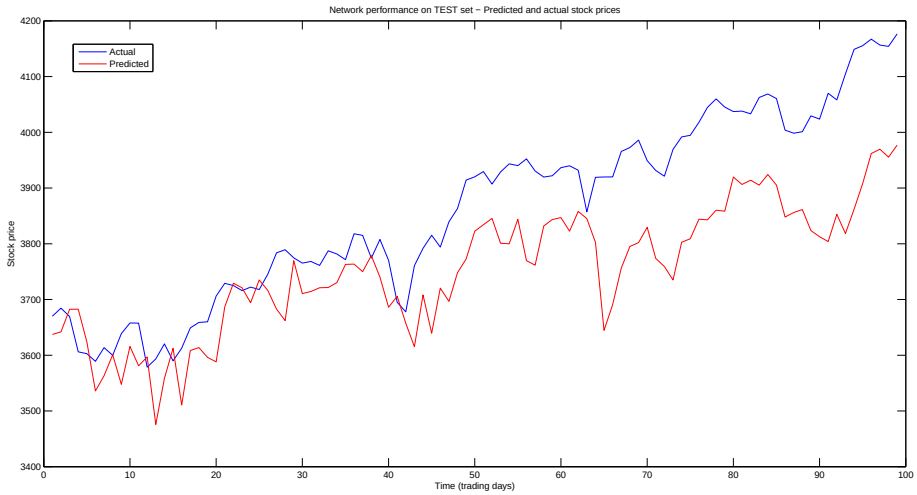

Figure 6.11: Neural network performance on TEST set. Vertical axis: Stock price. Horizontal axis: Time (trading days). Red curve: Predicted Prices. Blue curve: Actual prices.

Here we can better see the network's somewhat poor prediction ability on the unseen test data. It performs poorly in the beginning and even worse after about 50 trading days, where the predictions generally fall further and further below the actual data. However, regarding the overall shape and movement, the network predictions closely resemble the real data—with one important caveat, though; the network is "too late" by what appears to be one trading day.[14]

Evidently, the network shows mediocre performance on the training set and furthermore seems unable to generalize to the unseen test data where it lacks predictive ability. This could be an indication of either overfitting, a suboptimal network architecture with too few hidden or input neurons, or exogenous inputs and lagged responses that were simply not predictive enough for the purpose. Finally, it could also be caused by the noisy, erratic nature of daily data—something we investigate and discuss in e.g. Section 6.1.6. There, we argue, among other things, that data smoothed using moving averages is a much more plausible object of prediction.

In this case of daily data we also tried different training algorithms. Generally, the Levenberg-Marquardt is the quickest—so quick, even, that it often converges too fast and yields poor performance. Conjugate Gradient and Momentum with Adaptive Learning Rate are slower and do not significantly improve results. We find that Bayesian Regularization is generally the most stable and robust, albeit being somewhat slow. But none of the training algorithms give rise to significantly better results on the daily data. We also tried varying the number of hidden neurons, and found only slightly better performance when using additional neurons.

Thus, regardless of the training algorithm and number of hidden neurons, the neural network has a difficult time handling daily data. The main problem is that the noisiness and apparent

[14]This is actually a somewhat common phenomenon when applying neural networks for predicting financial time series, as seen in some of the research articles mentioned in Section 1.3.

randomness of the data cause the network to use today's price as the prediction for tomorrow. This is also very clear in Fig. 6.12 (especially the right plot), where we show the training and test performances of a neural network trained with Bayesian Regularization and 50 hidden neurons (the figure caption also includes the MSE and Hit Rates). This is good support for the claim that noisy, erratic daily data is implausible to use for prediction, and that we should pursue smoothed data or data with other frequencies instead.[15] Of course, the possibility still exists that our suboptimal results for daily data are caused by input features that lack predictive power.

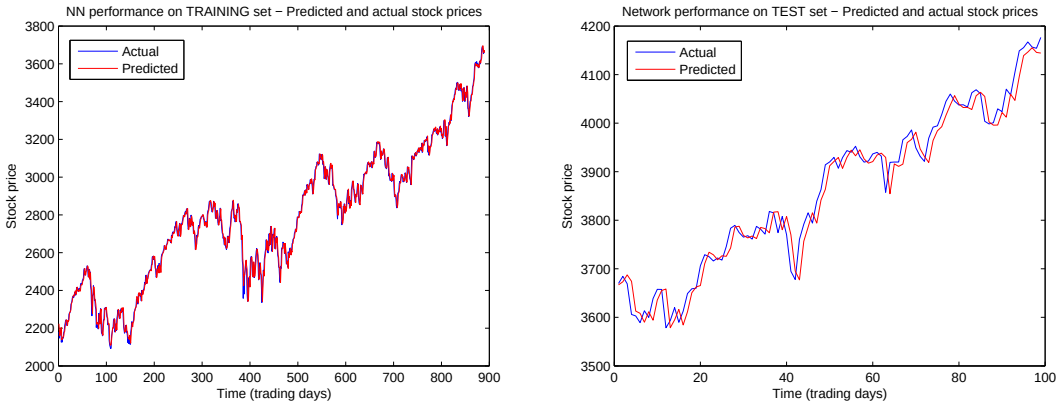

Figure 6.12: Training (left) and test (right) predictions for daily NASDAQ data of an ANN trained with Bayesian Regularization and 50 hidden neurons. Training performance: MSE = 997.02, Hit Rate = 52.36%. Test performance: MSE = 900.37, Hit Rate = 49.49%. The problem with using noisy daily data is clearly apparent from the fact that the predictions are shifted by one time step.

Finally, an aspect of neural networks that is sometimes an advantage, but in this case a shortcoming, is the random initialization of weights and biases. This can cause networks with identical architectures, inputs, etc. to end up in different local error minima and thus give different results. This is a major drawback of neural networks compared to the machine learning methods, Support Vector Machines and Random Forests, that we also consider in this study. However, the problem may be mended by e.g. training multiple networks on the same data and choosing the one with the best training performance (e.g. lowest MSE), or perhaps averaging the outputs of multiple networks.[16] We have implemented these features in our attached MATLAB programs, and it is something we will experiment with in the later experiments in Section 6.2. We have also implemented cross-validation to find the optimal number of hidden neurons.

6.1.6 Support Vector Regression Models

In the general case of SVM regression (SVR) there are two recurrent parameters, C and γ. C is the cost parameter, while γ appears in the kernel function. The specific cases of ϵ-SVR and ν-SVR introduce the respective extra parameters ϵ and ν. Moreover, as yet another feature to adjust and experiment with, there is the kernel function itself.

[15] We give a thorough analysis and discussion of this in Section 6.1.6.

[16] Training multiple networks and averaging their outputs would be a sort of *ensemble method* like Random Forests.

Investigating the Effect of γ (and C)

We begin by considering the effect of changing γ, and to a lesser extent C, all the while holding everything else constant. More specifically, we consider ϵ-SVR with fixed $\epsilon = 0.1$ and ν-SVR with fixed $\nu = 0.5$, keeping the kernel as a radial basis function[17]. The model attempts to predict tomorrow's price $y(t+1)$ using prices with 3 lags and a number of technical indicators also with 3 lags.[18] For the specific indicators, see Table 6.7 where we also show the results.

		ϵ-SVR				**ν-SVR**			
C	γ	MSE train	HR train	MSE test	HR test	MSE train	HR train	MSE test	HR test
1	0.001	1077.9	56.02 %	2283.5	48.49 %	835.93	60.41 %	2241.1	50.51 %
	0.01	1077.9	56.02 %	2283.5	48.49 %	835.93	60.41 %	2241.1	50.51 %
	0.1	1077.9	56.02 %	2283.5	48.49 %	835.93	60.41 %	2241.1	50.51 %
	1	1077.9	56.02 %	2283.5	48.49 %	835.93	60.41 %	2241.1	50.51 %
	5	1077.9	56.02 %	2283.5	48.49 %	835.93	60.41 %	2241.1	50.51 %
	10	1077.9	56.02 %	2283.5	48.49 %	835.93	60.41 %	2241.1	50.51 %
	100	1077.9	56.02 %	2283.5	48.49 %	835.93	60.41 %	2241.1	50.51 %
10	0.001	747.46	58.94 %	2286.1	52.53 %	244.05	80.20 %	2232.6	51.52 %
	0.01	747.46	58.94 %	2286.1	52.53 %	244.05	80.20 %	2232.6	51.52 %
	0.1	747.46	58.94 %	2286.1	52.53 %	244.05	80.20 %	2232.6	51.52 %
	1	747.46	58.94 %	2286.1	52.53 %	244.05	80.20 %	2232.6	51.52 %
	5	747.46	58.94 %	2286.1	52.53 %	244.05	80.20 %	2232.6	51.52 %
	10	747.46	58.94 %	2286.1	52.53 %	244.05	80.20 %	2232.6	51.52 %
	100	747.46	58.94 %	2286.1	52.53 %	244.05	80.20 %	2232.6	51.52 %
1000	0.001	732.52	59.51 %	2312.5	52.53 %	48.611	96.4 %	2797.7	51.52 %
	0.01	732.52	59.51 %	2312.5	52.53 %	48.611	96.4 %	2797.7	51.52 %
	0.1	732.52	59.51 %	2312.5	52.53 %	48.611	96.4 %	2797.7	51.52 %
	1	732.52	59.51 %	2312.5	52.53 %	48.611	96.4 %	2797.7	51.52 %
	5	732.52	59.51 %	2312.5	52.53 %	48.611	96.4 %	2797.7	51.52 %
	10	732.52	59.51 %	2312.5	52.53 %	48.611	96.4 %	2797.7	51.52 %
	100	732.52	59.51 %	2312.5	52.53 %	48.611	96.4 %	2797.7	51.52 %

Table 6.7: SVM performance results for varying γ (and, to a smaller extent, for varying C). All other parameters have been held constant; $\epsilon = 0.1$ for ϵ-SVM regression (leftmost columns) and $\nu = 0.5$ for ν-SVM regression (rightmost columns). The kernel used in all cases was the radial basis function (RBF) kernel. The particular SVM model attempts to predict the future stock price $y(t+1)$ using as predictors the past 3 prices as well as the technical indicators Momentum, RSI, Chaikin Oscillator, Price Rate of Change, and Volume Rate of Change, also with lags up to 3.

Evidently, varying γ seems to have no effect on the performance of either type of model; in Table 6.7, for each value of C considered, the performance measures are exactly the same for the different values of γ. For this reason, γ is not as important a parameter to adjust as some of the others, whereby we, in the analyses to come, will keep it fixed at its default value $\gamma = 1/n_i$, where n_i is the number of input features, or predictors.

As for the cost parameter C, which was also varied a bit, we found that large C significantly increased the computation times, especially for ν-SVR. Although this also resulted in improved training performance, the hit rates for the unseen test data were unchanged while the MSE actually increased. This remarkable training performance and disappointing test performance

[17]Through numerous tries we found that the RBF kernel was generally faster (computation time) and yielded better results than the alternatives of linear, polynomial, and sigmoid kernels.

[18]This particular combination of inputs was not chosen for any specific reason, but only to get started with the analysis.

observed for very large C is a strong sign of overfitting—a sign that fits well with the theory (Section 5.3) which exactly says that the increased penalty (from large C) might lead to overfitting (while small C might result in underfitting). Thus, due to its clear effect on the performance and relation to the overfitting problem, C is an important parameter to be monitored and adjusted. Let us therefore further analyse its effect on the performance.

Investigating the Effect of C (and ϵ and ν)

We now vary C, and to a lesser extent ϵ and ν for ϵ-SVR and ν-SVR, respectively. Other parameters are held constant, i.e. $\gamma = 1/n_i$, and the model and kernel function are the same as above. The results are shown in Table 6.8. We found that varying ϵ higher than 0.2 only worsened the results. In addition, reducing it below 0.01 gave even better training performances for increasing C, but also even worse test performances (i.e. even more overfitting).

	ϵ-SVR					**ν-SVR**				
C	ϵ	MSE train	HR train	MSE test	HR test	ν	MSE train	HR train	MSE test	HR test
0.01	0.01	37998	50.84 %	9380.7	49.49 %	0.1	110860	52.42 %	22655	51.52 %
0.1		5577	52.19 %	3670.3	52.53 %		14243	52.31 %	5252.3	53.54 %
0.5		1670.9	57.71 %	2677.3	47.47 %		2989.6	53.21 %	2793.8	49.49 %
1		837.75	61.75 %	2222.1	47.47 %		1806.9	53.21 %	2400.2	50.51 %
5		355.09	76.04 %	2197.7	49.49 %		861.81	58.04 %	2253.4	47.47 %
10		244.21	80.54 %	2231.5	52.53 %		655.6	61.19 %	2303.6	54.55 %
100		51.746	90.89 %	2822.1	46.46 %		173.5	78.85 %	2753.1	50.51 %
1000		12.834	93.25 %	3432.7	46.46 %		0.15147	99.78 %	3715.6	46.46 %
0.01	0.1	39974	49.04 %	9797.9	50.51 %	0.5	53122	48.71 %	12405	52.53 %
0.1		5958.2	51.52 %	3731.9	52.53 %		5833.9	51.74 %	3683.2	52.53 %
0.5		1865.3	53.21 %	2740.8	51.52 %		1682.1	55.01 %	2730.7	50.51 %
1		1077.9	56.02 %	2283.5	48.48 %		835.93	60.4 %	2241.1	50.51 %
5		774.16	58.83 %	2236.5	51.52 %		357.43	73.34 %	2183.9	51.52 %
10		747.46	58.94 %	2286.1	52.53 %		244.05	80.2 %	2232.6	51.52 %
100		732.52	59.51 %	2312.5	52.53 %		48.611	96.4 %	2797.7	51.52 %
1000		732.52	59.51 %	2312.5	52.53 %		0.14213	99.78 %	3718.5	45.45 %
0.01	0.2	59373	50.17 %	13571	54.55 %	0.9	38447	50.62 %	9461	49.49 %
0.1		13463	52.19 %	5120.5	53.54 %		5574.2	52.08 %	3664.9	52.53 %
0.5		7694.4	51.31 %	3632.2	48.48 %		1697.5	58.72 %	2673.3	45.45 %
1		7601.1	52.53 %	3544.6	50.51 %		855.54	64.23 %	2255.8	46.46 %
5		7598.3	52.53 %	3544.3	50.51 %		360.9	78.63 %	2206.7	52.53 %
10		7598.3	52.53 %	3544.3	50.51 %		251.79	84.48 %	2224.9	52.53 %
100		7598.3	52.53 %	3544.3	50.51 %		48.758	96.63 %	2799.4	51.52 %
1000		7598.3	52.53 %	3544.3	50.51 %		0.14214	99.78 %	3724.1	46.46 %

Table 6.8: SVM performance results for varying C and, to a lesser extent, for varying ϵ and ν for ϵ-SVR (leftmost columns) and ν-SVR (rightmost columns), respectively. All other parameters have been held constant; $\gamma = 1/n_i$, where n_i is the number of input features. The kernel was kept as the radial basis function (RBF). In terms of inputs/predictors, the particular SVM model is the same as that described in 6.7; i.e., the predictors are the past 3 prices and the technical indicators Momentum, RSI, Chaikin Oscillator, Price Rate of Change, and Volume Rate of Change, also with lags up to 3 (in total, giving $n_i = 18$).

It can be difficult to discern the results and make conclusions by looking at the raw data in Table 6.8. We therefore visualize the results in Fig. 6.13, where we have included additional values for ϵ and ν.

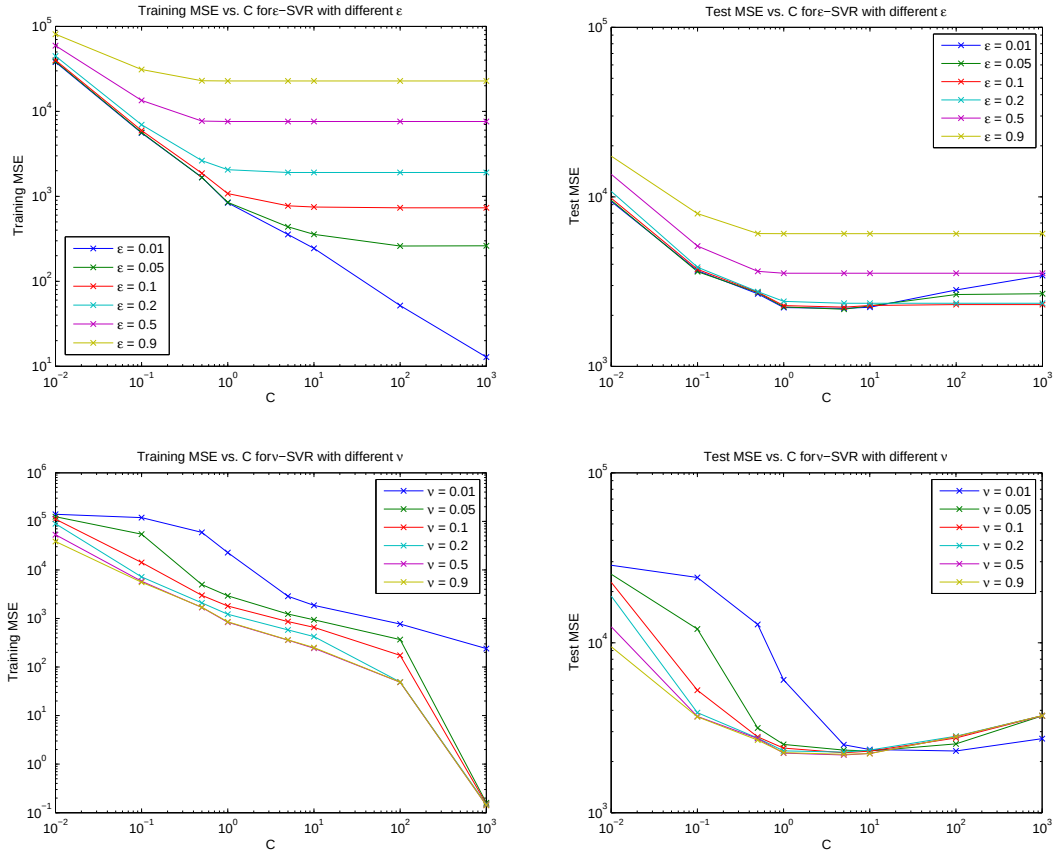

Figure 6.13: Effect on SVM regression MSE performance when varying C (and ϵ or ν). Top plots show ϵ-SVR training (left) and test (right) performances against C for different ϵ. Bottom plots show ν-SVR training (left) and test (right) performances against C for different ν.

What we see from Table 6.8, but even better so from Fig. 6.13, is the following:

- For ϵ-SVR, increasing C yields better *training* performance (lower MSE)—but only up to a point ($C \approx 1$), after which the performance generally remains the same. (For $\epsilon = 0.01$, it seems that the training MSE continues to decrease, but this will also even out eventually.) The *test* performance shows similar behaviour, evening out after $C \approx 1$. It is interesting to see that, for $\epsilon = 0.01$, the test performance began to increase again for larger C—a clear indication of overfitting. The overfitting occurring for large C can also be seen from the Hit Rate (accuracy) data in Table 6.8, where we sometimes observe exceptionally good training performances (hit rates upwards of 90%) but with corresponding test performances that are very, very bad. Finally, smaller (larger) ϵ generally seem to shift the MSE performance curves down (up), although the test results become very much alike when ϵ is below 0.2.
- For ν-SVR, increasing C also improves *training* performance (lower MSE)—and, as opposed to ϵ-SVR, it does not seem to even out; the MSE just decreases further. The *test* performance also improves for increasing C, at least initially. It flattens out between $C \approx 1$ and $C \approx 10$, after which the MSE increases again. The fact that the test MSE increases for

very large C, all the while the training MSE continues to decrease, is a clear sign of overfitting. As explained in the previous paragraph, this overfitting can also be seen from the Hit Rate data in Table 6.8. The performance is almost unchanged for different ν (except for $\nu = 0.01$ (blue), which seems to shift the test curve to the right).

Thus, regardless of ϵ or ν, varying C shows the trade-off between a model with mediocre training and test performances (underfitting) and one with great training performance but poor test performance (overfitting). This agrees perfectly with the theory, from which we know that larger C increases complexity and thus the risk of overfitting. For ϵ-SVR, we also see that low ϵ (here 0.01) makes the overfitting even more pronounced, which also agrees with ϵ-SVR theory [29].

Finally, we have implemented cross-validation in our MATLAB programs for finding the optimal SVM parameters.

Qualitative Performance Assessment

Above, we have only considered the numerical quantities for performance, i.e. MSE and Hit Rate. It would be very interesting to inspect more closely the actual predictions of a particular model, and get a more visual, qualitative feel for the performance. This is done in Fig. 6.14 where we plot the training and test predictions for the (arbitrarily chosen) ν-SVR model with a radial basis function kernel, $C = 10$, $\gamma = 1/n_i = 1/12 = 0.0833$, $\nu = 0.9$, and Price (also the response variable), Momentum, RSI, and Price Rate of Change with lags up to 3 as inputs/predictors (giving $n_i = 4 * 3 = 12$). Besides showing the plots, our MATLAB program also outputs the performance results shown in Table 6.9.[19]

	MSE	Hit Rate
Training	578.18	64.12 % (570 out of 889)
Test	829.94	61.62 % (61 out of 99)

Table 6.9: Performance results for the SVM model described in the text above. The corresponding predictions are shown in Fig. 6.14.

[19]Note that the model employed here achieves better overall training and test performances than the models in e.g. Table 6.8. The reason is that these previous models employed the Volume Rate of Change indicator which suffered under an extreme outlier. A single observation was thus the cause for the large magnitude of the test errors. Not using the Volume Rate of Change as an input, this new model does not suffer under the extreme outlier and the test MSE value is significantly lower than the values in Table 6.7 and 6.8.

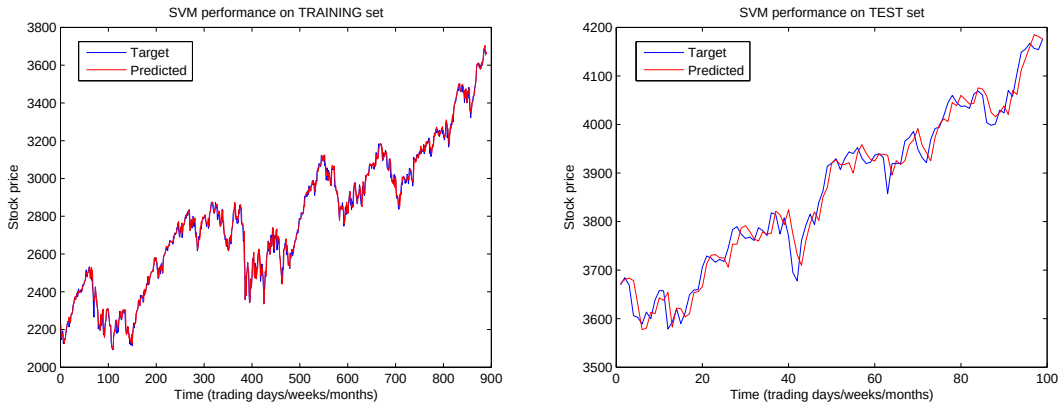

Figure 6.14: Training (left) and test (right) predictions (red curves) for the SVM model described in the text above. The actual targets—NASDAQ daily prices—are included (blue curves) for comparison and performance assessment. The horizontal axis is time in trading days, and the vertical axis is the daily closing price.

