

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Libor Grafnetr**

Study programme: Open Informatics

Specialisation: Artificial Intelligence

Title of Diploma Thesis: **An environment for testing financial series predictors and learning such predictors with artificial neural networks**

Guidelines:

1. Produce a review of applications of instance-based learning and artificial neural networks in the prediction of financial time series.
2. Develop a software environment for experimental evaluation of algorithms for financial time series prediction through machine learning.
3. Design and apply a method based on splitting the input data into clusters, training of individual neural networks on these clusters, and subsequent combination of the networks outputs for prediction.

Bibliography/Sources:

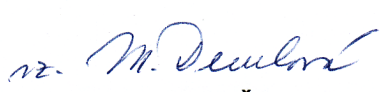
- Brockwell P. J., Davis R. A., Introduction to Time Series and Forecasting, New York, 2002 (Springer)
- Schelter B., Winterhalder M., Timmer J., Handbook of Time Series Analysis, Weinheim, 2006 (Wiley)
- Tsay R. S., Analysis of Financial Time Series (Third), 2010 (Wiley)
- Gupta N., Hauser R., Forecasting Financial Time-Series using Artificial Market Models, 2005
- Brockwell P.J., Davis A.D., Time_Series - Theory And Methods SE, SPR, 2006
- Holger K., Schreiber T., Nonlinear Time Series Analysis, Dresden, 2002 (CUP)

Diploma Thesis Supervisor: Doc.Ing. Filip Železný, Ph.D.

Valid until: to the end of the summer semester 2011/2012


prof. Ing. Vladimír Mařík, DrSc.
Head of Department




prof. Ing. Boris Šimák, CSc.
Dean

Prague, March 22, 2011

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



An environment for testing financial series predictors and learning such predictors with artificial neural networks

Author: Libor Grafnetr
Supervisor: doc. Ing. Filip Železný, Ph.D.
Year: 2011

Acknowledgments

I would like to thank my thesis supervisor doc. Ing. Filip Železný, Ph.D.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v příloženém seznamu.

V Praze dne

11.5.2011



.....
podpis

Abstract

This thesis describes analysis, design, implementation and testing of an environment for evaluation of financial time series predictors. The purpose of this environment is to allow uncomplicated set up of a market forecasting process from data input and preprocessing to prediction and trading simulation. For each step of the process, multiple commonly used methods are available and new ones can be added effortlessly.

After the configured prediction process is executed, the prediction accuracy and performance of trading based on made predictions are calculated, which allows to assess the quality of the prediction process. The environment also supports optimization of parameters of the process and logging of outputs of the experiment.

In addition, an experimental prediction method using combination of input data clustering and a neural network group is designed and implemented into the testing environment.

Environment's functionality is demonstrated by optimizing multiple machine learning methods and comparing their performance by standard accuracy measures and profits of simulated trading.

Abstrakt

Tato práce popisuje analýzu, návrh, implementaci a testování prostředí pro hodnocení prediktorů finančních časových řad. Účelem tohoto prostředí je umožnit jednoduchou tvorbu procesu pro předpověď vývoje trhu, a to od datových vstupů a předzpracování po stanovování předpovědí a simulaci obchodování. Pro každý krok procesu je k dispozici několik běžně používaných metod a nové mohou být jednoduše přidány.

Po spuštění předpovědního procesu je vypočtena přesnost předpovědí a výkonnostní ukazatele obchodování založeného na vygenerovaných předpovědích, což dovoluje zhodnotit kvalitu předpovědního procesu. Vytvořené prostředí také podporuje optimalizaci parametrů procesu a záznam výstupů experimentu.

Dále byla navržena a implementována experimentální předpovědní metoda založená na kombinaci shlukování vstupních dat a skupině neuronových sítí.

Funkčnost prostředí je názorně předvedena optimalizací několika metod strojového učení a jejich porovnáním pomocí standardních měřítek přesnosti a zisku ze simulovaného obchodování.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objectives	2
1.3	Project breakdown	4
1.4	Thesis structure	4
2	Background	6
2.1	Existing approaches to forecasting	6
2.2	Prediction	8
2.3	Instance-based learning	9
2.3.1	Preprocessing	9
2.3.2	Methods	10
2.4	Neural networks	12
2.4.1	Preprocessing	13
2.4.2	Methods	14
2.5	Performance evaluation	16
3	Design	18
3.1	Software environments evaluation	18
3.2	General structure	19
3.3	Data input	21
3.3.1	OHLCV representation	21
3.3.2	Datasets	21
3.4	Preprocessing	22
3.4.1	Initial shaping	22
3.4.2	Indicators	23
3.4.3	Target value	23
3.4.4	Normalization / Discretization	24
3.4.5	Instance space conversion	25
3.4.6	Component analysis	25
3.5	Prediction	25
3.6	Trading	26
3.7	Performance evaluation	28
3.8	Logging	29
3.9	Optimization	29
3.10	Clustering + Neural Networks group	29