First off, the parameters of the model seem to give a good balance in terms of the underfitting-overfitting problem discussed earlier; from Table 6.9 we see that the training and test MSEs and Hit Rates are very close, indicating that overfitting has not occurred. Indeed, the Hit Rates of 64% and 62% obtained here are much better than those shown in Tables 6.7 and 6.8 (to be fair, though, these previous results only served to analyse the effect of varying parameters).

Turning to the left plot in Fig. 6.14, we see that the predictions for the training data practically lie atop the actual targets (as they should, since the model was trained on this data). At first glance, the predictions for the unseen, unused test data in the right plot also seem to very good, following the actual data closely. However, there is a crucial deviation; namely that the predictions appear lagged, or shifted, compared to the targets. And upon further scrutiny, one notices that they are lagged by exactly *one* period—in this case one day. Essentially, apart from a few exceptions, this means that the model's prediction of $y(t+1)$ is basically just $y(t)$, or, put another way, the model predicts that tomorrow's price will just be today's price. We saw the same for neural networks in the previous section.

Had it not been for this lag, or shift, the predictions would have been exceptionally good (too good perhaps). Of course, wanting one's predictions to be good, the first reaction to this phenomenon is that something is wrong with the code. However, we have thoroughly perused our programs several times over, finding no bugs that may lead to this behaviour.

Example with a Smooth and Noisy Sine Wave

To be entirely sure that nothing is wrong with our SVM programs and implementations, we have tried using some other, perhaps simpler, data. More specifically, we will try to predict the *sine* function $f(x) = \sin(x)$. In the attached MATLAB program we generate the target data for $x \in [0, 100]$ in steps of $\Delta x = 0.1$. To predict $sin(x)$ we will employ a ν-SVR model with a radial basis function kernel, $C = 1$, $\gamma = 1/n_i = 1/5 = 0.2$, $\nu = 0.5$, and which uses as inputs/predictors the past 5 values, i.e. $\sin(x - i\Delta x)$ with $i = 1, \ldots, 5$ (giving $n_i = 5$).[20] The model is trained on

[20]This scenario corresponds to predicting next period's (e.g. day's) stock price using the prices for the past 5 periods (e.g. days).

the first 90% of the data and tested on the remaining (unseen) 10%. The results of predicting this smooth sine wave are shown in Fig. 6.15.

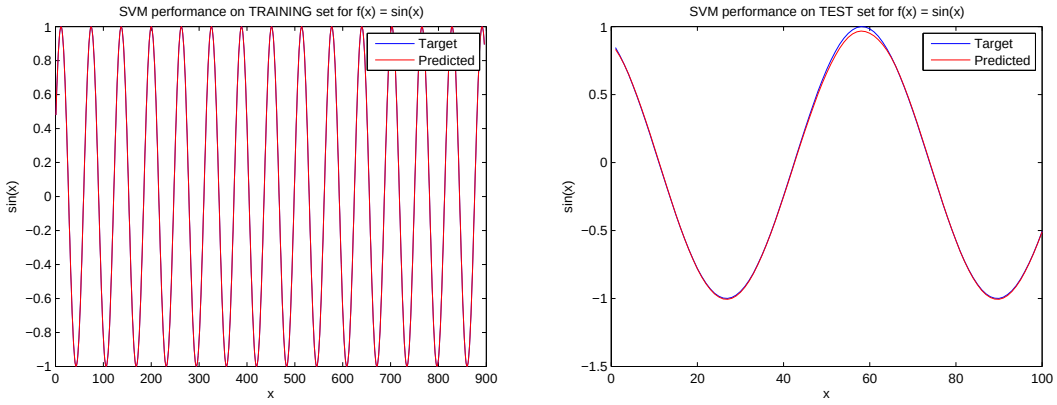

Figure 6.15: SVM model performance for predicting the smooth sine wave $f(x) = \sin(x)$ for $x \in [0, 100]$ in steps of $\Delta x = 0.1$. Left: performance on training set. Right: performance on unseen, unused test set.

Our MATLAB program also outputs the MSE (and Hit Rates, but these are not so informative for the smooth sine wave). The MSEs are 0.000000 and 0.000158 for the training and test set, respectively. These errors are very low, indicating perfect performance on the training set and near-perfect performance on the test set. This is also what we see in Fig. 6.15; in the left plot (training set), the predictions lie atop the targets; in the right plot (test set), we observe only a minuscule deviation near the top of the function. In conclusion, the model perfectly predicts the smooth sine wave, and there seems to be no bugs or errors in our code which could cause a shift, or lag, of the predictions as we saw earlier in Fig. 6.14.

For the sake of curiosity, let us try to add random noise to the sine wave. The model parameters and inputs are the same as before, and we train it on 90% of the data and test it on the remaining 10%. The training and test MSEs are 0.009888 and 0.011724, respectively, and the Hit Rates are 55% and 51% (they were both 100% for the smooth sine wave). The predictions and actual targets are shown in Fig. 6.16.

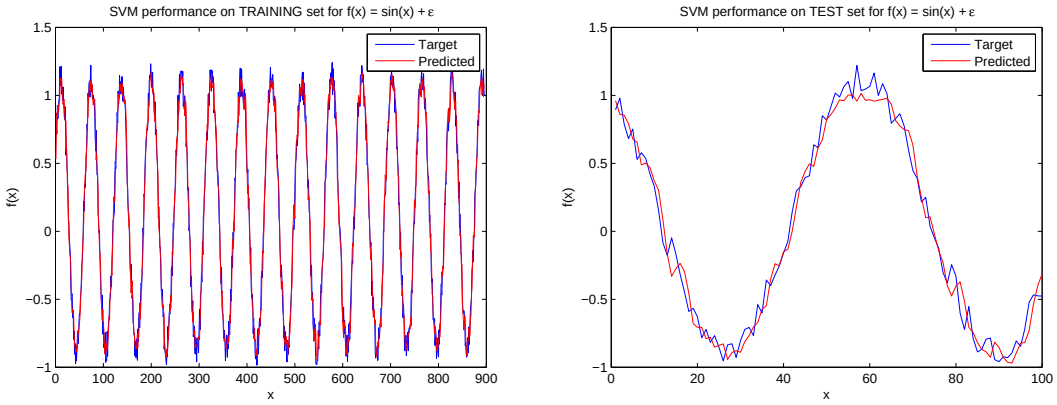

Figure 6.16: SVM model performance for predicting the noisy sine wave $f(x) = \sin(x) + \epsilon$ for $x \in [0, 100]$ in steps of $\Delta x = 0.1$, and where ϵ is random noise. Left: performance on training set. Right: performance on unseen, unused test set.

From the quantitative performance measures we see that the performance, albeit good, is not as perfect as for the smooth sine wave. This is supported qualitatively by Fig. 6.16. Especially interesting is the right plot showing the performance on the unseen test data. Before, for the smooth sine wave, we saw near-perfect agreement between the predictions and targets. Here, however, with the inclusion of noise, there are clear deviations throughout. What is even more interesting, though, is the nature of these deviations; upon close inspection, one can see that the predictions (red curve) are in many cases shifted to the right. In particular, for many of the noisy "spikes" in the target data[21], the model predicts a similar spike—only it does so too late (e.g. at $x \approx 15, 80$).

Thus, most often, the model just retains the direction, or momentum so to speak, of the observed data; when everything is headed the same way, the predictions reflect this. But then, when a sudden move in the opposite direction appears (a noisy spike), the model just keeps the original direction. Only after the next period, when the spike has happened and the model knows this, the model uses this information for its prediction, resulting in a predicted spike occurring one period after the actual, observed spike. Basically, the model uses for its prediction the most recent difference in response value $f(x)$. This is exactly what we discussed earlier and saw in Fig. 6.14 for the daily NASDAQ data! But this was not the case for the smooth sine wave, i.e. in the absence of noise. So, our sine wave examples indicate that the inclusion of random noise in the data cause most of the predictions of next period's value to just be the current period's value (plus some overall trend[22]). Thus, since we saw the same phenomenon of shifted predictions for the daily NASDAQ data, a tentative conclusion is that daily stock price fluctuations are, to a certain extent, random (at least for the NASDAQ index). This is very interesting because, for a Random Walk, it holds that

$$E[y(t+1)] = y(t), \tag{6.3}$$

which states that the expected future value is just the current value.[23] Translated to our

[21] These noisy spikes could correspond to the noisy up and down movements of e.g. daily stock prices.
[22] The trend is negated when using differences that make the data stationary.
[23] For more details, see our Random Walk benchmark model in Section 2.1.1.

case, this says that the expected value of tomorrow's price is just today's price[24]—which is exactly what our predictions have indicated so far! Thus, it may be argued that the daily NASDAQ prices follow, to some extent, a Random Walk, which would lend empirical support to the Efficient Market Hypothesis discussed in Section 2.1.1. Another possibility is that the models we have used to predict the daily NASDAQ prices are simply not advanced enough. This lack of predictive power may or may not then be found in the variables used as input features.

In any case, though, daily data may not be the proper object of prediction due to its erratic nature. Instead, it might be more interesting to consider e.g. a moving average which smooths the data. Predicting a moving average instead of the raw price itself is still highly relevant, as it provides an indication of the trend. In place of daily data, it would also be interesting to consider weekly or monthly data, which may not exhibit the same degree of fluctuation.

The motivation for smoothing data is further supported by other research; indeed, [10] argues that forecasting trends is more plausible and suggests smoothing both input and output data using simple or exponential moving averages. This recommendation is based on empirical work showing that price changes around a trend are somewhat random [49], thus making prediction of such price changes very difficult. Our tentative conclusion above (of somewhat random daily fluctuations) agrees nicely with this. In many of the later experiments we will therefore use smoothed data.

6.1.7 Support Vector Classification Models

Above, we considered the problem of predicting a continuous response—the stock price—which is inherently a regression problem. This proved a difficult task, especially for the highly fluctuating daily data. However, the task may be simplified if we only want to predict the movement *direction* (up or down), i.e. whether the stock price will increase or decrease. Of course, since we replace the continuous response with categorical/binary data (letting 1 denote an up movement, and 0 a down movement), this simplification removes some information in the data. It also leaves us with the Hit Rate, or classification accuracy, as the only performance measure. In any case, though, it might help to reduce the complexity of the prediction problem and possibly improve the results. We will investigate this in the following, after briefly going through the effects of varying the parameters.

Investigating the Effect of Varying C in C-SVC and ν in ν-SVC

The only parameters in C-SVC are C and γ. In ν-SVC, the parameter ν replaces C. We will briefly analyse the effect of changing these two main parameters. In both cases we will use a radial basis function (RBF) kernel[25] To predict the daily up/down movement direction of the NASDAQ index, the model employs as predictors the past 3 movement directions as well as the technical indicators Momentum, RSI, Chaikin Oscillator, Price ROC, and Volume ROC, all with lags of 3 (giving $n_i = 18$). Thus, it is the same model we considered in the previous SVM regression section. For the sake of simplicity, we keep γ constant at $\gamma = 1/n_i = 1/18 = 0.56$. The Hit Rate (classification accuracy) performances are shown in Fig. 6.17.

[24] More specifically, we should use *returns* (i.e. price differences) instead of prices.

[25] After numerous analyses, we found that the RBF kernel was both computationally faster and generally yielded better, more stable results than the alternatives (i.e. the linear, polynomial, and sigmoid kernels).

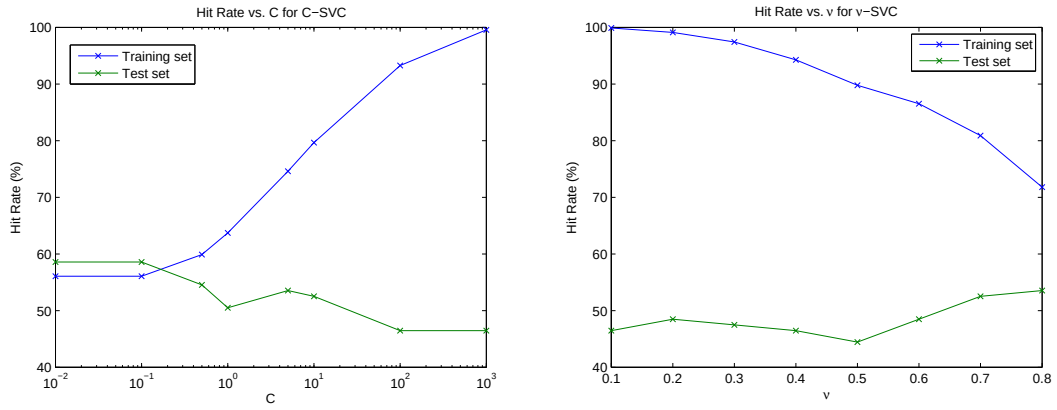

Figure 6.17: Effect on training and test performance (classification accuracy) when varying C for C-SVC (left) and ν for ν-SVC (right).

The left plot in Fig. 6.17 shows how the training performance for C-SVC improves (larger Hit Rates are better) as C is increased. However, the performance on the unseen test data gets worse. The model thus appears to memorize the training patterns and, as a result, loses its generalization ability—a clear sign of overfitting. As discussed in the above, this fits excellently with the theory.

In the case of ν-SVC, the right plot in Fig. 6.17 indicates that small ν may lead to overfitting (perfect training Hit Rates of 100%, but poor test Hit Rates). However, increasing ν balances things out, with lower (but still good) training performances and improved test performances.

Classification vs. Regression

In our MATLAB programs we have implemented cross-validation for finding the optimal parameters. The best SVM performances we were able to obtain (for the classification problem) were test set Hit Rates of $\sim 60\%$. Since this is not better than the Hit Rate results obtained for regression (in e.g. Table 6.9), and since classification takes out a good bit of information and leaves us with this single performance measure, we will focus our energy on the regression problem.[26]

6.1.8 Random Forests Models

For Random Forests there are much fewer adjustable parameters and settings than for Support Vector Machines and Neural Networks. In fact, there are really just three; the number of trees N in the forest, the leaf size, and the amount of randomly chosen input features m to consider when splitting a node. A larger number of trees is generally better, but also increases computation time, and the performance gains quickly become unnoticeable. Default values for the leaf size generally work quite well. Thus, the parameter m is the only one to which Random Forests are somewhat sensitive. We explore these things, and more, in the following analysis.

[26] Note that Random Forests and Neural Networks can likewise be used for classification, which we have also implemented in our programs. But again, the results from the classification problems are not better than those for the regression problems, so we stick with regression since this enables us to compute additional performance measures and make nice, visual plots of the predictions.

As in all of the above, we consider the daily NASDAQ data in the period Jan 1 2010 to Jan 1 2014. For predictors we arbitrarily choose 3 lags of the response data and 3 lags of each of the Momentum, RSI, and MACD technical indicators, giving $n_i = 12$ input features. 90% of the data will be used for training, parameter optimization, and feature importance evaluation. The remaining 10% of the data are kept entirely unseen, as if it was future, unknown stock prices.

Varying the Number of Randomly Chosen Features m

We begin by training Random Forests with $N = 100$ trees for different values of the number of randomly chosen input features m. The measure by which we will assess the performance is the out-of-bag MSE, i.e. the error on the observations which were not used to grow the tree. Giving, during the training phase, an indication of the generalization ability, this out-of-bag error renders cross-validation obsolete and is one of the best features of Random Forests. The results are shown in Fig. 6.18.

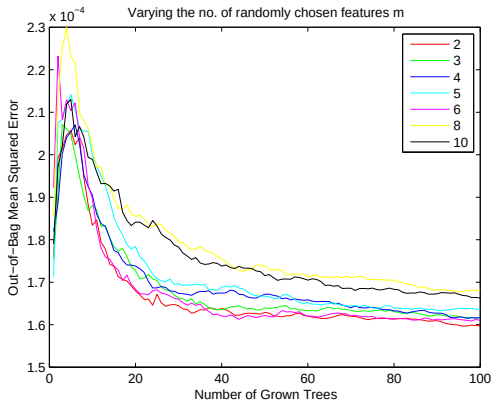

Figure 6.18: Out-of-bag mean-squared error (MSE) vs. number of trees for Random Forests with different $m = 2, 3, 4, 5, 6, 8, 10, 12$ (number of randomly chosen input features).

Fig. 6.18 plots the out-of-bag MSE against the number of trees for different m. Sharing the same general shape, the curves tell us that the error decreases as more trees are grown, in accord with the theory that larger forests are generally better. However, as we also mentioned above, the performance gains decrease, which is exactly what see from the flattening of the curves. This general relation between error and forest size appears to hold true irrespective of m. But, the magnitude of the error seems to depend on m, which is important when searching for the optimal parameters. Although the results vary due to the random nature of the method, the outcome of numerous test runs indicates that m between 2 and 6 give the best performance (for this particular case with $n_i = 12$), which is also what we see in Fig. 6.18 where $m = 2, 3, 4, 6$ yield almost equally low errors. In fact, these values agree nicely with the default value $m = n_i/3 = 12/3 = 4$.

Varying the Leaf Size

We now vary the leaf size which determines the minimum number of observations per tree leaf. Again, we are training forests with $N = 100$ trees. The out-of-bag performance for different leaf sizes is shown in Fig. 6.19.

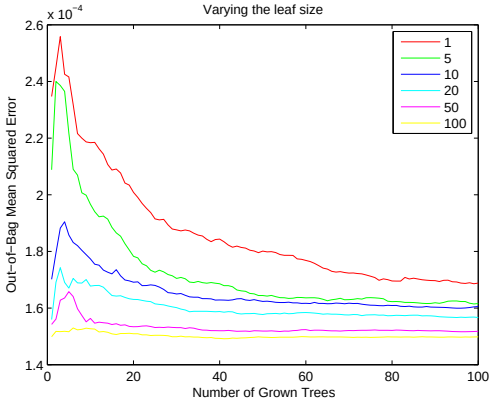

Figure 6.19: Out-of-bag mean-squared error (MSE) vs. number of trees for Random Forests with different leaf sizes.

Like the previous figure, Fig. 6.18 also plots the out-of-bag MSE against the number of trees, but this time for varying leaf sizes. The overall shape of the curves is the same; the out-of-bag error first decreases and then flattens out as the number of trees is increased, agreeing perfectly with theory. Moreover, this seems to hold regardless of leaf size. As for the optimal leaf size, the different values all yield rather similar errors (for sufficiently many trees grown). Still, it seems that larger leaf sizes produce lower out-of-bag errors, but with diminishing improvements as we approach 100 (as can be seen from the fact that the purple and yellow curves are very close despite the large jump in magnitude from 50 to 100).

Estimating Feature Importance

Another extremely appealing aspect of Random Forests is its ability to estimate the importance of input features. That is, the method can actually be used—in quite a simple manner—to rate the predictive power of e.g. technical indicators and find the most optimal combination of input variables. Normally, for other methods, this can be a very involved task requiring computation-intensive feature selection algorithms or tiresome manual variation of the inputs. But Random Forests provides a quick, straightforward approach.[27]

We grow a forest comprising $N = 100$ trees. Fig. 6.20 plots the increase in MSE due to permuting the out-of-bag observations across each input variable; the larger this value, the more important the variable [35].

[27] We refer to e.g. [35] for details on using Random Forests to measure feature importance.

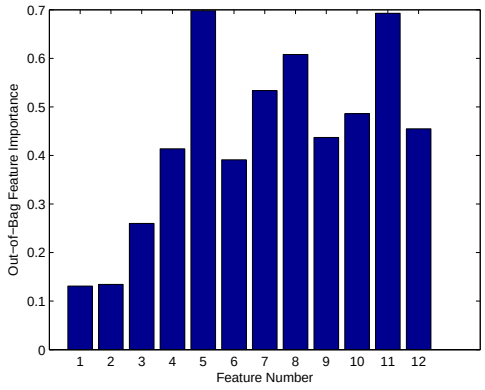

Figure 6.20: Analysis of feature importance. Our inputs are formatted such that the feature numbers are as follows: 1-3 denote the lagged response data; 4, 7, and 10 are the lagged Momentum; 5, 8, and 11 are the lagged RSI; and 6, 9, 12 the lagged MACD.

The most prominent features in Fig. 6.20 are 4 through 12—which is, in fact, all the three lags of each of the three technical indicators used here (Momentum, RSI, and MACD). If we impose an arbitrary cutoff at 0.6, then the most important features are 5, 8, and 11—exactly all three lags of the RSI.

Using a reduced model with just these three features, it is interesting to see if we can obtain a predictive power similar to that of the full model with all features. We therefore grow a forest with $N = 100$ trees using only features 5, 8, and 11 (RSI with lags of 1-3). The results are shown in Fig. 6.21.

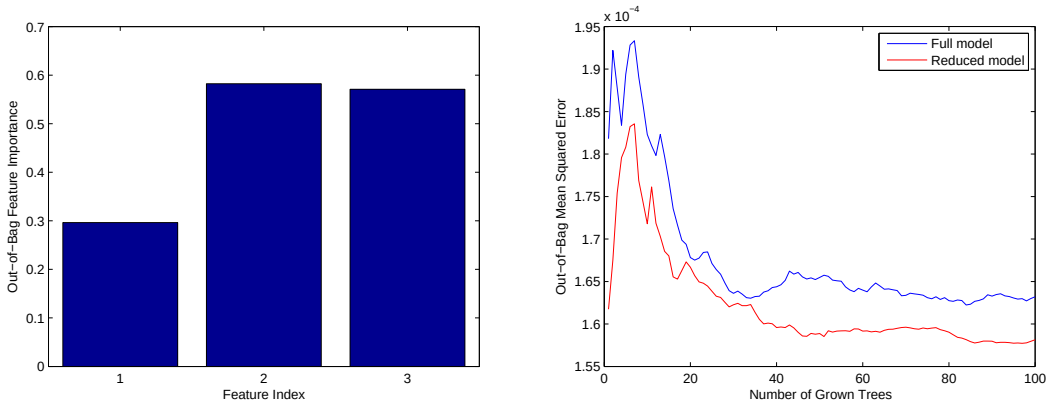

Figure 6.21: Left: Feature importance for reduced model using only features 5, 8, and 11 (lagged RSI). Right: Comparison of out-of-bag MSE for the full model (all 12 features) and the reduced model.

The left plot in Fig. 6.21 shows the feature importance for the reduced model; the features are ranked somewhat similarly to each other as in the full model, although most power now seems to be ascribed to lags 2 and 3 of the RSI. The right plot compares the out-of-bag error for

the two models. Evidently, the reduced model actually gives slightly lower MSE than the full model.

Plotting and Evaluating the Predictions

We now plot the predictions and compute performance measures for both the training set and the unseen test set. Note that this is done after transforming the logreturns (used to build the models) back to stock prices (hence the large difference in magnitude of the MSE in e.g. Fig. 6.21 and Table 6.10).

The MSE, MAPE, and Hit Rate performance measures for the full and reduced model are shown in Table 6.10. The predictions are plotted along with the actual prices in Fig. 6.22.

	TRAINING performance			TEST performance		
Model	**MSE**	**MAPE**	**HR**	**MSE**	**MAPE**	**HR**
Full model	425.84	0.53%	87.47%	815.23	0.57%	57.14%
Reduced model	567.23	0.63%	79.38%	833.25	0.58%	59.18%

Table 6.10: Training and test performances for the full and reduced Random Forest models.

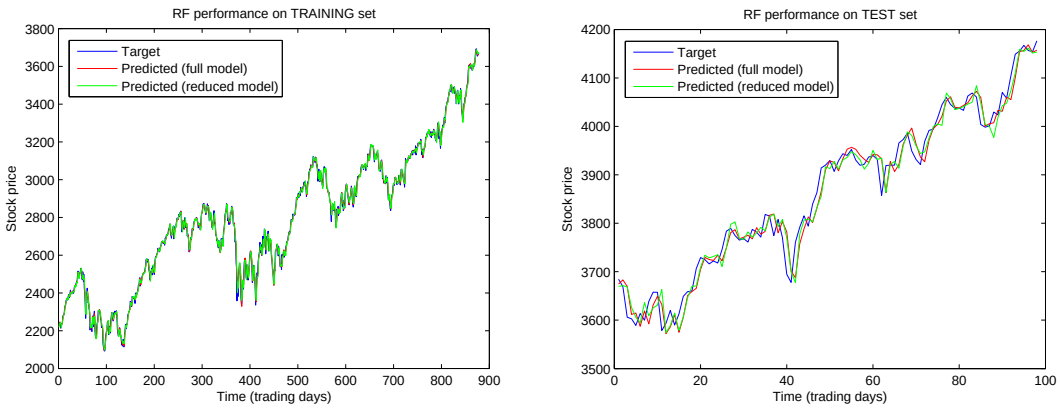

Figure 6.22: Predictions and actual values for training data (left) and test data (right). Full model: 3 lags of the response, Momentum, RSI, and MACD. Reduced model: 3 lags of the RSI.

In Table 6.10 we see that the full model performs slightly better than the reduced one, with lower MSE and MAPE and a larger Hit Rate. This can be difficult to visually assess from the left plot in Fig. 6.22 due to the large time period, but deviations are still visible. As for the test set, the performance measures in Table 6.10 are very similar; although having slightly higher MSE and almost identical MAPE, the reduced model actually achieves a slightly higher Hit Rate. But these differences are so small as to make no matter. Thus, whereas the full model showed better training performance, the two models perform almost identically on the unseen test data. Of course, the previous section showed that the reduced model had slightly lower (though not by much) out-of-bag error than the full model. Still, however, it is incredible that the same test performance can be achieved by using just three lags of the RSI (reduced model) compared to using three lags of the response, Momentum, RSI, and MACD.

Finally, the right plot in Fig. 6.22 shows an interesting feature; the predictions of the daily data appear shifted by one time step. This seems a common occurrence when attempting to

predict noisy, erratic daily data. Indeed, we saw it earlier in e.g. Section 6.1.6 where we also gave a thorough discussion of the phenomenon and possible reasons for it.

6.2 Experiments

In the previous sections we analysed the NASDAQ Index and performed several initial experiments that investigated, among other things, the effects of varying the model parameters. We were able to draw several conclusions in this regard, all of which were generally consistent with theory and previous research. However, we were also able to draw some tentative conclusions in another regard. More specifically, we found, in accord with other research, that it may be implausible to forecast the daily stock price fluctuations themselves. Rather, it may be more plausible to consider e.g. weekly data or, perhaps better yet, smoothed data. For example, a 5-day moving average of the daily stock price provides a measure of the average price over the past trading week—an interesting and relevant thing to try to predict.

Whereas the experiments in the previous section were largely method-specific, the numerous experiments in this section focus on comparing the performances of different models in a variety of scenarios. We dig deeper into data type and amount, parameter and input combinations, etc. In addition, we will also perform some predictions for other assets than just the NASDAQ Index.

Experiment 1

In this experiment we compare the performances of SVMs, ANNs, and RFs for the following case of data, input combination, etc.:

Data and time period: Daily prices of the NASDAQ Index from 01 Jan 2010 - 01 Jan 2014.

Pre-processing and transformation: In respective order: 5-period EMA of **daily** prices; logarithmic returns; normalization to zero mean, unit variance.

Inputs: 2 lags of the response data and 2 lags of MOM, RSI, and MACD. In total, $n_i = 8$.[28]

Data Split: The total 1006 observations[29] are split into training/test sets comprising 90%/10% of the data.

Model(s): ν-SVR with RBF kernel, $C = 1$, $\gamma = 1/n_i = 1/8$, $\nu = 0.9$. Neural Network (NN) trained with Levenberg-Marquardt algorithm and $m = 4$ hidden neurons. Random Forest (RF) with $n = 100$ trees and a leaf size of 20.

The model parameters were optimized using cross validation (for NN and SVM) and the out-of-bag error (for RF), which we have implemented in the attached MATLAB program. This ensures a fair comparison where each respective method is properly represented, thus avoiding matching e.g. a good Support Vector Machine to a bad Neural Network.

In addition to the three SVM, NN, and RF models, we have included the Random Walk (RW) and Random Guess (RG) models, which serve as benchmarks. The RW model uses the current period's price as the prediction for the next period's price. The RG model basically flips a coin to guess whether the price will increase or decrease (hence, its only performance measure is the Hit Rate). On average, the RG model's Hit Rate will, of course, be 50%, but our implementation

[28] MOM = Momentum, RSI = Relative Strength Index, and MACD = Moving Average Convergence Divergence.

[29] The effective number of observations is slightly smaller (976) since some inputs, e.g. technical indicators and lagged data, require a certain amount of data points to be initiated. Synchronization of targets and inputs thus results in the first few observations being cut away.

in the attached MATLAB program simulates a real coin flip for finite sample sizes, whereby the actual accuracy will deviate around this average.

For performance measures we consider the mean-squared error (MSE), the mean absolute percentage error (MAPE)[30], and the hit rate (HR). The results are shown in Table 6.11. The predictions are plotted in Figure 6.23.

	TRAINING performance			TEST performance		
Model	**MSE**	**MAPE**	**HR**	**MSE**	**MAPE**	**HR**
SVM	105.65	0.264%	77.45%	92.69	0.191%	77.55%
NN	133.21	0.324%	74.83%	106.91	0.208%	76.53%
RF	92.73	0.257%	78.70%	108.10	0.208%	74.49%
RW	191.64	0.404%	73.55%	165.01	0.272%	75.51%
RG	N/A	N/A	48.80%	N/A	N/A	48.98%

Table 6.11: Performance results for Experiment 1. The different models are: SVM = Support Vector Machine; NN = Neural Network; RF = Random Forest; RW = Random Walk; RG = Random Guess. The training set is used for model building and comprises 878 observations. The test set is entirely "unseen" and comprises 98 observations. Lower MSE and MAPE values are better, while higher HR values are better.

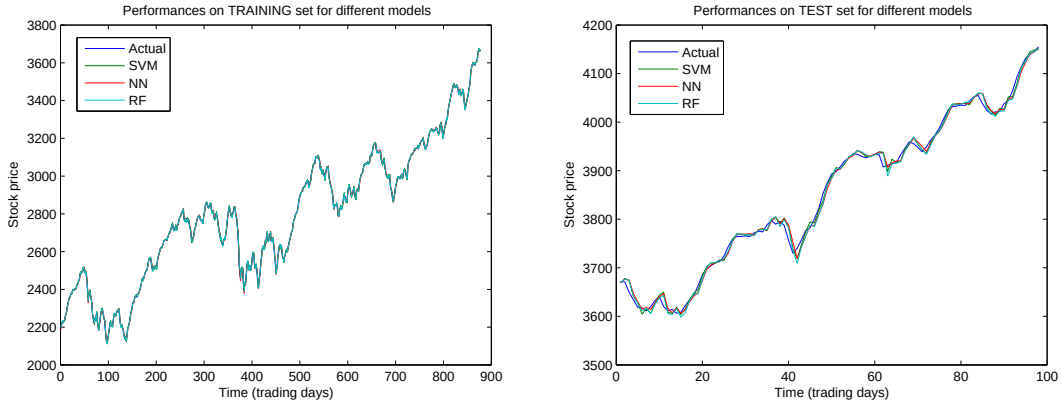

Figure 6.23: Prediction results for the different models in Experiment 1. Left: training set. Right: unseen test set.

Table 6.11 shows that, for each of the three machine learning methods, the training and test performances are rather similar. This indicates that overfitting has not taken place—and that our cross-validation and out-of-bag validation has worked. RF has slightly better training performance (lower MSE and MAPE, higher Hit Rate) than NN and SVM, but also slightly worse test performance. SVM performs best on the test set on all three performance measures. But the differences are marginal, as can also be seen from Fig. 6.23 where the predictions from the three models almost lie atop each other.

These results are interesting in themselves when assessing which machine learning method is best (in which case it is difficult to announce a clear winner). However, when it comes to real-world application, we must compare them to the benchmarks. RG, the simple coin-flip model,

[30]In order to take into account the possible changes in magnitude of the stock price, we have included the MAPE to get a scale-invariant error measure.

only predicts around 50% (as expected) of the up/down price movements. All three models—SVM, NN, and RF—are clearly much better than this. As for RW, the "tomorrow = today" model, this shows training and test Hit Rates roughly on par (but mostly lower) with our three models. However, in terms of MSE and MAPE, RW performs significantly worse on both the training and test set. Thus, although our SVM, NN, and RF models predict roughly the same percentage of up/down movements as the RW benchmark, they do so in a much better and more stable way—with lower deviation from the actual prices.

In conclusion, the SVM, NN, and RF models show exceptionally good performances on the training set—but, more importantly, also on the unseen test set. This can also be seen in Fig. 6.23 where the predictions lie very close the actual data. They each succeed in correctly predicting about 75% of the up/down movements. This is a very good Hit Rate, but it does slightly wane when compared to the comparable Hit Rate obtained by the Random Walk benchmark[31]. However, in terms of the MSE and MAPE—both of which take into account the magnitude of the predictions (as opposed to the Hit Rate measure)—our three models perform much better than the RW benchmark.

Experiment 2

In this experiment we investigate the effect of varying the number of lags for both the response variable and the exogenous inputs. This will be done for the particular exogenous input combination comprising the MOM, RSI, and MACD technical indicators. The experiment settings can be summarized as follows:

Data and time period: Daily prices of the NASDAQ Index from 01 Jan 2010 - 01 Jan 2014.

Pre-processing and transformation: 5-period EMA of **daily** prices; logarithmic returns; normalization to zero mean, unit variance.

Inputs: Lagged response data and MOM, RSI, and MACD. The number of lags is subject of variation; the response lags and exogenous input lags are both varied from 0 to 5, resulting in a total of 35 different lag combinations.[32]

Data Split: The effective 976 observations are split into training/test sets comprising 90%/10% of the data (i.e. 878/98 observations).

Model(s): ν-SVR with RBF kernel, $C = 1$, $\gamma = 1/n_i$, $\nu = 0.9$. Neural Network (NN) trained with Levenberg-Marquardt algorithm and a number of hidden neurons optimized using 10-fold cross-validation. Random Forest (RF) with $n = 100$ trees and a leaf size optimized using the out-of-bag error.

As mentioned, the NN and RF parameters have been optimized using cross-validation and the out-of-bag error, respectively. This is in important because the varying number of lags results in different numbers of inputs. This is especially crucial for neural networks, where the choice of hidden neurons depends on the number of inputs so as to obtain the best balance between underfitting and overfitting. Now, we have also implemented cross-validation for SVMs in the attached MATLAB programs, but this was not used here since we found that the C and

[31] It is important to add that the good performance (particularly the Hit Rate) of the RW benchmark model is mainly due to the general upward trend observed in the NASDAQ Index in the time period considered here (as can also be seen in Fig. 6.23). Of course, this doesn't lend less importance to the RW model results, because such trends are indeed a common phenomenon on the financial markets.

[32] The combination with zero lags of both the response lags and the exogenous inputs is obviously not possible, since no input data is available in this case.

ν parameters yielded the best performances irrespective of the number of lags. However, the varying number of lags—and hence the varying number of inputs n_i—is indeed accounted for in the SVM model via the γ parameter that depends n_i.

With 35 different lag combinations—and six error measures for each of three different models—the resulting amount of raw data is much too large to present in tables. Not only is visualization in figures much more appropriate, it also showcases the potential relationships in a way that is much easier to grasp. However, there is still an abundance of figures, so we have chosen a sample of them to show here.[33] The selected sample reflects that we are most interested in the MSE and MAPE performances (because these contain more information) on the unseen test data (because this gives an indication of true prediction ability). Moreover, we have included slightly more SVM plots than NN and RF plots, the reason being that the latter two methods include random initialization of weights (for NN) and sampling of input features (for RF). Even though we have implemented measures to reduce the effects of this randomization—by e.g. re-training numerous neural networks (to re-initialize the weights) and choosing the one with lowest error on some unseen part of the training data—the differences in results across different lag combinations may still partly be ascribed to these random effects. SVM does not include any randomization and is thus more consistent and appropriate for this experiment. The performance results are shown in Fig. 6.24.

[33]Note that all figures are available in the attached `/Results/Experiment-2` folder and can furthermore be re-created in the attached MATLAB programs.

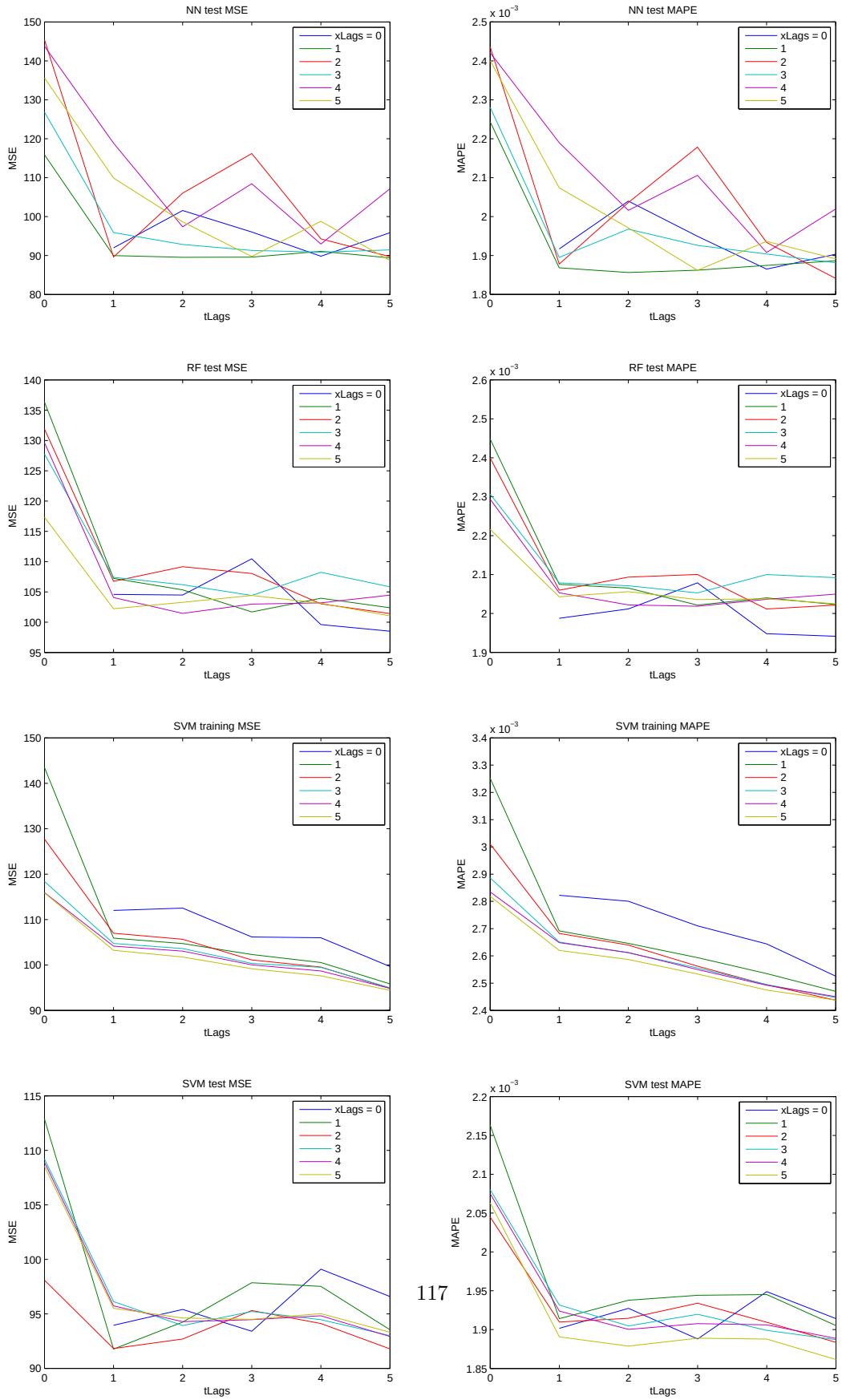

Figure 6.24: Performance results for Experiment 2 where the number of lags is varied. "tLags"is the number of lags of the response variable. "xLags" is the number of lags of each exogenous input variable. We focus on the MSE and MAPE The plot titles indicate what is being shown; e.g. "NN test MSE" is the MSE test performance for the Neural Network model.

The main conclusion to be drawn from Fig. 6.24 is that, regardless of the exogenous input lags ("xLags"), including at least one lag of the response variable ("tLags") significantly improves performance (lower MSE and MAPE). From the third row of plots, showing the *training* performances for the SVM model, we see from the downward sloping curves that including additional lags of the response variable further improves the *training* results (and this also holds for all values of "xLags"). However, the curves in the other plots, for the *test* performances, somewhat flatten out for larger "tLags", which means that including additional response lags does not seem to improve the *test* performance—which is really what we are interested in since this is a measure of the model's ability to generalize to unseen (i.e. unknown, future) data. Of course, this is the general picture; in some cases, larger "tLags" is still seen to yield better performances.

The results in Fig. 6.24 also indicate that the number of lags of the exogenous inputs is not as important as that first lag of the response variable. Moreover, the importance of "xLags" appears to somewhat vary. In most cases of the *test* performance, there are only marginal differences between using few lags (e.g. 0 or 1) of the exogenous inputs and using many lags (e.g. 4 or 5). However, especially for SVM—which, due to not using randomization, gives the most concrete and consistent results in this experiment—we actually observe significant training performance improvements for additional "xLags" (as can be seen from third row of plots, where larger "xLags" yield lower MSE and MAPE).

Experiment 3

Whereas Experiment 2 above investigated the effect of varying the number of lags for a particular exogenous input combination, this experiment turns things around and explores a variety of different exogenous input combinations for a given number of lags. The data and models used are:

Data and time period: Daily prices of the NASDAQ Index from 01 Jan 2010 - 01 Jan 2014.

Pre-processing and transformation: 5-period EMA of **daily** prices; logarithmic returns; normalization to zero mean, unit variance.

Inputs: The input variables are subject of variation; we will mainly consider an array of technical indicators and response data. The number of lags of each variable is kept at 3.[34]

Data Split: The effective 976 observations are split into training/test sets of 90%/10%.

Model(s): ν-SVR with RBF kernel, $C = 1$, $\gamma = 1/n_i$, $\nu = 0.9$. Neural Network (NN) trained with Levenberg-Marquardt algorithm and hidden neurons optimized using cross-validation. Random Forest (RF) with $n = 100$ trees and leaf size optimized using out-of-bag error.

The parameters have been chosen and/or optimized for the same reasons discussed in Experiment 2 (section 6.2) above. The input combinations are listed in Table 6.12.[35] Note from Table 6.12 that the odd input combination IDs do *not* include lagged response data. The input combinations with even IDs do so, though, allowing us to investigate the effect of adding the response variable. Note also that we have chosen to consider this limited array of combinations;

[34]Note that the exogenous inputs have been differenced to obtain stationarity, similarly to the use of logarithmic returns instead of raw prices. The use of differenced exogenous inputs is investigated in Experiment 4 below.

[35]MOM = Momentum. MACD = Moving Average Convergence Divergence. RSI = Relative Strength Index. OBV = On Balance Volume. WPCTR = Williams' %R. CHOSC = Chaikin Oscillator. CHVOL = Chaikin Volatility.

investigating each and every possible permutation of the available exogenous inputs is an extremely involved task beyond the scope of this study. The results (on the test set) are shown in Fig. 6.25.

ID	**Input combination**
1	MOM
2	MOM + y
3	MOM + MACD + RSI
4	MOM + MACD + RSI + y
5	MOM + MACD + RSI + OBV
6	MOM + MACD + RSI + OBV + y
7	MOM + MACD + RSI + OBV + WPCTR
8	MOM + MACD + RSI + OBV + WPCTR + y
9	MOM + MACD + RSI + OBV + WPCTR + CHOSC + CHVOL
10	MOM + MACD + RSI + OBV + WPCTR + CHOSC + CHVOL + y

Table 6.12: Input combinations in Experiment 3. y denotes the response variable. We have used lags from 1 to 3 for each input combination.

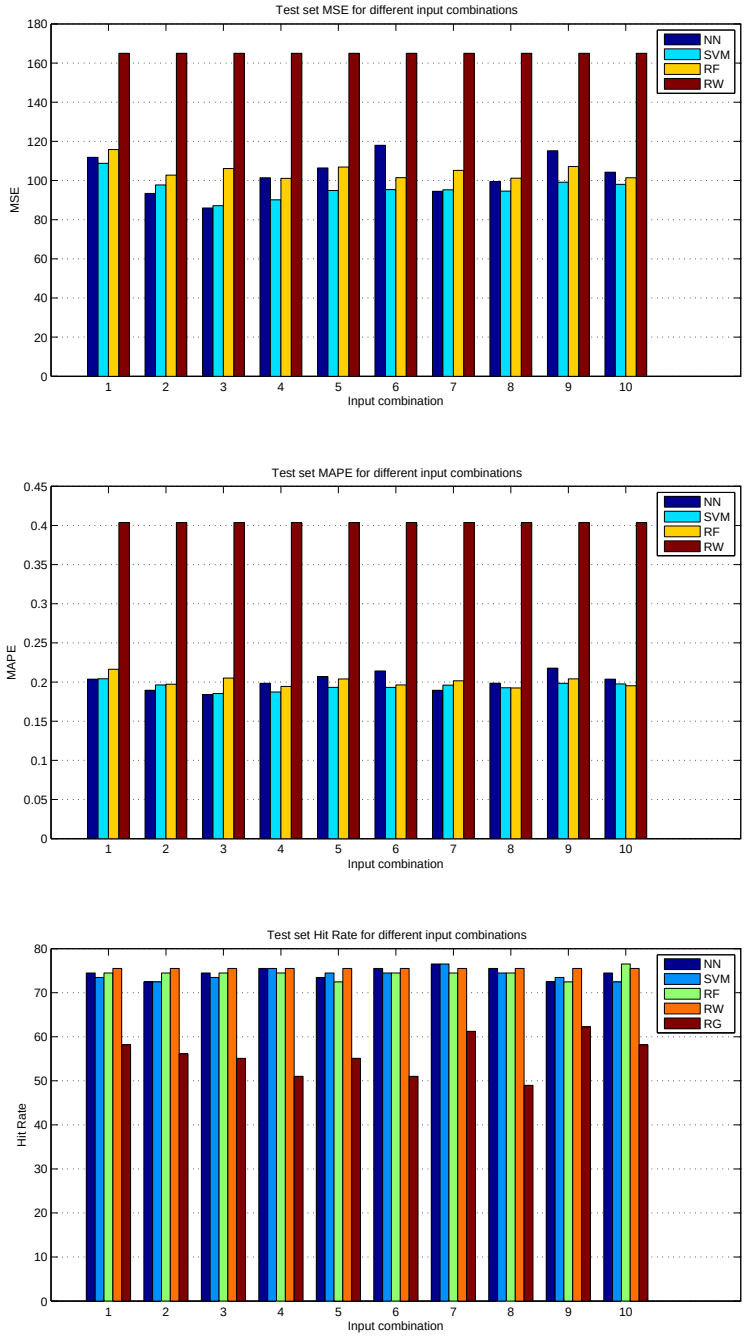

Figure 6.25: Experiment 3: Test set performances for different input combinations. Top: MSE (lower is better). Middle: MAPE (lower is better). Bottom: Hit Rate (higher is better). See Table 6.12 for what input combination the numbers on the horizontal axis refer to. The models are: Neural Network (NN), Support Vector Machine (SVM), Random Forest (RF). We have included the Random Walk ("tomorrow = today"), and the Random Guess ("coin flip") as benchmarks. RG can only give a Hit Rate performance.