3.11 Platform	30
3.11.1 Rapidminer	30
3.11.2 R environment and packages	31
4 Implementation	32
4.1 Data input	32
4.2 Preprocessing	33
4.3 Prediction	34
4.4 Trading	34
4.5 Performance evaluation	35
4.6 Logging	35
4.7 Optimization	35
4.8 Clustering + Neural Networks group	36
4.9 Environment parameters	37
5 Testing	42
5.1 General comparison	42
5.2 Conclusion	45
6 Future work	47
7 Conclusion	48
References	49
A Notable trading strategies	54
B CD directory structure	59

Chapter 1

Introduction

*Markets can remain irrational
longer than you can remain solvent.*

John Maynard Keynes

1.1 Motivation

Trading of stocks, commodities and financial instruments on global markets is a highly interesting area of focus for science scholars since it combines unique challenges with very obvious potential rewards. Although sophisticated research into prediction of financial time series has been going on for many decades, it was almost exclusively relying on econometric methods.

However, during ongoing evolution of Machine Learning (ML) as a scientific discipline, prediction of time series has always been one of its primary applications. As a result ML has been slowly but steadily gaining interest from researchers in financial markets prediction. In past two decades this has led to vast amount of published research in the area. Most of this research follows a similar path: choosing a certain set of ML methods and testing their behavior when applied to prediction on a specific time series. Depth of these researches ranges from general comparative study of multiple methods where various measures of accuracy and performance are evaluated to an in-depth analysis of specific aspects of a method and their effect on the method's prediction behavior.

In most cases an extensive stack of heterogeneous components was set up and often even developed from scratch before the research itself commenced. From data collection, representation, cleansing and transformation to the ML algorithm itself, efficient data structures it requires and the final evaluation of the prediction performance - all these segments had to be created or selected from existing tools and adapted to correctly interact with each other.

When applying a ML approach to a research task, the most time is always spent on experimentation and tuning of the individual components of the process. And we believe that this is where the researcher's valuable time should be invested

as opposed to recreating the whole prediction stack yet again, or struggling with replacing individual components by their alternatives just to verify whether they don't perform better in the current task.

Although several categories of tools try to facilitate the task of predicting market movements and every one of them has their strong point, none succeeds at providing a prepared powerful environment ready for experimenting with automated predictions:

- **Trading tools** - Software specifically targeted at analyzing market data. It provides means to manipulate with the data and usually allows to access online data sources. Rich visualisation capabilities give good notion of the market. Many common financial indicators such as moving averages, Bollinger Bands, etc. can be layered onto the visualisation allowing the user to deduce predictions based on empirical knowledge. However, automation environments in these tools are rather limited and not at all optimized for development of sophisticated prediction systems based on ML.
- **General computing environments:** Environments such as R, MATLAB and others are designed to allow user to easily programmatically operate with data such as financial time series. Thanks to their extendibility, specialized packages for common processing or financial data are available. Likewise, in the area of machine learning algorithms these tools provide some options, although their usage and architecture may differ notably. Paradoxically, their disadvantage lays in their versatility, as finding and putting all the necessary parts together may be a lengthy process, may require implementation of intermediate steps or often resolution of compatibility problems between the components.
- **Machine learning environments:** Tools such as RapidMiner or WEKA provide a large number of components for input, preprocessing, evaluation and most importantly machine learning. Most popular methods are present, often in several modifications. Yet, the user still needs to put together the whole prediction process and is rather limited when custom operations, such as financial-related preprocessing are necessary.

As can be seen none of the tools alone provides prearranged, end to end solution for financial prediction. Creating this solution will be our objective as described in the next section.

1.2 Objectives

Our objective will be to create a testing environment for the financial time series prediction task. This environment will be extendible and highly adjustable prediction and evaluation process in itself. In addition, it will allow automatic testing of various paths through the process and parameters tuning. The testing will not only yield the best performing sets of methods and their parameters, but will provide a complete overview of performance through the parameter space.

As input, the testing environment will use several chosen sample datasets, but also should be able to automatically obtain up-to-date data from freely available sources. The environment should be able to operate on multiple scales of data, e.g. minutely, hourly, daily. Also several types of instruments must be supported, such as stocks and futures.

The data will then need to be preprocessed. This includes adding financial indicators derived from the data, removing the absolute value of the variates, eventually normalization, discretization or filtering. Apparently, there are multiple choices for all of these steps and the environment needs to provide them and let the user choose which is to be used. Further transformations of the data such as Principal Component Analysis are also a desirable functionality.

Defining the predicted value is as well a step where several alternatives are possible. It may be a numerical value or an observation's class. This will define whether the prediction will be a regression task or a classification task - our environment needs to support both.