The top and middle plots in Fig. 6.25 show that our three models (NN, SVM, RF) yield significantly better performances (lower MSE and MAPE) on the unseen test set than the simple Random Walk (RW) model (which predicts tomorrow's price as today's price). In terms of Hit Rate (correct up/down movement predictions), however, the four models are very equal. But the NN, SVM, and RF models still outperform by far the simple Random Guess (RG) "coin flip" method.[36]

What is more interesting in this experiment, however, is the performances for different input combinations. Here, we remark the following conclusions that can be drawn from Fig. 6.25:

- Especially the MAPE and Hit Rate performance measures for each model are all fairly constant across the different input combinations. The models (especially NN) exhibit more variability in the MSE performance measure (top plot).
- The roughly constant performances indicate that adding additional technical indicators as exogenous inputs (as is done by going through input combination IDs 1, 3, 5, 7, and 9) does not necessarily improve performance. For example, adding MACD and RSI to MOM (combination 3 vs. 1) yields better MSE and MAPE for all three models. But further adding OBV (input combination 5 vs. 3) worsens performance. On the other hand, performance is again slightly improved in combination #7 which adds WPCTR, but worsened in #9 which adds CHOSC and CHVOL. It is important to note, though, that the differences are very small, sometimes marginal.
- Comparing combinations with odd IDs to those with even IDs (e.g. #1 to #2, #3 to #4, etc.) shows that adding lagged response data most often improves performance. For example, adding lagged response data to the MOM input (#2 vs. #1) decreases MSE and MAPE. The same holds when comparing #9 and #10, and also other pairs. But it is not always that adding lagged response improves performance.

The results are therefore, to some extent, inconclusive. Indeed, the randomness inherent in Neural Networks and Random Forests cause different simulations and test runs to yield slightly different results—even though we have minimized these effects through various measures; e.g. by training several neural networks to re-initialize the random weights, etc. This inherent randomness is likely the reason for the varying results observed, in particular, for the Neural Network—which, as we have found after numerous runs and simulations, is also the model yielding the most inconsistent results across runs with identical settings.

However, irrespective of the slight variations, our results show that very good performances can be achieved through the use of technical indicators alone (i.e. without lagged response data), which is actually quite remarkable.

Experiment 4

Spurred by the use of (stationary) logarithmic returns instead of (non-stationary) raw prices, we now investigate whether differencing[37] (to obtain stationarity) the exogenous input data improves forecasting performance. The setup is as follows:

Data and time period: Daily prices of the NASDAQ Index from 01 Jan 2010 - 01 Jan 2014.

Pre-processing and transformation: 5-period EMA of **daily** prices; logarithmic returns; normalization to zero mean, unit variance.

[36] Our Random Guess method simulates a real scenario consisting of a finite number of coin flips, which is why the Hit Rate results in the bottom plot in Fig. 6.25 are different from (but fluctuate around) 50%.

[37] The first difference of a series $y_{i=1}^{N}$ is the series $y_i - y_{i-1}{}_{i=2}^{N}$.

Inputs: Two combinations: 1) 3 lags of MOM, RSI, and MACD alone ($n_i = 9$). 2) 3 lags of MOM, RSI, and MACD and 3 lags of the response ($n_i = 12$). For each combination we try compare raw vs. differenced exogenous inputs.

Data Split: The effective 976 observations are split into training/test sets of 90%/10%.

Model(s): ν-SVR with RBF kernel, $C = 1$, $\gamma = 1/n_i$, $\nu = 0.9$. Neural Network (NN) trained with Levenberg-Marquardt algorithm and hidden neurons optimized using cross-validation. Random Forest (RF) with $n = 100$ trees and leaf size optimized using out-of-bag error.

Note that we consider two different input combinations: 3 lags of MOM, RSI, and MACD with and without 3 lags of the response. This enables us to see whether differencing not only has an effect in general, but also if the potential effect is different in the absence or presence of lagged response data. As a result of these two input combinations, and hence two different numbers of inputs, we optimize the model parameters by specifying an n_i-dependant γ for SVM (the chosen C and ν have been found to work best in all cases), choosing for NN a number of hidden neurons based on cross-validation, and choosing for RF a leaf size based on the out-of-bag error. This ensures improved performances and a more fair and balanced comparison between different input combinations.

Moreover, as evident in the attached MATLAB program, we perform for each case and model 10 iterations and compute the mean performance measures. In addition to our other implemented measures as described earlier, this further reduces the random effects inherent in Neural Networks and Random Forests, and allows us to see in a much more consistent and conclusive way any potential performance effects between raw and differenced exogenous inputs.

The results are shown in Fig. 6.26. The RW and RG benchmarks are not included, since they are not relevant to the purpose of this experiment.[38]

[38]However, our MATLAB program can still output and show the benchmark performances. The results are the same as earlier; our NN, SVM, and RF models have similar Hit Rates as RW, but significantly outperform it in terms of MSE and MAPE. They also outperform by far the simple RG.

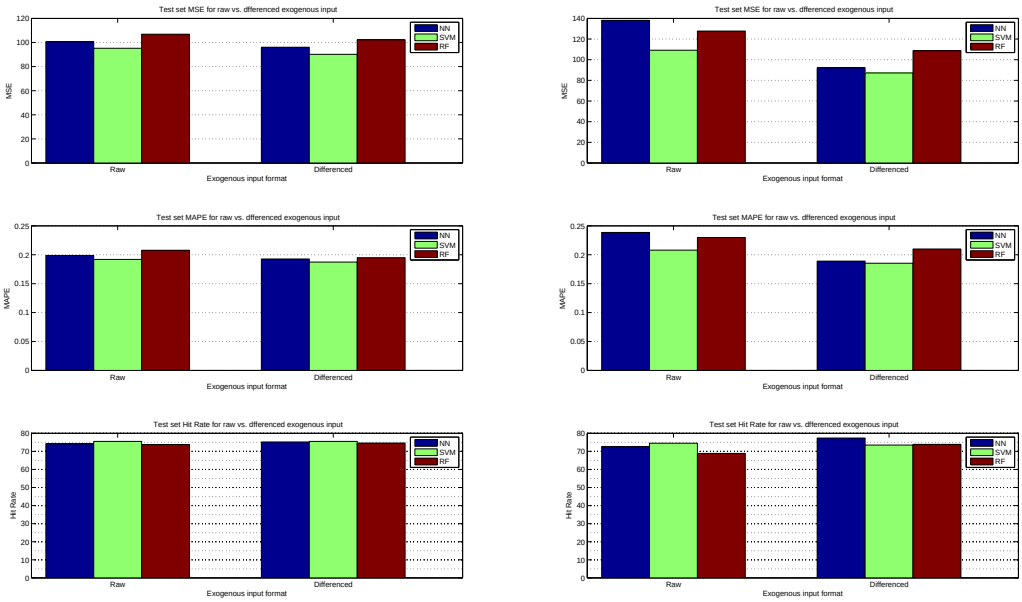

Figure 6.26: Experiment 4: Test set performances for raw vs. differenced exogenous input. Left side: Input combination with lagged MOM, RSI, and MACD and **with** lagged response. Right side: MOM, RSI, and MACD and **without** lagged response. Top: MSE (lower is better). Middle: MAPE (lower is better). Bottom: Hit Rate (higher is better). The inputs are 3 lags of MOM, RSI, and MACD, including 3 lags of the response variable. The models are: Neural Network (NN), Support Vector Machine (SVM), and Random Forest (RF).

When including lagged response as input (left column of plots in Fig. 6.26), we see that differencing the exogenous inputs does yield slightly better performances (lower MSE and MAPE, and higher Hit Rate[39]). This goes for all three models.

When not including lagged response data (right side of Fig. 6.26), i.e. using exogenous inputs alone, we also see that differencing them yields better performance than when using them in raw form. But the important thing to notice is that the degree of improvement is larger; there are greater decreases in MSE and MAPE, and increases in Hit Rate, than before.

In conclusion, differencing the exogenous inputs yields better performance in general. Also, the performance *improvement* is greater when using exogenous input alone than when including lagged response data. So the effect of differencing the exogenous inputs is more pronounced when using them alone as predictors—but the effect is still there even when including lags of the response variable.

Experiment 5

This experiment investigates whether additional training data (i.e., historical data further and further in the past) improves performance. The setup is as follows:

Data and time period: Daily prices of the NASDAQ Index. The start date of the time period is varied, while the final date is kept at Jan 01 2014.

[39]The differences in Hit Rate are very small because this performance measure is not as distinguished as the other two.

Pre-processing and transformation: 5-period EMA of **daily** prices; logarithmic returns; normalization to zero mean, unit variance.

Inputs: 3 lags of MOM, RSI, and MACD and 3 lags of the response ($n_i = 12$).

Data Split: The total number of observations vary due to the varying time periods. In each case, we keep the number of test set observations at 50, while the number of training observations vary.[40]

Model(s): ν-SVR with RBF kernel, $C = 1$, $\gamma = 1/n_i$, $\nu = 0.9$. Neural Network (NN) trained with Levenberg-Marquardt algorithm and 5 hidden neurons. Random Forest (RF) with $n = 100$ trees and a leaf size of 20.

We consider 12 different starting dates, ranging with increasing increments from Jun 01 2013 (giving a time period of 6 months) all the way back to Jan 01 2000 (a time period of 14 years $\sim$ 3500 trading days). This gives 12 different values for the number of training observations. In each case, and for each model, we compute performances 20 times and average them. This reduces the performance influences that may be due to the random effects inherent in Neural Networks and Random Forests, and thus allows us to better see the potential performance effects resulting from varying the amount of training data. The *test set* performance results are shown in Fig. 6.27.[41]

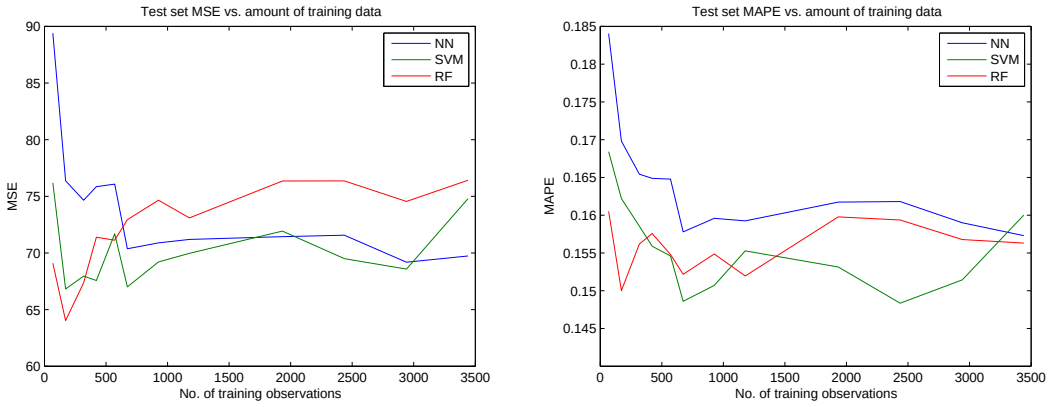

Figure 6.27: Experiment 5: Test set performances for varying amounts of training data. Left: MSE (lower is better). Right: MAPE (lower is better). The inputs are 3 lags of the response and 3 lags of MOM, RSI, and MACD. The models are: Neural Network (NN), Support Vector Machine (SVM), and Random Forest (RF).

For each model, the MSE and MAPE test performance measures in Fig. 6.27 generally exhibit the same behaviour. It should be noted, though, that the MAPE might be slightly more reliable when including prices very far back in the past. The reason is that stock prices over such a

[40] It is important that the amount of test data is kept constant in this case, so as to ensure a fair comparison of performances across the cases with different amounts of training data.

[41] Due to its relatively less quantitative nature (measuring only the percentage of correct up/down predictions), the Hit Rate performance measure is not as distinguished as the MSE and MAPE, which is why we have excluded it from Fig. 6.27. However, our attached MATLAB program still shows it. In addition, the program also shows plots of the training performances.

long time period may change scale. Due to its percentage nature, MAPE is invariant to such scale changes. MSE, on the other hand, may lend more weight (larger error contribution) to the most recent observations, or generally those with largest magnitude.[42] This aside, Fig. 6.27 shows a significant *initial* improvement in test performance (large drop in MSE and MAPE), but mainly for NN and SVM. The RF error also decreases slightly, but rises back up. For the remaining number of training observations, the RF error is fairly constant. The slight differences between the RF error and those for NN and SVM may be ascribed to the fact that RF is an ensemble method while NN and SVM are not. Consequently, through the added robustness and power, RF might need fewer observations to achieve good generalization—which explains its minimum MSE and MAPE occurring before (at a smaller value for the number of training observations) the minimum for NN and SVM. As for these latter two models, they exhibit a significant performance boost in the beginning; i.e., increasing the training data set when it is small. The performance further improves up to the point of minimum MSE and MAPE occurring at roughly 700 training observations ($\sim$ 3 years of historical daily prices). After this, NN in particular is fairly constant; neither more training data seems to neither improve or reduce performance. The SVM performance varies a bit more and actually ends up increasing near the end, indicating that such large amounts ($\sim$ 14 years) of historical data are improper.

In conclusion, using too much training data is not good for any of the models; it does not significantly improve performance (in many cases quite the contrary), and, additionally, it only increases the computation times. We find that using about 3 years' worth of historical data yields the best performance.[43]

Experiment 6

Experiment 3 investigated a selection of different technical indicators as exogenous inputs; we tried different numbers of indicators, but we only considered *one* combination in each case. In this experiment we consider the same seven technical indicators, but for each number (one through seven) we go through *all* possible combinations, thus finding the optimal combinations.

Data and time period: Daily prices of the NASDAQ Index from 01 Jan 2010 - 01 Jan 2014.

Pre-processing and transformation: 5-period EMA of **daily** prices; logarithmic returns; normalization to zero mean, unit variance.

Inputs: Subject of variation. All possible combinations of $n = 7$ technical indicators[44] taken $k = 1, 2, \ldots, 7$ at a time. 3 lags are used for each input, including the response data.

Data Split: The effective 976 observations are split into training/test sets of 90%/10%.

Model(s): ν-SVR with RBF kernel, $C = 1$, $\gamma = 1/n_i$, $\nu = 0.9$. Neural Network (NN) trained with Levenberg-Marquardt algorithm and hidden neurons optimized using cross-validation. Random Forest (RF) with $n = 100$ trees and leaf size optimized using out-of-bag error.

So, the question we examine in this experiment is: Using k technical indicators as exogenous input, which combination yields the best performance?[45] I.e., which technical indicators possess

[42] Computing the MSE performance using the normalized and differenced data solves this problem altogether, but also takes away some of the information of the resulting error value; it is easier to interpret and understand an error value computed using prices than one calculated via some pre-processed, transformed data type.

[43] 3 years $\sim$ 750 trading days. The effective number of observations is slightly smaller, because some observations are needed to initialize the technical indicators, etc.

[44] The indicators are: MOM, RSI, MACD, OBV, WPCTR, CHOSC, CHVOL. They will be denoted by numbers 1 through 7 for easy notation. See Experiment 4 (Section 6.2) or Appendix A for further information.

[45] In this experiment we have used the MSE measure as the performance criterion, as evident in or MATLAB program. Lower MSE is better.

the best predictive power? The results are shown for each k (up to $n = 7$) in Table 6.13.

k	${}_nC_k$	**NN**	**SVM**	**RF**
1	7	2	5	1
2	21	3, 6	5, 6	1, 6
3	35	1, 2, 5	3, 5, 6	1, 2, 6
4	35	1, 3, 5, 6	2, 3, 5, 6	1, 2, 6, 7
5	21	1, 2, 3, 4, 5	2, 3, 4, 5, 6	1, 2, 3, 5, 6
6	7	1, 2, 3, 4, 5, 7	1, 2, 3, 4, 5, 6	1, 3, 5, 5, 6, 7
7	1	1, 2, 3, 4, 5, 6, 7	1, 2, 3, 4, 5, 6, 7	1, 2, 3, 4, 5, 6, 7

Table 6.13: Experiment 6: Best input combinations. k is the number of inputs used out of the total $n = 7$. The number of ways of choosing k from n is ${}_nC_k \equiv \binom{n}{k} \equiv \frac{n!}{k!(n-k)!}$. The numbers in the NN, SVM, and RF columns are the inputs: 1 = MOM, 2 = RSI, 3 = MACD, 4 = OBV, 5 = WPCTR, 6 = CHOSC, 7 = CHVOL.

The case $k = 7$ is trivial, as it has only one possible combination. For a particular k, the optimal combinations are clearly not the same for the three models. This may be ascribed to randomness or the possibility that some indicators might work better or worse when used alone or in conjunction. But some indicators do appear more than others. Since most of them are obviously included for large k, it interesting to see which indicators are excluded as we go down from $k = 7$. One striking feature is that CHVOL (no. 7) is quickly left out when we reach $k = 5$. And MACD (no. 4) is left out when we reach $k = 4$. The most prominent indicators for $k = 3$ are MOM, RSI, WPCTR, and CHOSC (1, 2, 5, and 6). And for $k = 2$, it is interesting to see that CHOSC (no. 6) appears for all three models.

We have also found the very best combination (over all the different k) for each method.

Best overall combinations:

NN: 1, 2, 5 (MOM, RSI, WPCTR)
SVM: 3, 5, 6 (MACD, WPCTR, CHOSC)
RF: 1, 2, 6 (MOM, RSI, CHOSC)

The most optimal input combination is thus not the same for the three methods. This may be ascribed to the randomness inherent in especially Neural Networks and Random Forests, or it may be due to the possibility that the methods actually favour different exogenous inputs.

In any case, though, it is interesting to see that, out of *all* possible combinations using $k = 1, 2, \ldots, 7$ technical indicators, the optimal combination for each method uses $k = 3$ indicators. It is also interesting to look at the specific indicators, as these are much the same; only five out of the total seven indicators are represented above, one of which only appears once. The four most prominent exogenous inputs are 1, 2, 5, and 6. In this case, the technical indicators with most predictive power are thus MOM, RSI, WPCTR, and CHOSC.

Experiment 7

So far, we have focused on daily data. This experiment investigates whether weekly or monthly data yields better results.[46]

[46]It is beyond the scope of this study to perform an in-depth analysis of which data frequency is best for forecasting. This would require tests for a multitude of different time periods and assets. However, the subject is

Data and time period: Weekly prices of the NASDAQ Index for the 14-year time period from 01 Jan 2000 - 01 Jan 2014.

Pre-processing and transformation: 5-period EMA of prices; logarithmic returns; normalization to zero mean, unit variance.

Inputs: 3 lags of RSI, MACD, WPCTR, and CHOSC, and 3 lags of the response ($n_i = 15$).

Data Split: The 700 effective observations are split into training/test sets of 600/100.

Model(s): ν-SVR with RBF kernel, $C = 1$, $\gamma = 1/n_i$, $\nu = 0.9$. Neural Network (NN) trained with Levenberg-Marquardt algorithm and hidden neurons optimized using cross-validation. Random Forest (RF) with $n = 100$ trees and leaf size optimized using out-of-bag error. ARMA(1,1)-GARCH(1,1) model.

The performance measures (for the unseen test data) are listed in Table 6.14 and the predictions shown in Fig. 6.28.

Model	MSE	MAPE	Hit Rate
NN	387.33	0.459 %	78.00 %
SVM	406.08	0.496 %	75.00 %
RF	369.49	0.466 %	74.00 %
AG	344.05	0.446 %	74.00 %
RW	736.87	0.720 %	74.00 %
RG	N/A	N/A	45.00 %

Table 6.14: Experiment 7: Test set performances for weekly NASDAQ data. NN: Neural Network. SVM: Support Vector Machine. RF: Random Forest. AG: ARMA-GARCH. RW: Random Walk. RG: Random Guess. See the summary in the text for details.

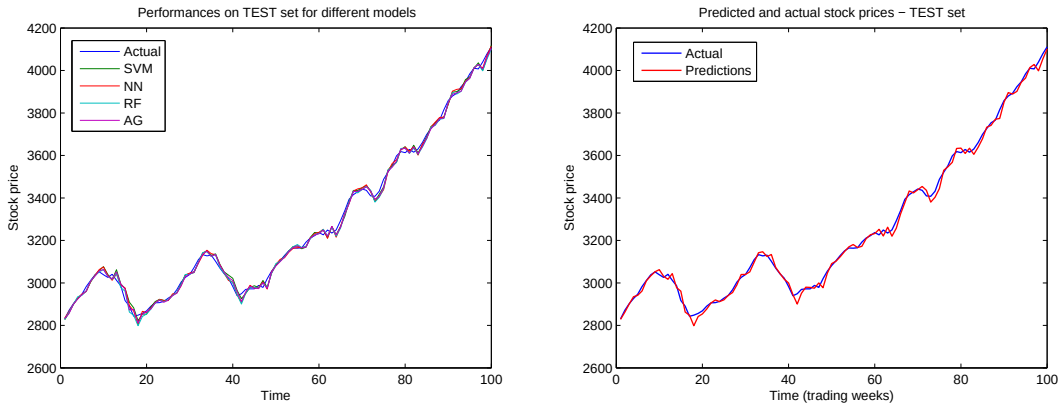

Figure 6.28: Experiment 7: Test set predictions (red) and actual prices (blue) for weekly NASDAQ data. Left plot: All models included. Right plot: Predictions for the different models lie extremely close, so only one is shown (RF) to avoid clutter.

a candidate for future research.

The results—both the quantitative performance measures and the plots of the predictions—indicate that using weekly data does not significantly relieve the problems; the crucial shortcomings encountered with daily data are still present here when using weekly data. We see this from the fact that the Hit Rates are not significantly better than that obtained by the Random Walk model. And we see it from the plots where the models fail to properly predict the stock price when it makes a sudden significant move in the opposite direction (away from the current trend). However, our models still significantly outperform the RW benchmark in terms of MSE and MAPE, indicating that their predictions are much more precise and closer to the actual prices. Our models also outperform by far the simple Random Guess "coin flip" benchmark.

Our MATLAB programs also allow for the option to use monthly data. This gives us much fewer observations to work with for training and testing the models. But we are still able to get some results. These, and the conclusions that may be drawn from them, are similar to the those for weekly data above, and we leave them out for the sake of brevity.[47]

Experiment 8

Regarding financial analysis, our main focus has been technical analysis, as reflected by the use of technical indicators in all of the previous experiments. In this experiment we draw upon some fundamental analysis in the shape of interest rates, oil and gold prices, etc. We also consider the S&P 500 Index instead of the NASDAQ Index. We summarize as follows:

Data and time period: Daily prices of the S&P 500 Index from 01 Jan 2010 - 01 Jan 2014.

Pre-processing and transformation: 5-period EMA of daily prices; logarithmic returns; normalization to zero mean, unit variance.

Inputs: 3 lags of the response, including 3 lags of each of the following: S&P 500 Volatility Index, CBOE Interest Rate 10-year Treasury Note Index, iPath S&P GSCI Crude Oil Total Return ETF Index, and SPDR Gold Shares Index.

Data Split: The effective 976 observations are split into training/test sets of 90%/10%.

Model(s): ν-SVR with RBF kernel, $C = 1$, $\gamma = 1/n_i$, $\nu = 0.9$. Neural Network (NN) trained with Levenberg-Marquardt algorithm and hidden neurons optimized using cross-validation. Random Forest (RF) with $n = 100$ trees and leaf size optimized using out-of-bag error.

Our exogenous inputs include indexes that track fundamental, macro-economic variables such as the volatility of the S&P 500 index, the interest rate, the oil price, and the gold price.

For the sake of comparison we actually perform a total of four experiments where the exogenous inputs are as follows: 1) none, 2) only fundamental, 3) only technical, and 4) both fundamental and technical. The technical indicators are MACD, RSI, WPCTR, CHOSC, and OBV. Each of the four cases also used 3 lags of the response data. In each case we compute model performances twenty times and average them, so as to reduce the effects of the randomness inherent in NNs and RFs. This better ensures that any potential differences in the results are actually due to the different types of exogenous input. The results are shown in Fig. 6.29.

[47] Later, in the Conclusion (Chapter 7), we discuss how monthly data may be used for long-term forecasting in a more adequate way.

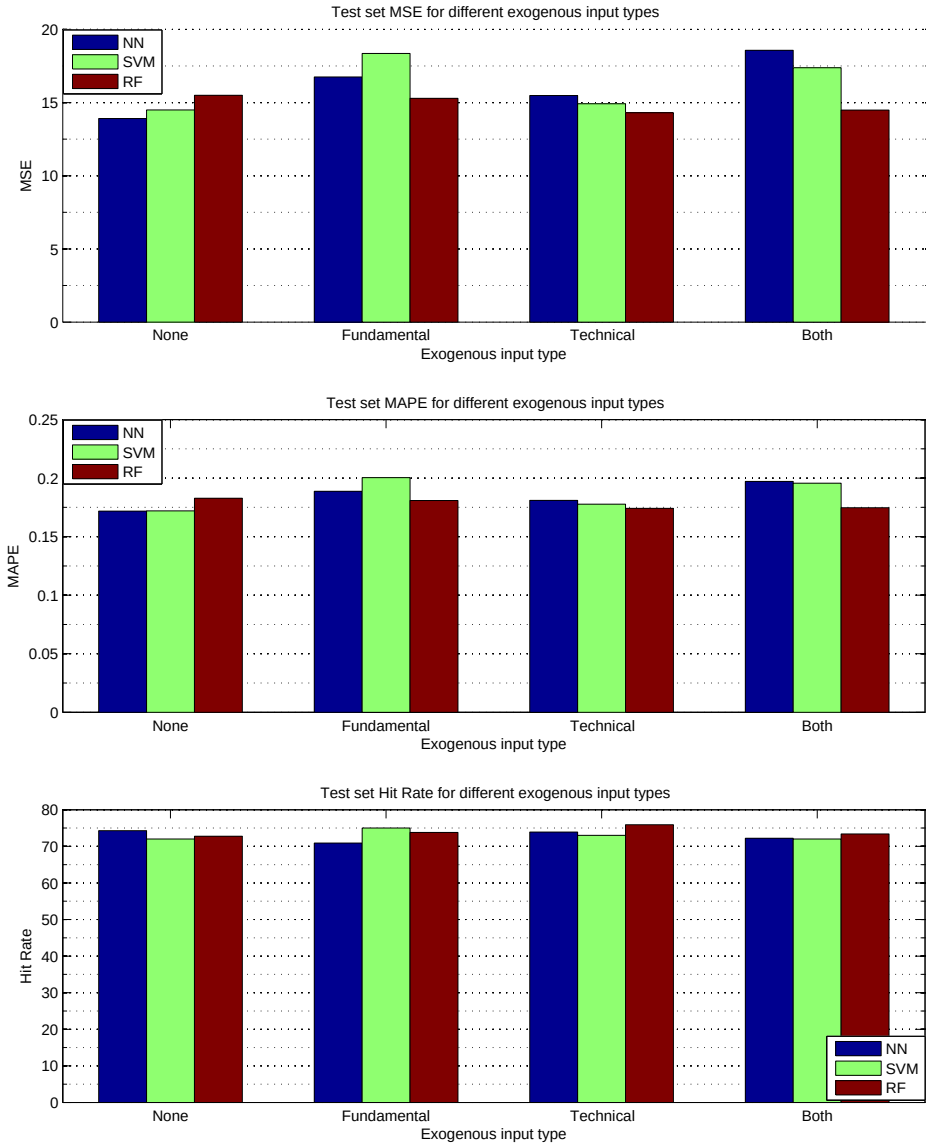

Figure 6.29: Experiment 8: Test set performances for the four cases using as exogenous inputs: 1) nothing, 2) fundamental data, 3) technical data, and 4) both technical and fundamental data. Top: MSE (lower is better). Middle: MAPE (lower is better). Bottom: Hit Rate (higher is better). The models are: Neural Network (NN), Support Vector Machine (SVM), and Random Forest (RF). For comparison, our Random Walk benchmark model yielded MSE = 26.5, MAPE = 0.25%, and HR = 76%. The Random Guess benchmark naturally gave HR $\approx$ 50%.

From Fig. 6.29 we see that using fundamental data on its own generally gives poorer performance (higher MSE and MAPE) than using technical data. This holds for all three methods, but is most pronounced for SVMs (green bars). The changes in MSE and MAPE for RF (red bars) are minuscule, highlighting the robust nature of this ensemble learning method.

In some cases (NN and SVM), using neither fundamental nor technical data (i.e. only past prices) yields better results than when using fundamental data, as can be seen from the blue and green MSE and MAPE bars increasing as we go from "None" to "Fundamental". But this doesn't hold for RF, which stays constant.

Finally, using both fundamental and technical data generally worsens the performance. But only for NN and SVM; RF still stays at roughly the same values. This worsening of the performance may be ascribed to the large amount of inputs and the models becoming increasingly complex. Consequently, even though we have taken measures against it, they may be more prone to overfitting. Fig. 6.29 only shows the test performances, but our program also outputs the training performances. Consulting these, one sees that the combination of technical and fundamental data leads to training performances that are better than when using only one type of exogenous input or none at all.

The results in Fig. 6.29 were obtained using daily data. The main conclusion was that using fundamental data did not give better results than using technical. First, this may be ascribed to the type of fundamental data used here; we considered four indexes that should track some fundamental quantities (the interest rate and the price of oil and gold). However, a full-fledged fundamental analysis of a company normally includes an abundance of additional data, including e.g. key numbers from their budget and account, price, earnings, etc. Similarly, a thorough analysis of the market includes numerous indicators for the overall state of the economy, e.g. housing prices, unemployment, debt, spending, interest rates, exchange rates, and much more. It is very difficult to obtain this type of data with the proper quality. In addition, this type of data is often quoted less frequently, which brings us to the next point: macro-economic indicators are related to the state of the economy over long time periods, and fundamental data and analysis are similarly associated to investments with long time horizons. Thus, using this type of data is not as adequate for day-to-day prediction as technical data (which is meant to be very proper for short-term trading). This may be a contributing factor to the conclusion drawn here. Fundamental data may be more suitable if one uses stock prices and other data quoted on e.g. a monthly basis.[48]

Experiment 9

In this section we perform predictions for numerous different assets, including stock indexes, individual stocks, commodities, exchange rates, etc. The general settings can be summarized as follows:

Data and time period: Daily prices from 01 Jan 2010 - 01 Jan 2014. Different assets are considered.

Pre-processing and transformation: 5-period EMA of daily prices; logarithmic returns; normalization to zero mean, unit variance.

Inputs: 4 lags of the response and 4 lags of each of the RSI, MACD, WPCTR, and CHOSC technical indicators.

Data Split: The effective 976 observations are split into training/test sets of 876/100 ($\sim 90\%/10\%$).

[48]Some of the fundamental indicator datasets used here only go as far back as around eight years. This is a whole lot of data if quoted on a daily basis. But on a monthly basis it only yields about 80-90 observations. This is too few observations to make out a properly-sized training set. And the remaining observations would make an even smaller test set, which makes the performance measures questionable. The aggregation of other, larger fundamental indicator datasets, and their use in financial forecasting, is a topic for future work.

Model(s): ν-SVR with RBF kernel, $C = 1$, $\gamma = 1/n_i$, $\nu = 0.9$. Neural Network (NN) trained with Levenberg-Marquardt algorithm and hidden neurons optimized using cross-validation. Random Forest (RF) with $n = 100$ trees and leaf size optimized using out-of-bag error. ARMA(1,1)-GARCH(1,1) with t innovation distribution. Random Walk and Random Guess models.

The best prediction performance results are shown in Table 6.15 below.[49] We only include the performances on the unseen test set, which is the most interesting. Note also that we only use the MAPE and Hit Rate measures; MSE is left out since it depends on the scale and magnitude of prices, which varies for different assets.[50]

	Test MAPE (%)					TEST Hit Rate (%)					
Asset	NN	SVM	RF	AG	RW	NN	SVM	RF	AG	RW	RG
NASDAQ	0.184	0.187	0.190	0.185	0.268	77.00	78.00	75.00	68.00	74.00	48.00
SP500	0.176	0.173	0.172	0.169	0.248	78.00	77.00	79.00	76.00	76.00	43.00
AAPL	0.405	0.427	0.394	0.392	0.517	73.00	76.00	71.00	72.00	76.00	39.00
MSFT	0.348	0.353	0.352	0.363	0.521	81.00	78.00	79.00	75.00	76.00	54.00
T	0.246	0.244	0.237	0.236	0.337	76.00	75.00	72.00	73.00	71.00	48.00
F	0.323	0.361	0.347	0.340	0.439	74.00	76.00	77.00	70.00	71.00	46.00
FB	0.673	0.655	0.731	0.642	0.927	80.00	80.00	86.00	79.00	81.00	47.00
GLD	0.324	0.321	0.303	0.307	0.414	76.00	77.00	74.00	74.00	73.00	52.00

Table 6.15: Experiment 9: Best prediction performances (on unseen test set) for different assets. NN: Neural Network. SVM: Support Vector Machine. RF: Random Forest. AG: ARMA-GARCH. RW: Random Walk. RG: Random Guess. See the summary in the text for details.

From Table 6.15 we see that our models (NN, SVM, RF, and AG) all outperform the Random Walk benchmark in terms of the MAPE measure. For each respective asset, the four models yield very similar MAPE results; e.g. $\sim 0.19\%$ for NASDAQ, $\sim 0.24\%$ for AT&T, etc. In each case, the Random Walk benchmark gives a noticeably larger MAPE; e.g. 0.27% for NASDAQ and 0.34% for AT&T.

In terms of Hit Rate accuracy, the four models also yield extremely good results (at first sight, at least) with Hit Rates upwards of 80%. This means that the models correctly predict the up/down price movement direction more than three out of four times! Indeed, the NN model shows a Hit Rate of 81% for the Microsoft stock, and the RF model gives a value as high as 86% for the Facebook stock! However, this initial excitement somewhat wanes when comparing the results to those obtained with the Random Walk benchmark. This benchmark model uses today's price as the prediction of tomorrow's price. So, when this model gives a Hite Rate of e.g. 74% (as it does for NASDAQ), it means that the stock price actually increased 74% of the time in the unseen test set. The SVM Hit Rate of 78% for NASDAQ thus means that the model only correctly predicted an additional 4% of the up/down movements. In almost all cases, the NN, SVM, and RF Hit Rates are a few percentage points larger than the Random Walk benchmark. Although the models are not significantly better than the benchmark in this regard, we still stress the fact that such high Hit Rates are rather good. Indeed, compared to a simple "coin toss" guess on the price direction (as given by the Random Guess benchmark model)[51], the NN,

[49]The abbreviations for the assets are: NASDAQ = NASDAQ Index, SP500 = S&P 500 Index, AAPL = Apple, MSFT = Microsoft, T = AT&T, F = Ford, FB = Facebook, GLD = Gold.

[50]MAPE also varies across different assets, but it is easier to interpret because it measures the percentage deviation.

[51]Note that we have implemented the Random Guess benchmark as an actual "coin toss", which is why the Hit Rates deviate slightly from 50%. In the event of infinite observations, this number naturally tends to 50%.

SVM, RF, and AG models are significantly better. Note also that the three machine learning methods (NN, SVM, RF) generally also give better Hit Rate performances than the conditional mean and variance model (AG).

To summarize, the models considered here yield Hit Rates comparable (but still a little better) than the Random Walk benchmark, and MAPE values that are significantly better. That the Hit Rates are comparable means that our models correctly predict tomorrow's price direction when it is the same as today; and that the models most often fail when this is not so (i.e., in the event of a sudden move in the opposite direction—a reversal). The latter part is crucial, since this is something that would be particularly useful. Failure of the models in this regard thus question their applicability for practical purposes, as well as the predictability of the markets in general.

Experiment 10

In this last experiment we will make use of our oscillator model from Section 3.4. We consider Standard & Poor's 500 Index, using about 100 observations of daily prices.[52] This corresponds to a time period of roughly 5 trading months.

The prediction model used here is the linear one governed by Eq. (3.7). It contains two parameters: the EMA period n of the price level p^*; and the angular frequency ω (or, equivalently, the period T). After trying different values we find that good results (for this particular case) are obtained for $n = 10$ and $T = 20$.[53]

Table 6.16 shows the quantitative performance results. Here we also compare the oscillator model (OM) to the two benchmarks. Fig. 6.30 plots the predictions and the actual data.

Model	MSE	MAPE	Hit Rate
OM	27.26	0.33 %	80.61 %
RW	40.59	0.46 %	79.59 %
RG	N/A	N/A	50 %

Table 6.16: Experiment 10: Performance results for the oscillator model (OM) with $n = 10$ and $T = 20$. RW and RG are the Random Walk ("tomorrow equals today") and Random Guess ("coin flip") benchmarks, respectively.

[52] As in earlier experiments, the raw, erratic prices are smoothed using a 5-period EMA.

[53] A better approach to finding the optimal parameters—which will be pursued in future work—is to use some subset of the data for model selection, searching parameter space for the combination yielding the lowest error compared to the actual data. This is somewhat similar to the training phase of machine learning models.

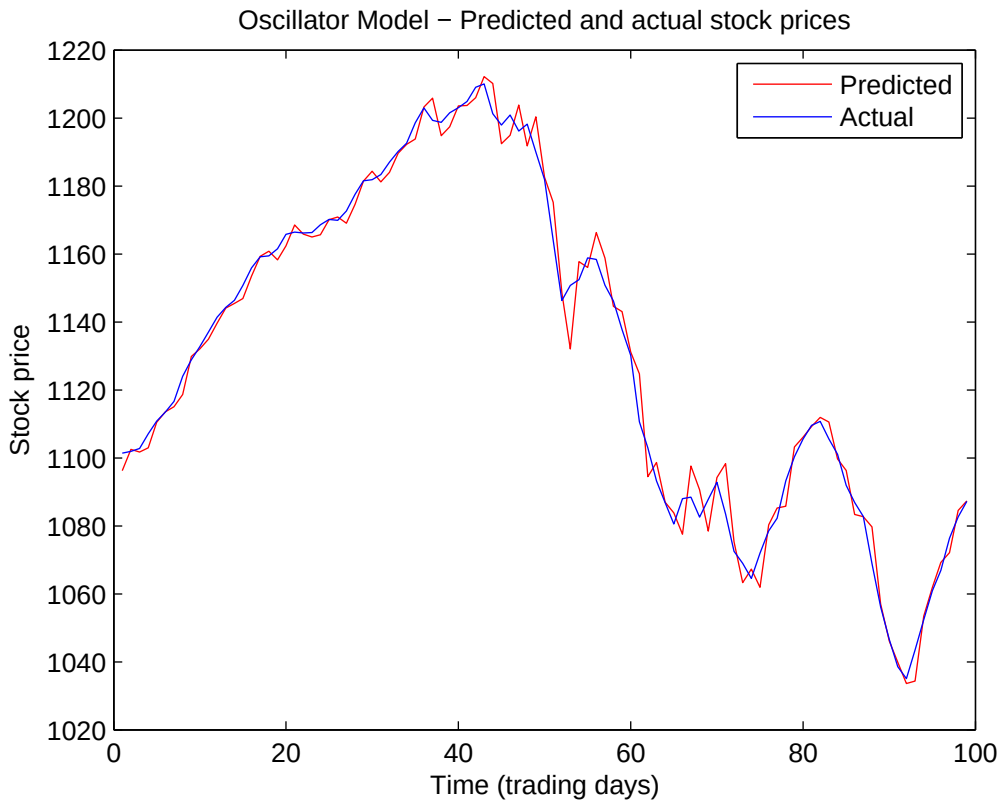

Figure 6.30: Experiment 10: Oscillator model. Predicted prices (red) and actual prices (blue) for ~ 100 daily observations of the S&P 500 Index.

First off, we see from Table 6.16 that the OM model is far better than the simple RG benchmark (which basically just flips a coin to guess whether the price will move up or down). The OM model's Hit Rate is as high as almost 81%, meaning that it correctly predicts the movement direction more than 4 times out of 5. This is truly excellent. And had it not been for our inclusion of the RW benchmark, we would get the wrong picture and think the model far better than it actually is (though it still is quite good). For as it happens, the RW benchmark shows a Hit Rate of almost 80%. Due to its "tomorrow equals today" nature, this means that the day-to-day price *directions* were actually the same almost 4 times out of 5. So, using this simple prediction rule nets a similar result as our model—but only in terms of the Hit Rate. Turning to the MSE and MAPE measures (lower are better), our OM model significantly outperforms the RW benchmark. The lower MSE and MAPE values indicate that the model's predictions are much more precise than the benchmark's, and closer to the actual prices.[54]

Fig. 6.30 gives a visual feel for how close the predicted and actual prices are. We see that there is splendid agreement when the price is moving in an even trend; e.g. in the first 35 days (up-trend), between days 55-65 (down-trend), and from day 75 until the end (up-trend, down-trend, up-trend). However, when the price is moving in a more erratic way, with sudden rises or drops, the OM model has a harder time keeping up. In these cases (e.g. between days 35-50,

[54]These results—comparable Hit Rate, but better MSE and MAPE, for our model compared to the RW benchmark—were also what we obtained for our NN, SVM, and RF models in the previous experiments.

50-55, and 65-75), the model's predictions are quite sharp and "spiky"; this is mostly where the model gets the direction wrong, and also the main contribution to the errors.

In conclusion, our OM model seems to exhibit some of the same qualities and results as our NN, SVM, and RF models. First, our models, whilst vastly outperforming the simple RG benchmark, yield comparable Hit Rates to the RW benchmark. However, they perform significantly better in terms of the more quantitative, refined MSE and MAPE measures. Second, like the NN, SVM, and RF models, our OM model performs best in those periods where the price exhibits a general trend, and poorest when the price fluctuates erratically back and forth.

However, the good results for the OM model indicate that there might be some use to modelling stock prices depending on their movement relative to some average, base price level (as is the essence of our OM model). And, seeing that we have here used only the simple, linear version of the model, there might be even better results to obtain by using the extended, non-linear version of the model, which we also described back in Section 3.4. This will be an interesting subject for future work.

Chapter 7

Conclusion

This thesis has considered financial forecasting, and more specifically the problem of stock market prediction. Initially, we described the financial markets and the fundamental question of whether they are predictable or not. We argued for different opinions on this delicate matter by discussing the Efficient Market Hypothesis, the evolution of the markets, and the technological advances and improvements. Furthermore, we discussed several areas in physics for which forecasting is a vital thing. We raised some striking conceptual and methodological similarities between these fields and financial forecasting, whilst also discussing the crucial differences that really set the disciplines—and perhaps also the possibility of prediction—apart.

After reviewing and discussing the relevant theory, we analysed daily data of the popular NASDAQ Index and performed predictions using ARMA-GARCH models, Artificial Neural Networks, Support Vector Machines, and Random Forests. Our approach to the prediction task was rooted mainly in technical analysis of stocks; that is, we used lagged values of both prices and a wide variety of technical indicators to predict the next period's stock price.[1]

We summarize our results and conclusions below.

- Through a computation of statistical properties and an exploratory data analysis of the daily returns of the NASDAQ Index (Sections 6.1.1 and 6.1.2), we find that the returns are not Gaussian—contrary to common assumption. The problem is the heavy tails of the observed distribution; the probability of extreme events (especially the risk of large losses) is significantly underestimated by the Gaussian model. This has vital implications for risk management. Our results indicate that the Student's t distribution provides a much better fit.

- In a quantitative and qualitative time series analysis (Section 6.1.3), we find empirical evidence of autocorrelation and conditional heteroscedasticity (volatility clustering) in the daily data of the NASDAQ Index. This has important implications for the potential prediction of the asset; the presence of autocorrelation indicates that recent past events are correlated to current events, and thus that recent past data may be used (to varying degrees of success) for prediction. Similarly, the presence of conditional heteroscedasticity implies that the volatility—a measure of risk—is far from constant in time, but instead gathers in clusters.

[1] Regarding predictions, our focus was on one-step ahead prediction, but we have also performed a little bit of multi-step ahead prediction. Regarding data frequency, our focus was daily data, but we have also made a few experiments with weekly and monthly data.

- We have constructed composite ARMA-GARCH models for the conditional mean and variance of the daily NASDAQ data (Section 6.1.4), and performed AIC- and BIC-based model selection to obtain the optimal parameters. Diagnosing the models, we find that a Student's t distribution is more suitable for the innovation process than the Gaussian, in accord with our observation of the heavy-tailed empirical distribution of asset returns. For multi-step ahead prediction (100 days into the future), we find that the optimized model correctly predicts the future overall trend of the daily NASDAQ prices. For one-step ahead prediction, we find that the optimized model correctly predicts the up/down movement direction 68% of the time. The average error (RMSE) is around 9.5 US Dollars, corresponding in percentage (MAPE) to less than 0.19%. The day-to-day predictions are good, but we make a crucial observation; the model's correct predictions mainly occur when the price is trending and headed in a particular direction.

- Based on parameter suggestions from the literature as well as arbitrary choices, we have trained a simple Artificial Neural Network for predicting and identifying patterns in daily NASDAQ data (Section 6.1.5), and diagnosed the training process using a variety of visual tools. Testing the model on unseen data yields suboptimal performances, indicating the need for proper data pre-processing, choice of inputs and network architecture, etc. We find evidence of a crucial caveat of using raw daily data; the model uses the current day's stock price as the prediction for tomorrow. That the model, during its training process, has learned this to be the best relationship may be considered an indication that the markets cannot be predicted, and consequently may be regarded as support for the Efficient Market Hypothesis—especially since the "tomorrow equals today" relation follows from the weak form of the EMH as we discuss for in our Random Walk model (Section 2.1.1). Finally, we discuss (i) how especially the number of hidden neurons may result in overfitting, and (ii) how the random weight initialization causes identical network architectures to produce different results. As implemented in our written computer programs, we amend these problems by (i) performing cross-validation and (ii) training a number of identical networks and choosing the one with best validation performance.