Prediction step should have multitude of options - Nearest Neighbors, Artificial Neural Networks, Support Vector Machines, etc. The environment should allow, as with most of the other components, to select a prediction algorithm regardless of what previous choices were made. Tuning prediction algorithms is where the most user's time will be spent, therefore at least some basic parameters for the included methods should be present.

Once predictions are generated for testing data, the environment will provide two kinds of evaluation - a synthetic one, based on common accuracy measures and an empirical one, based on simulated trading. Trading on financial markets is usually governed by a strategy, which uses a set of rules and signals to decide whether and what types of market orders should be place. We need to find means to build and execute such strategy on the testing data. Additional costs, such as order fees and market slippage need to be taken into account.

The environment will likely be based on a combination of existing tools that will suitably supplement each other. For transformation of the data and simulation of trading based on predictions a general computational environment will be used, whereas the prediction and direct evaluation of its quality will be performed by a machine learning environment.

A crucial feature will be the ability to automatically optimize the prediction process. This will be achieved by parametrization of the whole process. These parameters will need to be externally accessible, so that general optimization algorithms may be connected to the process. A metric for evaluation by the optimization algorithm (which metric will be used has to be configurable as well) will be provided at the output of the process.

In order to ensure reproducibility of results, the environment will need to provide extensive logging capabilities. This includes automatic storage of performance measures, predictions and used parameters. In addition, visualisations of the trading process should be provided in order to manually estimate the rationality of the predicted signals.

To make this process more than just a single-purpose monolith, it has to be easily extendible and modifiable. This means that the user should be able to add his own methods without much effort, as well as modify the behavior of the existing ones. This also requires the environment to be well-designed - to ease orientation in it.

1.3 Project breakdown

To achieve the above we have divided the project into following tasks that need to be accomplished:

- Explore time series preprocessing and prediction methods, best practices and experimental results in the machine learning area with focus on financial series. This will help us decide what components to include and how to structure the environment.
- Find suitable existing machine learning and computational/statistical environments. Verify that they are adequate to meet our needs, including appropriate range of choices for each step of the process described above. These environments will need to be able to integrate well with each other.
- Prepare a design of a modular system starting with predefined financial datasets and ending with relevant metric output. Reflect on the nature of selected tools and earlier requirements such as extendibility and support for automated optimization.
- Implement this system in previously selected environments. Also implement the encapsulating optimization method. Both implementations need to incorporate detailed reporting - statistics of the prediction process associated with used parameter combinations and helpful visualisations.
- Test the developed environment by evaluating multiple preprocessing and ML prediction algorithms using parameter optimization. Show influence of several major parameters on the prediction accuracy. Investigate relation between absolute accuracy and the trading behavior.

1.4 Thesis structure

This thesis will be divided into 6 chapters, which will describe the author's progress on tasks laid out in the previous section:

- In chapter 2 we will introduce the general characteristics of Machine Learning (ML) task and its similarity with econometric methods. Next, we will list possibilities of what can constitute a predicted value. Afterwards, a deeper review of two major methods of ML and associated preprocessing practices is given. This review is based on our research into existing literature in the area and therefore provides many references to notable resources. Finally,

basics of performance evaluation from both, ML and trading, point of view are described.

- In chapter 3 the environment's architecture will be gradually built. First, the general structure and then the specifics of individual components of the process. The optimization process, which will encapsulate the main prediction environment, will be constructed as well. This chapter will also describe the software tools and their extensions, on top of which we will implement the designed project.
- Implementation of the project according to the design is described in chapter 4. This includes details on connecting the tools, which parts of the project are realized in which environment and what precise processing steps were applied to the data. The chapter concludes with overview of configurable parameters that the implemented process supports with explanation of possible values.
- To test how the environment meets its purpose of allowing fast experimentation, in chapter 5 we describe experiments we have executed. These experiments show general prediction performance, trading performance and influence of various parameters on these results.
- In chapter 6 we outline directions, in which we would like to expand the environment in the future.
- Final conclusion of what has been achieved and what experience has this work given to the author is in chapter 7.

Chapter 2

Background

*Essentially, all models are wrong,
but some are useful.*

George E. P. Box

Successful prediction of financial series using econometric methods is always based on a more or less general assumption about the behavior of the market. Such assumption needs to provide leads on how to approximate the assumed behavior. Usually, the derived approximation method is parametric, meaning it depends not only on the processed data itself, but also on additional values not directly calculable from the data. Very often these parameters are to be found by experimentation or empirically evolved estimation methods (e.g. Box-Jenkins methodology for fitting autoregressive moving average models[Box et al., 1994]).

The situation is very similar in Data Mining / Machine Learning. Every method, be it from the area of preprocessing, prediction or evaluation has its parameters, often a significant number of them. Furthermore, there are multitude alternative methods for each individual step. These circumstances provide many degrees of freedom, which open many wrong turns to take.

In order to be able to better understand what choices should be made during creation of our testing environment we need to explore existing knowledge in the area. In this chapter we will give an overview of what consist a prediction and of two prominent methods of machine learning. We will describe preprocessing methods for each of them, variations of both methods and their specifics. This description is based on our review of relevant scientific articles in the area and therefore should provide enough references for eventual deeper exploration.

2.1 Existing approaches to forecasting

There are myriad approaches to forecasting financial time series outside the ML area. They vary in sophistication, the time scale of their predictions and the amount of knowledge and time necessary to extract a prediction. They may be divided into following categories:

- **Fundamental analysis** of financial time series examines the object of the series. In case of stocks, the object is the company whose stock is analysed, but also the industry in which it carries its business and any other relevant factor that may influence the success of the company. In case of commodity futures, the examined factors will likely include the state of the industry where the commodity comes from (mining, agricultural, etc.) and of industries where it is consumed. Obviously, the insight and analytic skills necessary to perform a prediction based on fundamental data are significant. The prediction horizon can be very variable, but is likely to be upwards of several days. The advantage is that behind every prediction there is an explicit rational process, which also includes an estimation of confidence of the prediction.
- **Technical analysis** of time series is based on analyzing the data itself. This is done by estimating certain states of the market, such as trends, price levels and momentums. This is done with aid of various transformations and projections - indicators - of the data. Some of the notable ones are Moving Average Convergence/Divergence, Relative Strength Index, Bollinger Bands or a Moving Average, but there many others. Predictions are extracted using rules that are associated with each indicator. Second method of the technical analysis is repetitive patterns recognition, which can suggest what behavior will the market exhibit in the future. Trading rules based on indicators may be automatized and many trading tools currently do so. Also, an automatic recognition of hard-coded patterns is imaginable, although the author is not aware of any well known tool performing this task. However, many traders still perform technical analysis to large extent manually, using trading tools only as a visualisation guide. The confidence of each prediction based on technical analysis is rather hardly predictable.

Nevertheless, adding technical indicators into the data input for machine learning based prediction is a reasonable step. The ML methods should to some extent be able to extract the presumed regularities in indicators' behavior if they are strong enough.

- **Econometric models** aim to approximate the market by employing one or more assumptions about its behavior (e.g. autoregressive nature, oscillation around the moving average, random variable heteroskedasticity, etc.) and building a parametrized generative apparatus. Many models (apparatuses) have been developed, often improving on each other - the more known are e.g. ARMA, ARIMA or the family of GARCH models. For prediction on certain data, a model is selected and fitted to the data, i.e. the best performing parameters are found. The predictions and assessments of the market can then be drawn from the model.

Although these types of models are built using sound theory based on statistics and probability and seem to be able to capture major features of many time series, they (perhaps subjectively) seem slightly too rigid to capture the finer properties of the market's behavior, which in the models' interpretation may be regarded as noise.

It should be noted that there are more sophisticated models relying on proba-

bility theory and advanced stochastic process modelling - Markov processes in continuous time; and Random Fields. However, the author hasn't investigated the extent of their usage or success when applied to financial data.

With regard to our testing environment, econometric models could be added in future development to allow comparison with pure ML techniques. Experiments with plugging these models into meta-algorithms such as bagging or boosting also seem viable. However, currently we will not employ these models in any way.

- **Machine Learning** methods are also being utilized in financial prediction. But although they have been researched and known for quite some time, they never gained popularity to the extent comparable with econometric approaches, despite records of both excellent theoretical results and successful applications within trading frameworks. However, much of the research in the financial area is done privately and not published, which leads us to believe that there might be a considerable disparity between the limited public exposure of these methods and the real extent of their usage.

We will give overview of lazy learners (k-Nearest Neighbors) and Neural Networks further in this chapter. Other methods such as decision trees, Markov models or Inductive Logic Programming are also being applied to prediction and are examples of possible future additions to our environment.

Most of the existing literature and research work deals with ML prediction methods in isolation and their results are measured using standard accuracy measures. Luckily, there are some recent works that connect all the necessary components into a complete solution and provide practical market performance results [Barbosa and Belo, 2008], [Lee et al., 2007], [Martinez et al., 2009].

A great overview of the financial time series data mining research area is provided in [Zemke, 2003].

Tightly connected and at least as important as making a prediction is the trading strategy governing the actions that are taken based on a prediction. The strategy specifies timing, what properties should an order or orders placed to the market have and whether to place them at all. This includes methods for order sizing, policies governing what type of order is placed or how limit orders lowering the risk in case the market goes against the prediction are used. More advanced schemes such as hedging, arbitrage or spread trading are also often utilized.