- Training and testing Support Vector Machines for predicting the NASDAQ Index prices (SVM regression, Section 6.1.6) and up/down direction movement (SVM classification, Section 6.1.7), we analyse the effects of varying the model parameters, including C, γ, ν, ϵ, kernel functions, etc. For our application, we find that the Radial Basis Function gives the best results. Our analyses and findings also exemplify the common machine learning problem of *overfitting*; the cost parameter C, in particular, is found to capture the trade-off between simplicity and complexity, with large C potentially giving rise to overfitting, in accord with theory and other research. We find that, for a particular input combination, varying γ does not change the results significantly; the most important thing is to change it according to the number of inputs. For ϵ-SVR, small ϵ are found to make the potential overfitting even more pronounced, also in agreement with other research. For ν-SVM, we find that large values ($\nu \sim 0.9$) generally achieves the best balance between underfitting and overfitting, and thus the best generalization ability. In our computer programs we have implemented cross-validation to find the optimal parameters. In regards to predicting the up/down movement direction, we find that SVM classification does not yield better results than SVM regression; the latter is thus the method of choice since it contains additional information and ways to assess prediction performance.

- Comparing our prediction results to the example of predicting a smooth sine wave vs. one with random noise (Section 6.1.6), we argue that the erratic daily stock price fluctuations

may to some extent be regarded as random. This is further supported by the fact that our models often use the current period's value as the prediction for the next period—which is exactly the idea behind our Random Walk model that has close ties to the Efficient Market Hypothesis. Again, our results may thus be regarded as support for the EMH and unpredictability of the markets. Or, it may be that our models are simply not advanced enough, and that the inputs (lagged prices and technical indicators) lack predictive power.

- We have trained and tested Random Forests and analysed the effects of varying the model parameters (Section 6.1.8). In accord with theory, we find for our application that larger forests generally produce better results, but with diminishing gains. We also analyse the number of randomly chosen inputs and the leaf size, and find optimal values; but, since optimal values of these parameters depend on the case at hand, we have implemented an automatic optimization procedure in our program. Moreover, we explore the method's straightforward approach to feature importance, finding that a reduced, simpler model performs at least as good as the full model. However, the predictions still suffer from the recurrent problem of being "shifted" from the actual values, indicating that the model often uses today's price as the prediction for tomorrow.

- In numerous experiments (Section 6.2) we try to reduce the effect of the noisy, erratic nature of daily data by smoothing it. This generally gives much better performances, but also slightly orients our predictions towards trend.

- In Experiment 1 (Section 6.2) we find that our NN, SVM, and RF models perform exceptionally well—not only on the training set but also on the unseen test set, indicating that our measures against overfitting (cross-validation and out-of-bag optimization) have worked. The three methods show have similar performances with only marginal differences, making it hard to point a winner among them. Each model correctly predicts about 75% of the up/down movements, with small deviations from the actual prices, as indicated by the low MSE and MAPE values. Compared to the benchmarks, our models significantly outperform the simple Random Guess ("coin flip") model. The Random Walk ("tomorrow equals today") model on the other hand, achieves a similarly large Hit Rate, indicating that the good performance of our models may be ascribed to the fact that the stock often rose or fell multiple days in a row. However, in terms of the more quantitative MSE and MAPE measures, our models yield significantly lower values the Random Walk benchmark, thus giving a much better precision in the predictions.

- In Experiment 2 (Section 6.2), where we vary the number of lags used of the input variables, we find that including at least one lag of the response (i.e. the most recent stock price) significantly improves performance. Including additional response lags generally improves training performance, but not always the test performance (which is the most important part). For the exogenous input (technical indicators in our case), we find that the number of lags is not as important as that first lag of the response variable. In some cases, including technical indicators does not even improve performance by a significant amount.[2] This indicates that, if technical indicators can or should be used, then they should be so in a more advanced way; e.g. by programming their interpretations and making a trading system that gives buy or sell signals depending on the values of one or more indicators. Regarding the fact that the first response lag is the most important, this is very interesting because our models often use today's price (i.e. the first response lags) as their prediction

[2]We point out that this also depends on how the technical indicator data is processed; for example, in Experiment 2 we only normalized them, whereas in Experiment 4 we show how differencing them yields better performances.

for tomorrow. So, even though we have tried using even more inputs, the models most often determine that the best thing, to some extent, is the "tomorrow equals today" relation. Of course, they do not use this relation exactly (the Random Walk benchmark does, and our models outperform it), but they seem to use part of it sometimes, and it seems that this is the best we are able to do in regards to prediction. So, since this relation is so close to the Random Walk model (and thus the EMH), and since we can't to better prediction-wise, the results may to some extent be regarded as empirical support for the EMH (weak version, at least) and thus the unpredictability of the markets. We emphasize, however, that another possible conclusion is that our prediction models are simply not advanced enough, and that more data, better models, etc. may or may not be able to predict the markets.

- Experiment 3 (Section 6.2) considers different exogenous inputs, and we find that adding additional technical indicators does not necessarily improve performance; it varies with the particular technical indicator under consideration. Thus, if using technical indicators, choosing the right ones is important. We also find that adding lags of the response variable most often improves results, indicating that the price data itself might possess most predictive power. However, it is remarkable to see that using technical indicators alone (without lagged response data) can yield good performances.[3]

- In Experiment 4 (Section 6.2) we find that differencing the exogenous input data yields better results than when using their non-differenced form. This is clear evidence of the importance of proper data handling, pre-processing, and transformation when it comes to machine learning methods; stationary data (as obtained by differencing) is much more suitable than non-stationary data.

- Experiment 5 (Section 6.2) investigates the effect of the amount of training data (how much historical stock data we use to train the models). We find that using too much training data is not good for any of the NN, SVM, and RF models; it does not significantly improve performance (in many cases quite the contrary) but only serves to increase the computation time. For our application, we find that using about 3 years' worth of historical stock data yields the best performance.

- Experiment 6 (Section 6.2) analysed which combinations of technical indicators gave the best performances, and thus which indicators possessed most predictive power. Out of the total seven indicators considered, we find the most prominent ones to be Momentum, RSI, William's % R, and Chaikin Oscillator. The very best exogenous input combination consisted of three technical indicators—and this was the case for all three machine learning methods. And the previous four indicators were exactly the most recurrent ones, and thus the ones found to exhibit most predictive power.

- In Experiment 7 (Section 6.2) we analysed the use of weekly and monthly data (instead of daily data). Our findings indicate that using weekly or monthly data does not relieve the problems with prediction encountered with daily data. This is seen quantitatively from the fact that our models' Hit Rate performances are still not significantly better than the Random Walk benchmark (although our models do still outperform the benchmark in terms of MSE and MAPE). And we see it qualitatively from the plots where the models fail to properly predict the stock price when it makes a sudden significant move in the opposite direction, away from the current trend.

[3]This may contradict the conclusions from the previous bullet on Experiment 2. However, it is important to note in this regard that we differenced the exogenous inputs in Experiment 3, which evidently made these inputs more significant—as we also find in Experiment 4. Thus, technical indicators are not useless; it just all comes down to proper data pre-processing, etc.

- In Experiment 8 (Section 6.2) we tried to use some fundamental data (interest rate, gold and oil price, etc.) as exogenous inputs. Our main finding was that using fundamental data yielded poorer results than using technical data. However, this may largely be ascribed to the humble type and amount of fundamental data used; a full-fledged fundamental analysis would include an abundance of additional data—which is very difficult to come by in the proper type and quality. Fundamental analysis is also often associated with long-term investments, which somewhat goes against our use of daily data; monthly data would be better, but there are not sufficient observations available (the fundamental datasets did not go sufficiently far back in time) to properly train our machine learning methods.[4]

- In Experiment 9 (Section 6.2) we performed predictions for a variety of different financial assets. We find that our NN, SVM, and RF machine learning methods, and our ARMA-GARCH (AG) conditional mean and variance model, all outperform (by far, with Hit Rates upwards of 80%) the simple Random Guess ("coin flip") benchmark. The Random Walk ("tomorrow equals today") benchmark, however, yields a similarly high Hit Rate, which slightly reduces the impressiveness of our models' performances. Our models do achieve much lower MAPE values, indicating that the predictions are still significantly more refined and precise than the benchmark. Comparing the four models to each other, NN, SVM, and RF generally yield better Hit Rates than AG, but roughly similar MAPE values. NN, SVM, and RF often yield very similar performances; however, in terms of computation times, RF is generally the fastest, followed by SVM and then NN.

- In Experiment 10 (Section 6.2) we made predictions using the linear version of our oscillator model (Section 3.4). The results are much the same as for our NN, SVM, RF, and AG models: first, a Hit Rate much better than the simple "coin flip" benchmark, but similar (or slightly better) than the Random Walk ("tomorrow equals today") model; second, MSE and MAPE values much better than the RW benchmark; and third, a performance that is best in periods where the price is trending, and poorest when the price fluctuates erratically back and forth. Still, the good results indicate that there might be some use to modelling stock prices depending on their movement relative to some average, base level (as is the essence of our OM model). And, seeing that we have here used only the simple, linear version of the model, there might be even better results to obtain by using the extended, non-linear version of the model (also described in Section 3.4).

Finally, regarding the predictability of the markets, the fact that we obtain Hit Rates comparable to the RW benchmark means that our models correctly predict tomorrow's price direction when it is the same as today; and that the models most often fail when this is not so (i.e., in the event of a sudden move in the opposite direction—a reversal). The latter part is crucial, since this is something that would otherwise be particularly useful. Failure of the models in this regard thus question their applicability for practical purposes, as well as the predictability of the markets in general. This may be viewed as support for the weak form of the Efficient Market Hypothesis[5] On other hand, the fact that our models outperform the RW/EMH benchmark in some regards (MSE and MAPE), and thus yield more refined and precise predictions, may work to argue in the opposite direction. It all comes down to how performance and predictability is measured[6], and where we put the threshold between failure and success.

[4] We note that the use of fundamental data was really beyond the scope of this thesis, and we save it as a candidate for future research.

[5] The weak form was the one that was linked to our Random Walk benchmark model.

[6] As we will discuss shortly, one of our primary topics for future work is the use of a performance measure more suitable for practical application.

On a final note, the models employed here have only been so advanced, limited in part by time and resources (e.g. machinery and computation power). More advanced models may or may not be more successful at predicting the markets.[7]

Uses for Our Results

Our results can be used for a variety of purposes in both practice and research. First, take our finding that asset returns are not Gaussian—in contrast to many assumptions—and that a better model is provided by e.g. the t distribution. This has important implications especially in risk management. For example, if a researcher wants to simulate asset returns for an experiment, or a practitioner wants to assess the risk of an investment (be it a private investor doing so for himself or a professional doing it for a client), then the assumed Gaussian model will produce wrongful results; although the body of the empirical distribution is adequately modelled by the assumed theoretical distribution, the same is not at all true for the tails. The result is that the simulated returns will contain much fewer extreme events than what is actually observed in reality. Similarly, the assessed risk of an investment will be significantly underestimated, giving the practitioner a flawed view of the actual circumstances and a result that might be inconsistent with his/her investment and risk profile. As we have shown here, the t distribution provides a better model that much more adequately accounts for the larger probability of extreme events.

We have also made numerous model-specific experiments that explore optimal settings and parameters, the models' strengths and possibilities (e.g. easy feature importance and validation for Random Forests), as well as their limits (in particular the overfitting problem). Our results in this regard may help other researchers studying the same field, making them aware of which settings (e.g. training algorithms for Neural Networks, kernel functions for SVMs, etc.) and parameter values might work better than others, perhaps enabling them to save precious time. Similarly, practitioners employing the same methods may be able to optimize their prediction models and consequently improve their trading systems.

The same goes for our numerous additional experiments. Our results in this regard may give other researchers and practitioners some helpful tools and ideas as to what works best when it comes to e.g.: how much historical data to use for training the models, which technical indicators to use as predictors, how many lags/past values to use for the inputs, etc. Our results and discussions may also help with the initial step of which method and model to use—a choice that depends on the user's needs and wishes for e.g. accuracy (where we find that the models, when sufficiently optimized, produce similar results) and speed/computation time (where we find much greater differences between the models).

Future Work

Financial forecasting, and even the subtype of stock market prediction, is a truly massive discipline with numerous different aspects and facets. The limited scope of this study obviously means that we have had to focus on a selection of areas. There are certain parts of the thesis that we would like to extend and analyse in even greater detail, and there are likewise completely new facets that we would like to explore. Our subjects for future work and research are listed below:

[7] Some of our suggestions (to be tried in our future work) for making the models more advanced include using a much wider array of exogenous inputs (e.g. technical and fundamental indicators, etc.) from which to find the optimal combination. One could also employ newer, more advanced versions of the methods which often surface in the present literature. Also, one could use the prediction models in a slightly different way: as part of a trading system for buying and selling at the right time. This renders it impossible to use MSE and Hit Rate as performance measures, but leaves us with a measure that is much more adequate for practical applications, as discussed in one of the bullets in the "Future Work" section below.

- Investigate stock market crashes and their prediction using e.g. log-period power laws and other models inspired by e.g. physics and the analysis and prediction of natural disasters and catastrophes. We touched upon this back in Section 3.2.

- Make more use of our oscillator-inspired model for stock market prediction, both the simple, linear version (as used in e.g. Experiment 10 in Section 6.2) but especially also the extended, non-linear version that had close ties to the Duffing oscillator (as we described in Section 3.4).

- Employ classification models from machine learning for categorizing stocks—e.g. in a portfolio—as "good" or "bad" investments based on a number of criteria. These criteria may be of a fundamental or technical nature; e.g. a company's price, earnings, debt, etc., or the value and interpretation of one or more technical indicators measuring whether the stock e.g. is overbought or oversold. For this purpose one could use e.g. Artificial Neural Networks, Support Vector Machines, and Random Forests, all of which have been put to extensive use here.

- Multi-step ahead prediction and generally forecasting over longer time periods. Here, we have mainly focused on one-step ahead prediction using daily data, i.e. day-to-day prediction. It would be interesting—and perhaps with better results to follow—to explore in more detail the possibility of forecasting the general market direction over longer time horizons. Of course, this requires a downgrading of expectations, since we will be considering the general trend (and not prices). But, with short-term day-to-day prediction seeming so difficult—as our results indicate—this long-term approach may be more plausible. It may also be more adequate from a practical point of view; buying and selling often, as required by day-to-day prediction, incurs massive transaction costs. Investing in the long-term does not.

- Forecasting using tools and knowledge from fundamental analysis. Here, we have mainly used technical indicators. It would be interesting to properly use fundamental data for forecasting, including e.g. company-related data from budgets and balance accounts, or macro-economic indicators for the overall state of the economy. This is related to the previous point on multi-step ahead prediction and long-term forecasting, since fundamental data is generally associated to changes and effects in the long term (several months or years).

- A more practical approach to stock market prediction. In this study, we have approached the problem from a more research-oriented perspective, using performance measures such as MSE and MAPE (average deviation) and the Hit Rate (movement direction accuracy) to assess our models and their predictions. However, a performance measure that may be more adequate for practical application is the *profit*, i.e. how much money the prediction models and trading systems are making. This is really what it all comes down to in business and industry. That the predictions lie close to the actual values, and that the models have a high accuracy, are only secondary; the primary concern (in practice) is money. And this goes for both professionals and private investors alike; low MSEs and MAPEs and high Hit Rates do not necessarily put food on the table—only money in the bank does. It is important to note that good values of our performance measures may very well also yield a good profit—but not necessarily; for the performance measures may tell different stories[8].

[8]For example, we may observe an excellent Hit Rate of almost 100% while at the same time having e.g. a sky-high MSE. As an example, consider a stock price currently at 100. Our prediction for the next period is, say, 125, but the actual price only ends up increasing to 105. The model correctly predicts the increase in price, but it does so with quite a large deviation.

Also, when we consider practical application and profit, it is imperative that we also take into account such things as transaction costs. If one buys and sells very often, transaction costs can significantly reduce one's profit, and even lead to losses. For future work, it would therefore be very interesting—and extremely useful—to consider profit as the performance measure.

- In relation to the previous bullet on a more practical approach, it would also be interesting to use the methods and models in a different way: Instead of performing e.g. day-to-day predictions, we could use the models as part of a trading system. This would entail a deeper, more thorough use of the technical indicators, as we would program their actual interpretations for identifying trends and signals, etc. A method such as one of those used here could then be used to make the final decision of if and/or when to buy or sell. With such a system, we can no longer use performance measures such as the MSE and Hit Rate—only the profit. But from a practical point of view, this is also the most adequate, as discussed above. This approach also pulls us away from the fundamental question of the predictability of the markets, and instead deals with beating the markets. This is a highly interesting distinction—predicting vs. beating the markets—for they are not necessarily the same. Depending on definitions, we argue that "Predict $\implies$ Beat". I.e., if one can predict the markets, then one can also beat[9] them—but not the other way around. The former follows naturally; if one can predict the future prices to a sufficient extent, then one can beat the market. To see the latter, note that one may beat the market via proper timing, i.e. buying and selling at the right times[10]. But buying and selling at the right times is not directly equivalent to predicting the markets (in the sense that they predict the future prices). This is an interesting distinction that we will explore more thoroughly in our future work.

Christian Birch Okkels
May 2014.

[9] By beating, we mean making a profit greater than what a "buy and hold" (buy the stock and keep it) strategy would have earned.

[10] Some of the large investment banks and hedge funds succeed at this, using highly customized specialist trading systems.

Bibliography

[1] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[2] Trevor Hastie, Robert Tibshirani, Jerome Friedman, *The Elements of Statistical Learning*, Springer, 2008.

[3] Rosario N. Mantegna, H. Eugene Stanley, *An Introduction to Econophysics - Correlations and Complexity in Finance*, Cambridge University Press, 2000.

[4] Mark H. Holmes, *Introduction to Numerical Methods in Differential Equations*, Springer, 2007.

[5] J. D. Lambert, D. Lambert, *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*, Wiley, 1991.

[6] C. Man-Chung, W. Chi-Cheong, L. Chi-Chung, *Financial Time Series Forecasting by Neural Network Using Conjugate Gradient Learning Algorithm and Multiple Linear Regression Weight Initialization*, Department of Computing, Hong Kong Polytechnic University.

[7] B. Junyou, *Stock Price Forecasting using PSO-trained Neural Networks*, 2007.

[8] K. Kim, I. Han, *Genetic Algorithms Approach to Feature Discretization in Artificial Neural Networks for the Prediction of Stock Price Index*, Expert Syst. Appl. 19 (2), pp. 125-132, 2000.

[9] C. N. W. Tan, *An Artificial Neural Networks Primer with Financial Applications in Financial Distress Prediction and Foreign Exchange Hybrid Trading System*, School of Information Technology, Bond University.

[10] I. Kaastra, M. Boyd, *Designing a Neural Network for Forecasting Financial and Economic Time Series*, Neurocomputing, 10, pp. 215-236, 1996.

[11] E. Kalyvas, *Using Neural Networks and Genetic Algorithms to Predict Stock Market Returns*, 2001.

[12] C. Siriopoulos, R. N. Markellos, K. Sirlantzis, *Applications of Artificial Neural Networks in Emerging Financial Markets*, 1996.

[13] S. Haykin, *Neural Networks, A Comprehensive Foundation*, Prentice-Hall, Inc., New Jersey, USA, 1994.

[14] J. Yim, *A Comparison of Neural Networks with Time Series Models for Forecasting Returns on a Stock Market*, RMIT Business Working Paper Series, 7, August, 2002.

[15] S. Walczak, *An Empirical Analysis of Data Requirements for Financial Forecasting with Neural Networks*, Journal of Management Information Systems, 17, 4, pp. 203-222, 2001.

[16] A. Abounoori, H. Mohammadali, N. Alikhani, E. Naderi, *Comparative Study of Static and Dynamic Neural Network Models for Nonlinear Time Series Forecasting*, Munich Personal RePEc Archive, 2012.

[17] S. Kulkarni, I. Haidar, *Forecasting Model for Crude Oil Price Using Artificial Neural Networks and Commodity Futures Prices*, 2009.

[18] M. Majumder, A. Hussian, *Forecasting of Indian Stock Market Index Using Artificial Neural Network*.

[19] G. Zhang, B. E. Patuwo, M. Y. Hu, *Forecasting with Artificial Neural Networks: The State of the Art*, International Journal of Forecasting, 14, pp. 35-62, 1998.

[20] M. Riedmiller, H. Braun, *A direct adaptive method for faster backpropagation learning: The Rprop algorithm*, Proceedings of the IEEE International Conference on Neural Networks, pp. 586-591, IEEE Press, 1993.

[21] M. Moller, *A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*, Neural Networks, Vol. 6, pp. 525533, 1993.

[22] P. Gill, W. Murray, M. Wright, *Practical Optimization*, 1981.

[23] M. T. Hagan, M. Menhaj, *Training feed-forward networks with the Marquardt algorithm*, IEEE Transactions on Neural Networks, Vol. 5, No. 6, 1999, pp. 989993, 1994.

[24] L. Cao, F.E.H. Tay, *Financial Forecasting using Support Vector Machines*, Neural Computing and Applications, 10, pp. 184-192, 2001.

[25] K. Kim, *Financial Time Series Forecasting using Support Vector Machines*, Neurocomputing, 55, pp. 307-319, 2003.

[26] W. Huang, Y. Nakamori, S. Wang, *Forecasting Stock Market Movement Direction with Support Vector Machines*, Computers & Operations Research 32, pp. 2513-2522, 2005.

[27] L. Yu, S. Wang, K.K. Lai, *Mining Stock Market Tendency using GA-based Support Vector Machines*, LNCS 3828, pp. 336-345, 2005.

[28] J. Mager, U. Paasche, B. Sick, *Forecasting Financial Time Series with Support Vector Machines Based on Dynamic Kernels*, 2007.

[29] *Training nu-Support Vector Regression: Theory and Algorithms*, Chih-Chung Chang, Chih-Jen Lin, National Taiwan University.

[30] C. J. C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery, 2, 121-167, 1998.

[31] V. N. Vapnik, A. Y. Chervonenkis, *Theory of Pattern Recognition: Statistical Problems of Learning*, 1974.

[32] MATLAB Toolboxes, http://www.mathworks.se/help/index.html, May 2014.

[33] C. Chang, C. Lin, *LIBSVM: A Library for Support Vector Machines*, ACM Transactions on Intelligent Systems and Technology, pp. 1-27, 2011.

[34] http://www.svms.org/anns.html, *Support Vector Machines*, May, 2014.

[35] L. Breiman, *Random Forests*, 2001.

[36] James D. Hamilton, *Time Series Analysis*, Princeton University Press, 1994.

[37] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, *Time Series Analysis - Forecasting and Control*, Prentice-Hall, 1994.

[38] R. H. Shumway, D. S. Stoffer, *Time Series Analysis and Its Applications*, Springer, 3rd edition, 2011.

[39] S. Bisgaard, M. Kulahci, *Time Series and Forecasting by Example*, Wiley, 2011.

[40] R. S. Tsay, *Analysis of Financial Time Series*, Wiley, 2nd edition, 2005.