2.2 Prediction

The prediction itself may have various forms. One class of methods may be a numeric predictor, where one or more future values of the time series are forecast. However, this output might be relatively inaccurate and the application necessarily doesn't demand as exact information as a specific numeric value.

Simply recognizing whether at a given moment the time series evolution will take some expected direction is very well sufficient in many scenarios. The taken

direction might be a rising trend, a movement assuring that the trend will continue or a precursor of a sudden drop of the series. This recognition is commonly formulated as a classification task. More advanced variations of the both numeric predictor (e.g. predicting expected minimum and maximum value on a day) and the classifier exist as well.

Combining approaches such as decision trees, ensembles and mixtures of experts are also frequently used.

2.3 Instance-based learning

To perform a given learning task the Instance-based learning (IBL) methods operate on a dataset of individual instances. For an input instance, the algorithm retrieves relevant instances and combines them into the output. The definition of relevant instances and their combination is defined by each method and eventually by its parameters.

In the prediction task, the algorithm is given an instance whose future evolution we would like to know and yields either a forecast or a predicted class of the instance.

Since time series are continuous, one needs to define how to extract an instance from the data flow. Practically, the only used method is application of a sliding window on the series. The width, shift and sampling rate of the window are again optimizable parameters. It is important to note that the instance extraction necessarily needs not to be literal. The algorithms working with the concept of an instance may merely operate on appropriate subsequences of data points in the time series stream.

The time series values may need to be altered before being processed into individual instances. Similarly, transforming the instances may be necessary. This is discussed in the next section.

2.3.1 Preprocessing

The financial time series are given as absolute values, be it in ticks or the actual price. Since the absolute value is in most cases relatively large in comparison to its change between neighboring data points, it is very advisable to apply a difference operator, i.e. $\Delta x_n = x_{n+1} - x_n$. This way we will obtain a returns series, which is much more suitable for further processing [Maggini et al., 1997]. Other noticed transformations of the time series are Moving Average of several past data points, ratio operator ($x_n = \frac{x_n}{x_{n-1}}$), or higher orders of the described difference operator.

Each instance extracted from the series, can be defined as $x_t^{m,\tau} = (x_t, x_{t-\tau}, \dots, x_{t-(m-1)\tau})$, where m is called an embedding dimension and τ is a delay parameter [Rodríguez et al., 1999]. The delay parameter is almost always set to 1 in the case of financial time series, i.e. no subsampling is taking place. There is also an implicit parameter of the window's shift, which is almost invariably equal to 1.

Embedding dimension (i.e. the length of the instance feature vector) is a very important parameter and many approaches for its estimation exist. A very popular one seems to be Casdagli's algorithm [Casdagli, 1992], while one of the more recent interesting approaches is [Zufiria and Campoy, 2009], where Self-Organizing Maps are used to provide whole probability distribution on the embedding dimension. More methods can be found in subsection 2.4.1, since Neural networks are sensitive to embedding dimension value and thus literature in the area lends more focus to available estimation methods.

Instance feature vectors may be further transformed into alternative representation in order to reduce dimension or to allow usage of more advanced metrics and data structures. This may lead to both increased efficiency and higher accuracy. Alternative representations are introduced in the next section along with the learning methods themselves.

2.3.2 Methods

A strong majority of works on time series prediction using Instance-based methods use Nearest Neighbor (NN) approach [Maggini et al., 1997][Barbosa and Belo, 2008] [Arroyo, 2010]. This method finds similar instances according to a provided metric and combines them into the desired output. The obvious assumption of this approach is that similar subsequences of a time series are likely to have similar continuation (evolution) of the subsequence.

If no dimension reduction is applied, commonly used metric is the Euclidean distance. Another very suitable, yet more advanced, metric is Dynamic Time Warping (DTW), which allows the compared sequences to vary in speed, i.e. it is able recognize two instances as very close if they contain very similar pattern, although one is realized in different number of data points than the other.

If the neighbor look up is taking place in reduced instances space, the applicable metrics are dictated by the representation of the instance. However, most of the distance metrics are modelled to exhibit similar behavior as the Euclidean distance or DWT - often the works state the problem as implementing Euclidean or DWT using alternative representation for the sake of efficiency improvement.

If the prediction is formulated as a classification task, known classes of the found neighbors determine the input's instance target class. Commonly, the target class is the class with the highest frequency (majority class) or highest normalized frequency (account for distance of each neighbor).

When a numeric forecast of future observations is the desired output, combination of the data points following after each of the neighbors is calculated. Popular options include linear combination, weighted linear combination based on the distance of each neighbor or linear autoregression [Rodríguez et al., 1999]. In the linear autoregression, the forecast data point is calculated using preceding points as $x_{n+1} = a_0x_n + a_1x_{n-1} + \dots + a_{m-1}x_{n-(m-1)} + a_m$. The nearest neighbors are used to perform least-squares error estimation of the coefficients $a_{0..m}$.