[41] G. E. P. Box, D. A. Pierce, *Distribution of Residual Autocorrelations in Autoregressive-Integrated Moving Average Time Series Models*, Journal of the American Statistical Association, 65, pp. 15091526, 1970.

[42] G. Ljung, G. E. P. Box, *On a Measure of Lack of Fit in Time Series Models*, Biometrika 66, pp. 6772, 1978.

[43] Robert F. Engle, *Autoregressive Conditional Heteroscedasticity with Estimates of Variance of United Kingdom Inflation*, Econometrica, 50, pp. 987-1008, 1982.

[44] R. F. Engle, V. K. Ng, *Measuring and Testing the Impact of News on Volatility*, Journal of Finance, 48, pp. 1749-1778, 1991.

[45] T. Bollerslev, *Generalized Autoregressive Conditional Heteroskedasticity*, Journal of Econometrics, 31, pp. 307-327, 1986.

[46] C. B. Okkels, *The Distribution of Stock Market Returns - An Analysis of Distribution Models for the Description of Financial Asset Return Data* University of Copenhagen, 2013.

[47] Stefan Zemke, *On Developing a Financial Prediction System: Pitfalls and Possibilities*, Stockholm University and Royal Institute of Technology, Department of Computer and System Sciences, Sweden.

[48] F. Black, M. Scholes, *The Pricing of Options and Corporate Liabilities*, Journal of Political Economy, 81, 3, pp. 637-654, 1973.

[49] W. G. Tomek, S. F. Querin, *Random Processes in Prices and Technical Analysis*, Journal of Futures Markets, 4, pp. 15-23, 1984.

[50] D. J. Acheson, *Elementary Fluid Dynamics*, Oxford Applied Mathematics and Computing Science Series, Oxford University Press, 1990.

[51] John D. Cox, *Storm Watchers*, John Wiley & Sons, Inc., pp. 222-224, 2002.

[52] Edward N. Lorenz, *Deterministic Non-periodic Flow*, Journal of the Atmospheric Sciences 20, 2, pp. 130-141, 1963.

[53] P. Chen, *A Random Walk or Color Chaos on the Stock Market? Time-Frequency Analysis of S&P Indexes*, Studies in Nonlinear Dynamics & Econometrics, 1, 2, 2996.

[54] David A. Hsieh, *Chaos and Nonlinear Dynamics: Application to Financial Markets*, 1990.

[55] W. Brock, C. Sayers, *Is The Business Cycle Characterized by Deterministic Chaos?*, Journal of Monetary Economics, 22, pp. 71-90, 1988.

[56] NASA, The Sunspot Cycle, http://solarscience.msfc.nasa.gov/SunspotCycle.shtml, May 2014.

[57] K. Petrovay, *Solar Cycle Prediction*, Living Rev. Solar Physics, 7, 6, 2010.

[58] David H. Hathaway, Robert M. Wilson, *Geomagnetic activity indicates large amplitude for sunspot cycle 24*, Geophysical Research Letters, 33, 18, September 2006.

[59] G. P. Zhang, *Time series forecasting using a hybrid ARIMA and neural network model*, Neurocomputing, 50, pp. 159-175, January 2003.

[60] K. H. Schatten et al. *Using Dynamo Theory to predict the sunspot number during Solar Cycle 21*, Geophysical Research Letters, 5, 5, pp. 411-414, May 1978.

[61] G. Zhang, E. P. Patuwo, M. Y. Hu, *Forecasting with artificial neural networks: The state of the art*, International Journal of Forecasting, 14 1, pp. 35-62, March 1998.

[62] R. A. Calvo, H. A. Ceccato, R. D. Piacentini, *Neural network prediction of solar activity*, Astrophysical Journal, 1, 444, pp. 916-921, 1995.

[63] R. Qahwaji, T. Colak, *Automatic Short-Term Solar Flare Prediction Using Machine Learning and Sunspot Associations*, Solar Physics, 241, 1, pp. 195-211, March 2007.

[64] Jing-Xin Xie, et al. *A hybrid adaptive time-delay neural network model for multi-step-ahead prediction of sunspot activity*, International Journal of Environment and Pollution, 28, 3-4, pp. 364-381, 2006.

[65] National Research Council, *Advancing the Science of Climate Change*, Washington, DC: The National Academies Press, 2010.

[66] J. B. Drake, *Predicting Climate Change*, http://web.ornl.gov/info/ornlreview/rev28_2/text/cli.htm, May 2014.

[67] Robert J. Geller, *Earthquake prediction: a critical review*, Geophysical Journal International, 131, 3, pp. 425-450, 1997.

[68] R. Console, *Testing earthquake forecast hypotheses*, Tectonophysics, 338, 34, pp. 261268, 2001.

[69] Yan Y. Kagan, *Are earthquakes predictable?*, Geophysical Journal International, 131, 3, pp. 505-525, 1997.

[70] International Commission on Earthquake Forecasting for Civil Protection, *Operational Earthquake Forecasting: State of Knowledge and Guidelines for Utilization* Annals of Geophysics, 54, 4, pp. 315-391, 2011.

[71] Lynn R. Sykes, Bruce E. Shaw, Christopher H. Scholz, Rethinking Earthquake Prediction Pure and Applied Geophysics 155, pp. 207-232, 1999.

[72] Geophysicist predicts new bursting financial bubbles, http://ing.dk/artikel/geofysiker-forudser-nye-bristende-finansbobler-167320 (Danish), May 2014.

[73] Didier Sornette, *Why Stock Markets Crash: Critical Events in Complex Financial Systems*, Princeton University Press, 2004.

[74] A. Johansen, D. Sornette, O. Ledoit, *Predicting Financial Crashes Using Discrete Scale Invariance*, Journal of Risk, 1, 4, pp. 5-32, 1999.

[75] A. Johansen, *Characterization of large price variations in financial markets*, Proceedings of International Econophysics Conference, 2002.

[76] W. Zhou, D. Sornette, *A case study of speculative financial bubbles in the South African stock market 2003-2006*, Physica A, 388, 6, pp. 869-880, 2009.

[77] D. Sornette, A. Johansen, *Significance of log-periodic precursors to financial crashes*, Quantitative Finance, 1, 4, pp. 452-471, 2001.

[78] Z. Jiang et al. *Bubble Diagnosis and Prediction of the 2005-2007 and 2008-2009 Chinese stock market bubbles*, Journal of Economic Behavior & Organization, 74, 3, pp. 149-162, 2010.

[79] J. M. T. Thompson, H. B. Stewart, *Nonlinear Dynamics and Chaos*, John Wiley & Sons, 2nd edition, 2002.

[80] E. Ott, *Chaos in Dynamical Systems*, 2nd edition, Cambridge University Press, 2002.

[81] Y. Ueda, *The Road to Chaos*, Aerial Press, 1992.

[82] Y. Ueda, *Randomly transitional phenomena in the system governed by Duffing's equation*, Journal of Statistical Physics, 20, pp. 181-196, 1979.

[83] P. Holmes, *A nonlinear oscillator with a strange attractor*, Philosophical Transactions of the Royal Society A, 292, pp. 419-448, 1979.

[84] A Wave Theory, http://www.financialwisdomforum.org/gummy-stuff/Wave-Theory-2.htm Financial Wisdom Forum, May 2014.

[85] Martingale, http://mathworld.wolfram.com/Martingale.html, Wolfram Mathworld, May 2014.

[86] Yahoo! Finance, http://finance.yahoo.com/, Yahoo!, May 2014.

[87] Historical Stock Data Downloader MATLAB program, http://www.mathworks.com/matlabcentral/fileexchange/18458-historical-stock-data-downloader, May 2014.

[88] Joseph de la Vega, *Confusin de Confusiones*, 1688.

[89] Steve Nison, *Japanese Candlestick Charting Techniques*, 1991.

[90] R. W. Schabacker, *Stock Market Theory and Practice*, 1930.

[91] R. W. Schabacker, *Technical Analysis and Stock Market Profits: A Course in Forecasting*, 1932.

[92] A. J. Frost, R. R. Prechter Jr., *Elliot Wave Principle*, 2005.

[93] Steven Achelis, *Technical Analysis from A to Z*, McGraw-Hill, 2nd edition, 2000.

[94] Harry V. Roberts, *Stock-Market Patterns and Financial Analysis: Methodological Suggestions*, The Journal of Finance, 14, 1, pp. 1-10, 1959.

[95] J. Stuart Hunter, *The Exponentially Weighted Moving Average*, Journal of Quality Technology, 18, 4, pp. 203-210, 1986.

[96] *NIST/SEMATECH e-Handbook of Statistical Methods*, http://www.itl.nist.gov/div898/handbook/, May 2013.

[97] J. Welles Wilder, *New Concepts in Technical Trading Systems*, 1978.

[98] G. Quong, A. Soudack, *Volume-weighted RSI: Money Flow*, Stocks and Commodities, 7, 3, pp. 76-77.

[99] *List of Technical Analysis Trading Indicators*, http://www.asiapacfinance.com/trading-strategies/technicalindicators, May 2013.

[100] *Technical Indicators and Overlays*, http://stockcharts.com/school/doku.php?id=chart_school:technical_indicato May 2013.

[101] John J. Murphy, *Technical Analysis of the Financial Markets*, New York Institute of Finance, 1999.

Appendix A

Technical Indicators

This appendix introduces a wide array of indicators used for technical analysis of financial assets. A technical indicator is merely a time series of data points derived by applying a particular formula to one or more types of data[1] for a given asset. We describe the usage and calculation, in terms of both words and formulae, of a variety of indicators. Moreover, we discuss their interpretations and how they can be used to e.g. find trends in stock prices, and/or determine peaks and bottoms, etc. An incredibly vast amount of technical indicators exist, varying in both complexity and applicability. In general, they can be divided into the following classes.

Trend Indicators reflect the tendency in price movements; the price is either moving up, down, or sideways. The indicators belonging to this class help define prevailing directions, or *trends*, in the stock price, often by smoothing data over a certain period of time.

Momentum Indicators measure the velocity and magnitude of directional price movements[2], thereby showing the strength or weakness of a trend as it progresses over a given period of time.

Volatility Indicators show the size and magnitude of price fluctuations, providing an insight into the level of market activity.

Volume Indicators are used to gauge the interest of investors in the asset, by including in some way the amount of assets traded over some time period.

Cycle Indicators are used in the analysis of potentially repeating patterns (cycles), which are dedicated to certain market or company events such as seasons (e.g. Christmas sales for retail stores, etc.).

Other Indicators is a class comprising about a dozen indicators which do not fit anywhere else.

It is beyond the scope of this study to give an exhaustive discussion of each and every technical indicator.[3] In fact, using too many indicators easily leads to confusion, potentially contrasting signals and forecasts, and in turn suboptimal prediction model performance. Also, in regards to

[1] These data types include any combination of the opening, high, low, or closing price over a period of time, as well as the trading volume.

[2] Evidently, as the name suggests, momentum indicators can be viewed as the financial pendant to momentum in physics—i.e. velocity times mass: $\vec{p} = m\vec{v}$.

[3] For an almost complete list, we refer to e.g. [93] or [99].

using technical indicators for machine learning methods, the use of too many indicators might result in too many inputs, making the models prone to overfitting, which in turn leads to bad performances on unseen data.[4] Therefore, we have chosen a subset—in which the above classes of technical indicators are well represented—deemed to be sufficiently representative and applicable for this analysis. The particular selection of indicators is inspired by the review of domain experts and prior research in input feature discretization [8] and stock market prediction [25], [27].

The following subsections give a description and formalization of some selected indicators as well as a discussion of their interpretations and use for forecasting. For additional indicators, we refer to e.g. [93].

A.1 Simple Moving Average

Moving averages are a type of "smoothing" method for reducing, or canceling, the random variation inherent in data taken over time. When applied properly, this technique reveals more clearly the underlying trend, seasonal, and cyclic components in the data.

In financial applications an n-period *simple moving average* (SMA) is the unweighted mean of the previous n data points. Letting the time series be denoted by $\{y_t\}$, the formula is

$$\mathrm{SMA}_{t,n} = \frac{y_{t-(n-1)} + \cdots + y_{t-1} + y_t}{n} = \frac{1}{n}\sum_{i=0}^{n-1} y_{t-i} \tag{A.1}$$

As an example, the 3-period simple moving average is

$$\mathrm{SMA}_{t,3} = \frac{y_{t-2} + y_{t-1} + y_t}{3} \tag{A.2}$$

The moving average is itself a time series, and when calculating successive values of it, a new data point enters the sum while an old one drops out. Consequently, a full summation for each element is unnecessary, and the simple moving average can be obtained recursively by

$$\mathrm{SMA}_{t,n} = \mathrm{SMA}_{t-1,n} - \frac{y_{t-n}}{n} + \frac{y_t}{n} \tag{A.3}$$

It is important to note that the simple moving average at time t is calculated using only past data. For some applications it is advantageous to avoid this shift by computing the *central moving average* (CMA):

$$\mathrm{CMA}_{t,n} = \frac{y_{t-\frac{n-1}{2}} + y_{t-\frac{n-1}{2}+1} + \cdots + y_t + \cdots + y_{t+\frac{n-1}{2}-1} + y_{t+\frac{n-1}{2}}}{n} = \frac{1}{n}\sum_{i=-\frac{n-1}{2}}^{\frac{n-1}{2}} y_{t-i}, \tag{A.4}$$

where n is uneven.[5] Compared to the 3-period SMA in Eq. (A.2) above, the 3-period CMA is

$$\mathrm{CMA}_{t,3} = \frac{y_{t-1} + y_t + y_{t+1}}{3} \tag{A.5}$$

Now, the central moving average for the present time cannot be computed, as it evidently requires data points in the future. However, the CMA can be interesting when analyzing past data.

[4] An explanation of this concept is given in Chapter 5, where we give a detailed description and discussion of selected machine learning methods and their specific and general merits and drawbacks.

[5] Even n can be included by "smoothing the smoothed values," as shown in e.g. [93].

A.2 Weighted Moving Average

One of the shortcomings of the simple moving average above is that it lends equal weight to all data points. This might not be preferable in real applications, seeing that past data points are often deemed to gradually lose their influence. Consequently, data points further in the past should be ascribed smaller and smaller weights. This can be achieved by using the *weighted moving average* (WMA) defined by

$$\mathrm{WMA}_{t,n} = \frac{\sum_{i=1}^{n} w_i y_{t-i+1}}{\sum_{i=1}^{n} w_i} \tag{A.6}$$

The weighting scheme is such that the weights decrease by one for each previous data point, resulting in the formula

$$\mathrm{WMA}_{t,n} = \frac{ny_t + (n-1)y_{t-1} + \cdots + (n-(n-1))y_{t-n+1}}{n + (n-1) + \cdots + (n-(n-1))} = \frac{\sum_{i=1}^{n}(n-i+1)y_{t-i+1}}{\sum_{i=1}^{n}(n-i+1)}, \tag{A.7}$$

from which the weights can be read off as

$$w_i = n - i + 1, \quad i = 1..n \tag{A.8}$$

Evidently, this results in the older data points (with higher i) having smaller weights.

A.3 Exponential Moving Averages

Exponential smoothing schemes are very popular for producing smoothed time series. They are much resemblant to the weighted moving average above—in fact, they are just a certain type of the WMA—in that recent observations are given more weight than older observations. The distinctive feature of the exponential moving average (EMA) is that it assigns *exponentially decreasing* weights as the observations get older. In the following we go through three exponential smoothing schemes of increasing complexity.

A.3.1 Single Exponential Smoothing

In **single exponential smoothing** the n-period EMA for the time series $\{y_t\}$ is calculated recursively by[6]

$$\mathrm{EMA}_{2,n} = y_1 \tag{A.9}$$

$$\mathrm{EMA}_{t,n} = \alpha y_{t-1} + (1-\alpha)\mathrm{EMA}_{t-1,n}, \quad 3 \le t \le n. \tag{A.10}$$

By substituting the expressions for $\mathrm{EMA}_{t-1,n}$ until the initial value $\mathrm{EMA}_{2,n}$ is reached, Eq. (A.10) can be expanded to the summation formula

$$\mathrm{EMA}_{t,n} = \alpha \sum_{i=1}^{t-2} (1-\alpha)^{i-1} y_{t-i} + (1-\alpha)^{t-2} S_2, \quad 2 \le t \le n. \tag{A.11}$$

[6]This formulation follows [95]. An alternative approach [94] replaces y_{t-1} with y_t in Eq. (A.10).

Here, $0 \leq \alpha \leq 1$ is the *smoothing parameter*, representing the degree of weight decrease. A high value of α (close to 1) makes the dampening quick, i.e. discounts older observations faster, whereas low values (close to 0) make the dampening slow.

Now, the question quickly arises of which α to use. Finding the best parameter value is often done by means of a search method, in which one chooses the value of α that minimizes some choice of error measure (e.g. the mean-squared error). Alternatively, α is often expressed in terms of the choice of time periods n as $\alpha = \frac{2}{n+1}$.

Moreover, $t = 1$ is here taken to be the first time period. Evidently, there is no $\mathrm{EMA}_{1,n}$, and the smoothed series starts at the second time period. There is also some degree of freedom in this initial value of the EMA. Setting $\mathrm{EMA}_{2,n} = y_1$ as in Eq. (A.9) is just one method of initialization. Another possibility is to start the EMA at a later time period and choose its initial value to be the average of the first few observations. The prominence of the effect of the initialization on the resultant EMA depends on α; as higher values of α discount older observations faster, and vice versa, the choice of the initial EMA value becomes relatively more important for small α than for large α.

Forecasting with this exponential smoothing scheme is done via the formula

$$\mathrm{EMA}_{t+1,n} = \alpha y_t + (1 - \alpha)\mathrm{EMA}_{t,n}, \tag{A.12}$$

which can be written as

$$\mathrm{EMA}_{t+1,n} = \mathrm{EMA}_{t,n} + \alpha E_t, \tag{A.13}$$

where $E_t = y_t - \mathrm{EMA}_{t,n}$ is the forecast error, i.e. the difference between the actual value and the forecast, for time period t.

Forecasting further into the future—for time periods at which the data is yet unknown—can be done via *bootstrapping*. Here, one chooses the most recent data point, denoted y_{recent}, and keeps it constant in the modified formula

$$\mathrm{EMA}_{t+1,n} = \alpha y_{\mathrm{recent}} + (1 - \alpha)\mathrm{EMA}_{t,n} \tag{A.14}$$

Like the standard weighted moving average in the previous subsection, the exponential moving average weighs recent prices more heavily than past prices. Compared to the simple, unweighted moving average, this gives the advantage of a quicker response to fluctuations. On the other hand, this sensitivity can also be a disadvantage and lead to false forecasting signals.[7]

A.3.2 Double Exponential Smoothing

As observed in e.g. [96], single smoothing does not excel in following the data when a trend is present. This shortcoming can be reduced through the introduction of a second equation with a second parameter, γ, which must be chosen in conjunction with α. This is called **double exponential smoothing** and the two associated equations are[8]

$$\mathrm{EMA}_{t,n} = \alpha y_t + (1 - \alpha)(\mathrm{EMA}_{\mathrm{t-1,n}} + b_{t-1}), \qquad 0 \leq \alpha \leq 1 \tag{A.15}$$

$$b_t = \gamma(\mathrm{EMA}_{\mathrm{t,n}} - \mathrm{EMA}_{\mathrm{t-1,n}}) + (1 - \gamma)b_{t-1}, \qquad 0 \leq \gamma \leq 1 \tag{A.16}$$

The first equation works as an overall smoothing, while the latter is a kind of trend smoothing.

Again, there are a variety of ways to select initial values of $\mathrm{EMA}_{\mathrm{t,n}}$ and b_t. In general, $\mathrm{EMA}_{1,\mathrm{n}}$ is set to y_1.[9] Some common choices for b_1 are: $b_1 = y_2 - y_1$, $b_1 = \frac{(y_2-y_1)+(y_3-y_2)+(y_4-y_3)}{3} = \frac{y_4-y_1}{3}$,

[7] In technical analysis of financial assets, these false signals are known as so-called *whipsaws*.

[8] In double exponential smoothing, the current value y_t is used instead of y_{t-1} to calculate the smoothed series.

[9] The EMA series starts at $t = 1$ for double smoothing.

or in general $b_1 = \frac{y_n - y_1}{n-1}$. As for α and γ, these parameters can again be obtained via optimization techniques.

For double exponential smoothing, the forecast F_{t+m} for m time periods ahead is given by

$$F_{t+m} = \mathrm{EMA}_{t,n} + mb_t \tag{A.17}$$

Double exponential smoothing fixes the shortcoming of single exponential smoothing when trends are present in the data. However, if the data show both trend *and* so-called seasonality, or periodicity, then double smoothing has its own shortcomings.

A.3.3 Triple Exponential Smoothing

Reducing the lacks of double exponential smoothing in the case of trend and seasonality can be achieved by introducing a third equation—and a third parameter. We are now dealing with **triple exponential smoothing**. The resulting set of equations used to obtain the smoothed series is also known as the **Holt-Winters method**:

$$\mathrm{EMA}_{t,n} = \alpha \frac{y_t}{I_{t-L}} + (1-\alpha)(\mathrm{EMA}_{\mathrm{t-1,n}} + b_{t-1}), \qquad 0 \le \alpha \le 1 \tag{A.18}$$

$$b_t = \gamma(\mathrm{EMA}_{\mathrm{t,n}} - \mathrm{EMA}_{\mathrm{t-1,n}}) + (1-\gamma)b_{t-1}, \qquad 0 \le \gamma \le 1 \tag{A.19}$$

$$I_t = \beta \frac{y_t}{\mathrm{EMA}_{t,n}} + (1-\beta)I_{t-L}, \qquad 0 \le \beta \le 1 \tag{A.20}$$

The first and second equations are again the overall and trend smoothing, respectively, while the third equation governs the seasonal smoothing.

The forecasting formula for m time periods ahead is in this case given by

$$F_{t+m} = (\mathrm{EMA}_{t,n} + mb_t) I_{t-L+m} \tag{A.21}$$

Initializing the Holt-Winters method requires at least one complete season, or cycle, of data to determine the initial estimates of the seasonal indices I_{t-L}. Here, L denotes the number of periods comprising a complete season. Commonly, one uses two complete seasons, i.e. $2L$ periods.

The general formula for the initialization b of the trend series b_t is

$$b = \frac{1}{L}\left(\frac{y_{L+1} - y_1}{L} + \frac{y_{L+2} - y_2}{L} + \cdots + \frac{y_{L+L} - y_L}{L}\right) \tag{A.22}$$

Setting the initial estimates for the seasonal indices I_i for $i = 1, \ldots L$ is somewhat more involved. Letting N be the number of complete cycles in the data, then

$$I_i = \frac{1}{N}\sum_{j=1}^{N} \frac{y_{L(j-1)+i}}{A_j}, \quad i = 1, \ldots L, \tag{A.23}$$

where

$$A_j = \frac{\sum_{i=1}^{L} y_{L(j-1)+i}}{L}, \quad j = 1, \ldots N. \tag{A.24}$$

Evidently, A_j is the average value of y in the jth cycle of the data.