While the idea of NN algorithm is very straightforward, the realization for massive datasets needs to deal with the complexity of finding the neighbors. This

problem (often more generally termed similarity search) can be resolved using appropriate data structures and/or alternative instance representations that reduce the dimension of instances.

M-trees is a popular data structures family based on organizing the instances into a tree based on their relation through the metric used for the neighbor lookup [Ciaccia et al., 1997]. Other used metric-based tree structures include VP-tree [Fu et al., 2000], Bk-tree, Pk-tree or R-trees. Combinations of a more complex pre-processing and a specific suitable data structure are also known, such as iDistance method, which creates a B^+ -tree based on clustering the instances [Yu et al., 2001]. Different representations of instances broaden the possibilities and are used for one or more of following reasons: reduce the dimension of the instance feature vector (i.e. reduce the amount of data), speed up the similarity search, provide better results. The dimension reduction is often the means to improve the search speed. However, the reduction transformation needs to preserve distance relations in a way that will prevent near relevant neighbor to be excluded from the result due to large distance in the transformed space. This property of combination of a feature space and a metric is called lower bounding.

Since some information is inevitably lost during the transform, the search results in the transformed space are approximate. Therefore, most works on the subject include a secondary step where the approximate result set, whose size is already reasonable, can be further processed to retrieve exact results.

Using Discrete Fourier Transforms (DFT) to significantly reduce the dimension and R*-tree as an index structure for several most significant DFT coefficients is one of the early approaches [Agrawal et al., 1993]. A Discrete Wavelet Transform (DWT), especially Haar Wavelets, is another option to reduce the dimension and is explored in [Chan and Fu, 1999]. Dynamic Time Warping metric can be applied to DWT as described in [Chan et al., 2003]. Symbolic Aggregate Approximation (SAX) is a transformation that is recently gaining popularity, which represents an instance as a string of symbols. Each symbol denotes a value range in one segment of the instance. SAX is based on Piecewise Aggregate Approximation and extends it with transformation from segment aggregate values into the symbols. Distance metrics for SAX that lower-bound Euclidean and DTW exist, as well as several composite metrics [Nguyen and Anh, 2007] [Liu and Shao, 2009].

The last representation we will mention is Piecewise Linear Approximation, which approximates segments of an instance with line segments. Application to improve efficiency of the DTW metric on extensive data sets has been pioneered in [Keogh and Pazzani, 1999].

An appealing variation to the k-Nearest Neighbor is the K* algorithm [Cleary and Trigg, 1995], which uses entropy as a distance metric. It has been applied in an ensemble in [Barbosa and Belo, 2008] trading agent implementation.

Most of these methods are applied to univariate time series. Multivariate time series pose considerably more complicated problem, since the methods often go from polynomial to non-polynomial complexity. However, more advanced attempts to predict financial time series require more dimensions (e.g. for open price, close price, mean, volume etc.), hence different approaches must be taken. These approaches are

in some cases extensions of the univariate ones or follow a different path altogether. A thorough review of existing methods and new interesting solutions can be found in [Yang and Shahabi, 2007].

Although all of the above works focus on efficiency, there are still time bounds that Nearest Neighbors can have big problems to meet. These tight time bounds are required for tick-by-tick scale (where High Frequency Trading operates). Some answers to these problems can be found in the research area of Data Streams processing, where input data may come in high volumes in non-constant intervals. The methods need to do very fast both - give a prediction and incorporate the new data. A related problem - classification on data streams employing adaptive nearest neighbor is presented in [Law and Zaniolo, 2005]

2.4 Neural networks

Artificial Neural Networks (ANN) have a long tradition of being used as approximators with training based only on function's input and output. The universal approximation abilities have been formally proven by [Cybenko, 1989]. Yet really successful application of ANN is a complicated process that may include many unsuccessful tweaking attempts. This is caused by non-transparency of the state of an ANN, i.e. in most cases one can't explain why the ANN tends to behave one way or another. On the other hand, ANNs are able to grasp the nature of the process to some extent and provide reasonable performance in most cases.

The basic ANNs are generally defined by $\langle nodes, weights, activation\ function \rangle$, where the weights can also be used to indirectly describe the structure of the network. The activation function defines the output of a neuron based on the weighted sum of its inputs. Common choices are the logistic, tanh or step function. More advanced architectures may include recurrent or over-two-layers connections of neurons or custom activation functions.

The parameters that need to be chosen are the number of hidden neurons and the structure of the network (depending on the type of the network). There are no widespread techniques for neither of these tasks and most configurations are determined through several rules of thumb and various search methods, such as Genetic Algorithms. Generally, it is recommended to keep the number of neurons high just enough to be able to capture the generating process' nature, but low enough to prevent memorization of the input data.

The input to the ANNs is an instance vector in the case of univariate time series. For multivariate series, the embedding dimension is in most works reduced to 1, so input is a vector comprising of individual values of each dimension of the multivariate series.

A great review of existing approaches to foreign-exchange rates prediction using ANNs is given in [Huang et al., 2004]. An overview on existing literature compared through used parameters, prediction horizon and success of the experiment is in [YU et al., 2007]. Other recent review of existing approaches to forecasting stock markets using softcomputing approaches (including ANNs) can be found in [Atsalakis and

Valavanis, 2009]. An older, general, but explanatory review of forecasting with ANNs is [Zhang et al., 1998].

2.4.1 Preprocessing

For ANN time series prediction the crucial part of preparing data is selecting the most beneficial inputs. The networks are sensitive to the number of input variables and to hidden dependences between the inputs, therefore the selection process should evaluate contribution of each input and its relation to other inputs. However, most works do not seem to have a rigorous process of selecting inputs, either with no explanation or with a reference to use of expert knowledge.

The most straightforward input setting for univariate application is to feed the network instances (time-delayed vectors) extracted from time series as described in subsection 2.3.1. Results for networks using just the time-delayed vectors aren't very satisfactory, as they are on par with the conventional ARIMA models and behind networks with other inputs [Yao and Tan, 2000]. Better results can be easily achieved using more advanced inputs.

The input can contain indicators of the time series, such as Moving Average (MA), Relative Strength Index or Momentum. These may be combined with time-delayed vectors (shortened accordingly in order to maintain reasonable number of input nodes) to provide more condensed information on the history of the time series to the network. Even several indicators of the same type but with different parameters may be used ([Yao and Tan, 2000],[Martinez et al., 2009]) uses MAs with lengths 5, 10, 20, 60).

By including inputs not directly derived from the time series, the network can be considered to perform fundamental analysis. Selection of such inputs depends on the predicted financial time series. E.g. for currency exchange rates the relevant information may be GDP, interest rates, consumer price index, etc.

In order to determine which inputs are advantageous, several methods may be used. Traditional choices are Akaike Information Criterion or Bayesian Information Criterion. When time-delayed vectors are used as input (or part of it), embedding dimension can be determined by methods mentioned in subsection 2.3.1, Autocorrelation Criterion[Huang et al., 2004], which is used specifically for NNs, False Nearest Neighbor or simply by leaving out individual inputs and comparing the performance of created networks.

ANNs give slightly more freedom in input normalization and transformation than Nearest Neighbor, since they are able to cope better with unnormalized data in most prediction tasks. Still, most literature transforms and normalizes the data. The available options are similar to those used for Nearest Neighbor (subsection 2.3.1), i.e. a difference operator, ratio operator, logarithm, mean and deviation correction, etc. When the ANN performs numerical prediction tasks, it is necessary to de-normalize the output values, since they will inevitably fall into an interval determined by the activation function. Normalization is also necessary when the input consists of values from multiple time series.

Transformation into wavelets has been applied with positive effect on accuracy in [Mitra and Mitra, 2006].

2.4.2 Methods

ANNs are divided into many types, each of them specific in the network's architecture, different type of activation function, specialized neurons or combinations of the above. Most of these types have been already tested in financial time series predictions, some of them have been consistently performing relatively well. We will describe several most common types below.

Apart from choosing the architecture, there are several options as to the output of a network. This again depends on the designated task. For conventional prediction tasks the network has one output neuron giving the predicted value (which might have to be de-normalized). More advanced variations of the prediction task may have several output neurons, whose values are subject to weighted summation. The weights may be adjusted dynamically with aim to improve performance [Baba and Suto, 2000]. Other noteworthy option is to have the network predict minimum and maximum for a predefined time unit [Martinez et al., 2009].

ANN may also be used as a classifier, in which case only the direction (sign in the case of differenced values) is usually predicted. While not yielding the actual value, classification approach may have better accuracy than direction prediction based on a network predicting the numeric value.

Multi-layer Feed-forward Neural Networks (MLFNN) are the basic type of network with one or more hidden layers, each layer connecting only to the next layer and a standard sigmoid function used as activation function. Majority of works use this type of network. The properties of the networks used in literature are following: there are 1 to 3 hidden layers, number of neurons are in ranges 2 - 60 / 5 - 60 / 1 - 3 in input / hidden / output layers, respectively. While they generally perform well, in most cases they are marginally or even significantly worse than more advanced types of networks. Well known methods for training are Backpropagation, Levenberg-Marquardt or Genetic Algorithm [El Shazly and El Shazly, 1999], [Mitra and Mitra, 2006]. General optimization methods such Gauss-Newton or Steepest Descent can also be applied, but generally do not achieve good performance. Detailed description of construction and application with comparison against ARIMA is in [Yao et al., 1999]. Another well-performed application (with 2 output neurons for min+max value prediction) was done in [Martinez et al., 2009].