A.4 Moving Average Convergence Divergence

As the name suggests, the Moving Average Convergence Divergence (MACD) indicator is closely connected to the moving average described above. The MACD is the difference between two exponential moving averages of the stock price; a "fast", or short-period (i.e. low n) EMA and a "slow", or long-period (large n) EMA. Letting p_t denote the stock price, the MACD (a time series itself) is given by the simple formula

$$\text{MACD}_t = \text{EMA}_{t,n_1}(p_t) - \text{EMA}_{t,n_2}(p_t), \quad n_1 < n_2. \tag{A.25}$$

The time series computed from this expression is also called the "MACD line". To help facilitate interpretations and establish trading signals, an exponential moving average of the MACD is also used, this being called the "signal line" and given by: $\text{MACD}_t^{\text{signal}} = \text{EMA}_{t,n}(\text{MACD}_t)$.

Furthermore, a bar chart of the difference between the MACD line and the signal line is also sometimes included.

Since moving averages are used to calculate the MACD, the freedom of choice related to the smoothing period n carries over to the case of the MACD indicator. Evidently, there are three smoothing periods to choose in this case; n_1 and n_2 in the computation of the MACD line, and a third n for the signal line. Most often, one uses data quoted on a daily basis and sets $n_1 = 12$days and $n_2 = 26$days. As for the moving average of the MACD itself, i.e. the signal line, the most common value to use is $n = 9$days. In order to easily distinguish between MACDs using different smoothing periods, one can use the notation $\text{MACD}(n_1, n_2, n)$.

Through its definition as the difference between exponential moving averages of different length, the MACD line gauges changes in the trend of a stock. Then, comparing differences between this line (the MACD line) to an average (the signal line), one can identify subtle shifts in the strength and direction of an asset's trend.

As for interpretations, the MACD can generate several trading signals.[10] First, a crossing of the MACD line through zero (i.e. when the two EMAs in (A.25) are equal) provides evidence of a change in the direction of a trend; an upward move from negative to positive MACD is considered a so-called "bullish" signal (indicating increasing stock prices), while a downward move from positive to negative values is a "bearish" signal (indicating decreasing prices). Evidently, this signal only includes the MACD line. Another cue, which includes also the signal line, occurs when the MACD line crosses the signal line; if it does in an upwards (downwards) fashion, i.e. with a positive (negative) slope, then a buy (sell) signal is generated. Respectively, these crossovers indicate that the trend in a stock is about to accelerate in the direction of the crossover.

Finally, like any forecasting algorithm, the MACD can generate false signals. For example, it could be that the MACD line crossed up through the signal line—which would generate a buy signal—but the stock price suddenly declined. A prudent strategy to filter out such false signals would be to only buy the stock if the MACD line crosses above the signal line and remains above it for a certain period of time. This reduces the probability of false signals, but does not eliminate them. Consequently, it also decreases the potential profit.

A.5 Rate of Change

The Rate of Change (ROC) is a momentum oscillator that measures the percent change in a security's price over a given number of time periods, as represented by the formula:

$$\text{ROC}_t = \frac{p_t - p_{t-n}}{p_{t-n}} * 100 \tag{A.26}$$

[10] Here, the trading signals are described in words; however, they can be implemented and coded in a computer program, as is also done in this study.

Evidently, the ROC value at time t is just how much the price has changed in percent since n periods ago. A common value is $n = 14$, but this should be modified to suit one's preferences and time horizon.

A.6 Momentum

The Momentum (MOM) is a rate of change indicator designed to identify the speed or strength of a price movement. The current value is just the difference between current price and the price n periods ago:

$$\mathrm{MOM}_t = p_t - p_{t-n} \tag{A.27}$$

Values above (below) zero clearly indicate that the price has upwards (downwards) momentum. The most common choice for the number of periods is $n = 12$.

A.7 Williams' %R

William's %R is a momentum indicator used to measure overbought/oversold levels. It oscillates in the interval $[-100, 0]$, as evident by the formula:

$$\mathrm{WR}_{t,n} = -100 * \frac{\max\left\{p_i^{high}\right\}_{t-n+1}^{t} - p_t}{\max\left\{p_i^{high}\right\}_{t-n+1}^{t} - \min\left\{p_i^{low}\right\}_{t-n+1}^{t}} \tag{A.28}$$

Some explanation might be needed to understand this formula. Firstly, as with many other indicators, one should choose the number of periods, n, to consider. Then, $\max\left\{p_i^{high}\right\}_{t-n+1}^{t}$ denotes the maximum value of the high price—or the so-called "highest high"—over the past n periods. Similarly, $\min\left\{p_i^{low}\right\}_{t-n+1}^{t}$ is the "lowest low" over the last n periods. As usual, p_t is simply the closing price for trading period t.

Regarding interpretation, readings of Williams' %R in the approximate range of -80% to -100% indicate that the security is oversold, while readings around 0% to -20% suggest that it is overbought.

A.8 Accumulation/Distribution Oscillator

The Accumulation/Distribution Oscillator (ADO) uses a variation of the Relative Strength Index (section A.9) to define an asset's buying and selling power. The ADO measures the implied direction of each period's trading and is given by

$$\mathrm{ADO}_t = \frac{\left(p_t^{\text{high}} - p_t^{\text{open}}\right) + \left(p_t^{\text{close}} - p_t^{\text{low}}\right)}{2 * \left(p_t^{\text{high}} - p_t^{\text{low}}\right)} * 100, \tag{A.29}$$

where the superscripts refer the highest, lowest, opening, and closing prices for the particular time period t.

A maximum value of 100 is reached when the asset opens at the price that will turn out to be the lowest and closes at the price that will turn out to be the highest. The minimum value of 0 is reached if the converse is true. An upward or downward turn in the ADO, from one time period to the next, can indicate a change in trend before the trend change happens.

When working with the Accumulation/Distribution Oscillator, one also considers the signal line, which is a moving average of the ADO. This smooths out the ADO's sometimes noisy appearance to better determine when the security, on average, is overbought and oversold, and thus when a trend change is more likely to occur.

A.9 Relative Strength Index

The Relative Strength Index (RSI) is a momentum indicator, measuring the velocity and magnitude of directional price movements, that is used to track the current and historical strength or weakness of a stock or market based on the closing prices of a recent trading period.

The calculation of the RSI proceeds as follows. First, for each trading period, an upward change U or downward change D is calculated from the closing prices p_t and p_{t-1} according to[11]

$$U = \max(p_t - p_{t-1}, 0) = \begin{cases} p_t - p_{t-1}, & \text{if } p_t \geq p_{t-1}, \\ 0, & \text{otherwise,} \end{cases} \tag{A.30}$$

and

$$D = \max(p_{t-1} - p_t, 0) = \begin{cases} p_{t-1} - p_t, & \text{if } p_{t-1} \geq p_t, \\ 0, & \text{otherwise.} \end{cases} \tag{A.31}$$

The average U and D are then calculated using an n-period exponential moving average, which was explained in Section A.3 above. The ratio of these averages is the *relative strength factor*:[12]

$$RS = \frac{\mathrm{EMA}_n(U)}{\mathrm{EMA}_n(D)} \tag{A.32}$$

This relative strength factor is then converted to a relative strength index that oscillates between 0 and 100:

$$RSI = 100 - \frac{100}{1 + RS} \tag{A.33}$$

Now, inherent in moving averages and smoothing methods is a degree of freedom in the choice of smoothing period n. As the RSI utilizes the exponential moving average, this freedom of choice carries over to the case at hand. The most common smoothing period to use for the RSI is $n = 14$, which is also the value recommended by its developer, J. Welles Wilder, [97].

There are several interpretations related to the RSI. First, the index is used as an indicator of tops and bottoms of stock prices. This interpretation is linked to the principle of overbought and oversold conditions; when a stock price moves up (down) very rapidly, the asset is at some point considered overbought (oversold). When this overbought or oversold level is reached, a reaction or reversal is deemed to be imminent. The corresponding RSI values are very commonly 70 and 30. Thus, RSI readings greater than 70 are interpreted as the stock being overbought, whereas RSI values lower than 30 indicate that the stock is oversold. The range between 30 and 70 is considered a "neutral" zone, with and RSI of 50 being a sign of no trend at all. The overbought and oversold interpretation leads to the following trading signals; a *buy* signal occurs when the RSI crosses above the 30 level (i.e. with a positive slope), while a *sell* signal occurs when the RSI crosses below the 70 (i.e. with a negative slope).[13]

[11] Note that the downward change D is also positive with this notation.

[12] In the diverging case of $\mathrm{EMA}(U)_n = 0$, RS is defined as 100.

[13] For other—sometimes more exotic and advanced—interpretations of the RSI, we refer to e.g. [97] or the numerous resources on technical analysis available on the Internet.

A.10 Money Flow Index

Like the MACD and RSI, The Money Flow Index (MFI) is an oscillator, meaning that its value fluctuates between a lower and upper bound—here 0 and 100, respectively. Known also as the volume-weighted RSI, the MFI uses both the stock price and the volume to measure buying and selling pressure.

The calculation of the MFI starts off with the *money flow*

$$\mathrm{MF}_t = \frac{p_t^{\mathrm{high}} + p_t^{\mathrm{low}} + p_t}{3} V_t, \tag{A.34}$$

where V_t denotes the volume, i.e. the number of assets traded, for the given time period. p_t^{high}, p_t^{low}, and p_t are the highest, lowest, and closing price, respectively, for the particular time period. The fraction $\frac{p_t^{\mathrm{high}}+p_t^{\mathrm{low}}+p_t}{3}$ is called the *typical price*.

A distinction must be made between positive and negative money flow. When the typical price rises (declines) from one period to the next, money flow is said to be positive (negative). Letting $\tilde{p}_t = \frac{p_t^{\mathrm{high}}+p_t^{\mathrm{low}}+p_t}{3}$ be the typical price, the positive and negative money flow are defined, respectively, by

$$\mathrm{MF}_t^+ = \begin{cases} \tilde{p}_t V_t, & \tilde{p}_t > \tilde{p}_{t-1}, \\ 0, & \tilde{p}_t \leq \tilde{p}_{t-1}, \end{cases} \tag{A.35}$$

and

$$\mathrm{MF}_t^- = \begin{cases} 0, & \tilde{p}_t \geq \tilde{p}_{t-1}, \\ -\tilde{p}_t V_t, & \tilde{p}_t < \tilde{p}_{t-1}. \end{cases} \tag{A.36}$$

Next, one must consider the *money flow ratio*. This is the ratio between the sum of positive money flow to the sum of negative money flow, with the sums taken over a given number of time periods n. The n-period money flow ratio is thus given by

$$\mathrm{MR}_{t,n} = \frac{\sum_{i=1}^{n} \mathrm{MF}_{t-i+1}^+}{\sum_{i=1}^{n} \mathrm{MF}_{t-i+1}^-} \tag{A.37}$$

Evidently, to compute the money flow ratio, one simply adds up all the positive money flows for the n periods and divides them by the sum of negative money flows.

Creating an oscillator ranging from 0 to 100 is then done in the final formula for the n-period money flow index:

$$\mathrm{MFI}_{t,n} = 100 - \frac{100}{1 + \mathrm{MR}_{t,n}} \tag{A.38}$$

Again, the choice presents itself of the number of recent time periods n to use. In technical analysis it is very common to use data quoted on a daily basis and set $n = 14$ days, which is also the value recommended by the creators of the MFI [98].

In regards to the interpretation of the MFI, values greater than 80 are generally considered as an indication that the stock is overbought—indicating a possible imminent decline and thus inducing a sell signal—while a value of 20 is taken to be an indication that the stock is oversold and that an increase in price is likely to follow, thus providing a buy signal. To reduce the risk of false signals, these MFI values should be instead regarded as caution levels, and for truly overbought and oversold levels one should instead use 90 and 10, respectively. This interpretation

and the corresponding trading signals can rather easily be implemented in a computer program, as is done in this thesis.

As with many other technical indicators, an interesting case also occurs when the stock price makes a new rally high but the MFI is less than its previous level. The interpretation of this divergence scenario is a weak advance of the stock price—one that is likely to reverse—thereby indicating a possible imminent decline. Of course, the opposite case also exists; if the stock price is decreasing, continuously reaching new, lower levels, all the while the MFI increases, then a signal is generated of the stock's downward trend about to reverse.

A.11 Commodity Channel Index

The Commodity Channel Index (CCI) is an oscillator that measures the variation of a security's price from its statistical mean. It is calculated as

$$\mathrm{CCI}_t = \frac{1}{0.015}\frac{\tilde{p}_t - \mathrm{SMA}_{t,n}(\tilde{p}_t)}{\sigma_t}. \tag{A.39}$$

Here, $\tilde{p}_t = \frac{p_t^{\text{high}} + p_t^{\text{low}} + p_t}{3}$ is the *typical price* at time t, $\mathrm{SMA}_{t,n}(\tilde{p}_t)$ is the n-period simple moving average of the typical price, and $\sigma_t = \frac{1}{n}\sum_{i=1}^{n} |\tilde{p}_{t-i+1} - \mathrm{SMA}_{t,n}(\tilde{p}_t)|$ is the mean absolute deviation.

The constant 0.015 ensures that approximately 70 to 80 percent of all CCI values fall in the interval $[-100, 100]$. The actual percentage of CCI values in this interval depends on the number of periods n. A shorter CCI, with low n, will be more volatile and have a smaller percentage of values within $[-100, 100]$, whereas larger n will result in a higher percentage of CCI values in the interval. $n = 20$ is a common default value for the CCI [101], but it can and should be adjusted to fit the time horizon under consideration.

The Commodity Channel Index can be interpreted in many different ways. One of them is to use the CCI to determine overbought and oversold levels. In this regard, values above 100 imply an overbought condition while values below -100 imply an oversold condition, each case indicating a possible price correction to more representative levels. Another interpretation of the CCI is as an indicator of trend signals. In this case, when the CCI rises above 100, the security is considered to be entering a strong up-trend and a buy signal is created. If one acts on this signal and buys the asset, then the position should be closed when the CCI falls back below 100. Similarly, when the CCI decreases below -100, the security is considered to be in a down-trend and a sell signal is generated. This signal ends when the CCI moves back above -100. A point of criticism to this interpretation is that the CCI often misses the early part of the price movement, resulting in smaller, potential profits for the investor. To overcome this, another interpretation is that a potential buy (sell) signal is generated when the CCI crosses zero from below (above). Finally, one can also integrate the interpretations and look for divergences. For instance, if the price is reaching new highs but the CCI is not, then the security is potentially oversold. On the other hand, if the price and the CCI are reaching new highs, then an up-trend is likely to follow.

A.12 Bollinger Bands

Bollinger Bands are a volatility indicator and is one of the most popular technical analysis techniques. The financial notion of volatility corresponds to the well-known statistical property of variance—or its square root, the standard deviation. Thus, Bollinger Bands show the spread of prices, thus giving an indication of how volatile the market or stock is.

Bollinger Bands consist of:

- an n-period moving average,
- an upper band located k standard deviations above the moving average,
- and a lower band k standard deviations below the moving average.

Letting B_+ and B_- denote the upper and lower band, respectively, these two bands can be formalized as

$$B_\pm = \mathrm{MA}_{t,n} \pm k\sigma, \tag{A.40}$$

where σ is the standard deviation

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_{t-i+1} - \mathrm{MA}_{t,n})^2} \tag{A.41}$$

Located between these two bands is the remaining, middle band, which is merely the moving average $\mathrm{MA}_{t,n}$ itself.

Evidently, there are again some parameters to choose. Since the Bollinger Bands make use of moving averages, the number of periods, n, to look back should be chosen. In this regard, the type of moving average should also be decided. And finally there is the number, k, of standard deviations. Most often, practitioners use a simple moving average with $n = 20$. However, exponential moving averages are a very common second choice. Finally, the number of standard deviations is typically chosen to be $k = 2$.

Now, when the market, or stock, becomes more volatile—i.e. when prices fluctuate more intensively—the bands widen, moving further away from the average. Conversely, during less volatile periods, the bands contract towards the average.

The standard interpretation is that Bollinger Bands do not give absolute buy and sell signals, but rather indicate whether the price is relatively high or low, thus allowing for more informed confirmation with other technical indicators. In any case, some characteristics of the Bollinger Bands are that sharp price changes tend to occur after the bands contract (corresponding to low volatility), and that prices moving outside the bands imply a continuation of the current trend. Moreover, tops and bottoms outside the bands followed by tops and bottoms made inside the bands call for reversals in the trend. Finally, a move originating at one band tends to go all the way to the other band.[14]

A.13 Chaikin Oscillator

The Chaikin Oscillator (CHOSC) measures the momentum of the Accumulation Distribution line (see Sec. A.8) using the MACD formula (Sec. A.4), making it an "indicator of an indicator". It is the difference between the 3-day EMA of the Accumulation Distribution line and the 10-day EMA of the Accumulation Distribution line:

$$\mathrm{CHOSC}_t = \mathrm{EMA}_{t,3}(\mathrm{AD}_t) - \mathrm{EMA}_{t,10}(\mathrm{AD}_t), \tag{A.42}$$

where AD_t is the Accumulation Distribution line (see Section A.8 above).

[14] There are numerous other advanced interpretations of the information given by Bollinger Bands, many of which include e.g. certain shapes and patterns made by the stock price (see e.g. [100]). Such interpretations can be difficult to properly implement in a computer program, however, and we therefore stick to the more manageable cases.

A.14 Chaikin Volatility

The Chaikin Volatility indicator compares the spread between a stock's high and low prices. It quantifies volatility as a widening of the range between the high and low.

The indicator is computed by first calculating an m-period exponential moving average ($m = 10$ by default) of the difference between the high and low prices:

$$\bar{\mathrm{HL}}_{\mathrm{t}} = \mathrm{EMA}_{t,m}(p_t^H - p_t^L), \tag{A.43}$$

where p_t^H and p_t^L denote the high and low price, and $\bar{HL}_t$ is the m-period EMA of the difference between these two prices. Th Chaikin Volatility is then found by calculating how much the moving average has changed (in percent) over a specified time of n periods ($n = 10$ by default):

$$\mathrm{CHVOL}_t = \frac{\bar{\mathrm{HL}}_{\mathrm{t}} - \bar{\mathrm{HL}}_{\mathrm{t-n}}}{\bar{\mathrm{HL}}_{\mathrm{t-n}}} * 100. \tag{A.44}$$

For additional indicators and their interpretations we refer to e.g. [93] or [99].

Appendix B

MATLAB Programs

As part of this thesis we have developed a number of different MATLAB programs. Some of them are the general functions for predicting a given stock using a particular method. The rest of the programs are our analyses and experiments. Although we have mainly considered the NASDAQ Index and a few other assets in this study, we have coded the programs in a general manner allowing one to perform analyses and predictions for almost any asset.

In total, the programs comprise tens of thousands of lines of code, and are thus much too long to show here in their entirety. We have uploaded them to

`https://www.dropbox.com/sh/3gbgf385bdqfjbv/AACkOWPFvmjjXwWOgVID1FZpa` [1]

where we will keep them available. Below, we list and describe each program file.

hist_stock_data.m : Function for retrieving historical stock data for one or more ticker symbols (assets) over a given time period. The data are downloaded from Yahoo! Finance [86].[2]

NN.m : Function for training and testing an Artificial Neural Network model. It takes as input a structure of stock data (as given by the `hist_stock_data.m` function), a matrix of exogenous inputs, the number of lags to use for prediction, the number of hidden neurons, a string defining the training function, the number of folds in cross-validation, the number of networks to train (the one with lowest MSE is chosen), and a structure with miscellaneous settings (show/hide plots, etc.). Its outputs include quantitative performance measures, plots of the predictions, and more.

SVM.m : Function for training and testing a Support Vector Machine model. The general structure, flow, and outputs are exactly the same as for the `NN.m` function above, so as to make the use and comparison of our programs and their results as easy as possible. Most of the inputs are also the same; the SVM-specific inputs include scalars specifying the type of SVM and the kernel function, and a structure with model parameters.

RF.m : Function for training and testing a Random Forest model. Again, the inputs, outputs, and general flow is the same as the other programs for the prediction methods. The RF-specific inputs include the number of trees to grow and the leaf size.

[1] Alternative links: `https://dl.dropboxusercontent.com/u/77296171/MSc%20Thesis%20-%20Christian%20Okkels/matlab-programs-msc-thesis-christian-okkels.rar` and `https://dl.dropboxusercontent.com/u/77296171/MSc%20Thesis%20-%20Christian%20Okkels/results-msc-thesis-christian-okkels.rar`.

[2] We have modified the original version (see [87]) to suit our needs.

ARMAGARCH.m : Function for training and testing an ARIMA(p,d,q)-GARCH(u,v) model. The inputs, outputs, and flow are again the same as for the `NN.m`, `SVM.m`, `RF.m` programs above. The unique inputs in this case are the specific model parameters, the type of conditional variance model, and the innovation distribution.

exogenousInputs.m : Function for computing a variety of technical indicators to use as exogenous inputs in the prediction models. The input is a structure of stock data (as provided by `hist_stock_data.m`).

main.m : Program that calculates statistical properties, performs exploratory data analysis, and computes technical indicators for a user-specified asset and time period. The program also performs a thorough time series analysis, including a quantitative and qualitative analysis of autocorrelation and conditional heteroscedasticity as well as tests for randomness and stationarity. It also includes the construction, diagnosis, and comparison of conditional mean and variance (ARMA-GARCH) models as seen in Section 6.1.4 for the NASDAQ Index; but the program allows for such an analysis for any user-specified asset.

main_nn.m : Program containing the ANN-specific analyses and experiments presented in 6.1.5 for the NASDAQ Index. Again, the user can specify any asset on which to perform the analyses and predictions.

main_svm.m : Program for performing SVM-specific analyses and experiments, as shown in 6.1.6 and 6.1.7.

main_rf.m : Program for performing RF-specific analyses and experiments, as shown in 6.1.8.

main_experiments.m : Program containing our numerous experiments in 6.2.

main_oscil_model.m : Implementation of the simple, linear version of our oscillator model, as described in Section 3.4 and used in Experiment 10 (Section 6.2).

The programs listed above are the main and most important ones. There are, however, additional programs we have used for smaller tasks. The vast majority of these we have written ourselves: e.g. `cutNanRows.m` which cuts NaN ("Not A Number") elements to properly prepare and synchronize vectors and matrices of time series data; `armagarchmodel.m` which is similar to `ARMAGARCH.m` (but less complex); `neuralnetwork_mse.m` which is a function for quickly calculating the Neural Network MSE performance on a training set and thus be used as an object function in an optimization technique; `sunspots.m` which finds and plots the sunspot data (as shown in Section 3.2); as well as `XX_regression.m` and `XX_classification.m` (where XX = NN, SVM, and RF) which are similar to (but slightly less complex than) their above `XX.m` equivalents and used for some of the initial analyses and experiments in the `main_xx.m` programs (where xx = nn, svm, and rf). All of these programs have been thoroughly commented to explain what is going on in the different lines of code.

The basic SVM implementation used in our SVM programs is contained in the files `svmtrain.mexw64` and `svmpredict.mexw64` from the LIBSVM framework [33]. Similarly, the basic MATLAB implementation of Breiman's Random Forests [35] used in some of our RF programs is found in the files `mexRF_train.mexw64`, `mexRF_predict.mexw64`, `regRF_train.mexw64`, `regRF_predict.mexw64`, `mexClassRF_train.mexw64`, `mexClassRF_predict.mexw64`, `classRF_train.mexw64`, `classRF_predict.mexw64`. The basic implementation of Random Forests used in some of our other programs, and the basic Neural Network implementation, are provided by built-in functions in MATLAB's Statistics and Neural Network Toolboxes [32], respectively.