Radial Basis Function Networks (RBFN) have architecture similar to the MLFNN. Their main difference is the activation function of neurons in the hidden layer. This function is defined by a center and a radius and is symmetric around the center - in most cases a Gaussian function is used for the purpose. RBFN performs quite well in most comparisons [Kodogiannis and Lolis, 2002]. While most of the training algorithms for MLFNN can be modified and applied to RBFN, there are two popular specializations - Orthogonal Least Squares [Kodogiannis and Lolis, 2002] and EvRBF [Rivas et al., 2004] (the work also provides categorization of training approaches for RBFN).

Other family of ANNs are *Recurrent Neural Networks* (RNN), in which connections in the network can form cycles. The basic type has all the neurons in the hidden layer strongly connected - this leads to large number of connections and slow convergence. Another type - the Autoregressive RNN has recurrent connections only within individual neuron (the connection originates and end in the same neuron). Third common type is the Elman network, which presents context neurons serving as memory cells. Outputs from the hidden layer are stored in the context neurons and fed to the hidden layer along with new inputs on the next iteration. The latter two types are described and compared with other types on exchange-rate prediction in [Kodogiannis and Lolis, 2002].

Probabilistic Neural Networks (PNN) and *General Regression Neural Networks* (GRNN) are basically representations of Bayes Classifier and Kernel regression, respectively, expressed as neural networks. They both consist of 4 layers: input, pattern, summation and output. The pattern layer contains a neuron for each instance of the training data with a Gaussian function centered at the respective instance. For PNN the summation layer weights are determined by loss ratio of individual classes and for GRNN the weights are the target values of individual instances (patterns). It is apparent that GRNNs are actually very similar to a subclass of RBFN. Both types of networks perform relatively well, better than MLFNN and RNN. Descriptive comparative application of GRNN is in [Leung et al., 2000], while PNN-based classification is compared to RNN in [Saad et al., 1998].

TDNN, *DBNN* are two other types of ANNs that, although applications in financial time series exist, are not so common and are left for reader's own research.

The basic type of *Support Vector Machine* (SVM) is a binary classifier model based on constructing separation hyperplane that maximizes overall margin from the training instances. Although not a neural network, SVM is often mentioned in connection with ANNs, since it can be expressed as a structure of a multilayer perceptron. Further work on SVM enabled it to provide different outputs, such as non-linear regression. In most literature it outperforms all the ANN types described, while requiring less parameter tuning. Comparative application on financial series prediction was done in [Tay and Cao, 2001], [Abraham et al., 2003] or [Huang et al., 2005].

Combining several instances of a particular method or even of several methods is a popular way to improve performance in many machine learning tasks and this is also true for ANNs and SVMs. Mixture of experts and Ensemble are two well-known methods in the area. In both cases several well-performing networks are combined - either by precalculated weights (Ensemble) or by additional ANNs called gating networks (Mixture of experts). The Ensemble weights are most commonly calculated based on individual networks' average error or error variance. [Yümlü et al., 2003] compares Mixture of Experts to several standalone ANNs. Several options for Ensemble weighting, Mixture with RBF gating network are compared along with standalone ANNs in [Yu et al., 2008]. General methods of bagging and boosting are also applicable.

2.5 Performance evaluation

In order to be able to make conclusions from a certain prediction setup and used parameter set it is necessary to measure the usefulness of the generated predictions. This usefulness can be either measured on testing data by comparing the prediction with the real values of the attribute we predicted or by applying the prediction to its use. In case of our testing environment, this means to perform a simulation of trading based on the predicted values.

There are certain specifics to time series prediction, namely the fact that observations are ordered by time. Not respecting this property during training and testing (e.g. by using stratified sampling for splitting training and testing data) phases would invalidate the performance measurement, because the prediction method executed on an observation could act based on information not possibly known at the time of the observation. Therefore, evaluation of time series usually relies on one of the following two approaches - split validation using linear sampling or sliding window validation.

Split validation on the dataset, where the splitting is done by linear sampling simply takes the older observations from the dataset for training and the newer for testing. The split is usually done around 75% / 25%. The sliding window validation starts with training and testing windows of predefined sizes. At the beginning the training window is placed at the start of the time series data, with the testing window placed right behind it. After each training/testing cycle, training and testing windows are shifted forward in time. Optionally, the training window can be increasing in size rather than shift, meaning more training observations are available in each cycle.

Sliding window validation is closer to a probable real-world application of prediction techniques where the algorithm would be periodically re-trained. It will also provide more stable results. Conversely, the split validation may yield less balanced results. However, it may be better at showing how well the algorithm generalizes the behavior of the series, since with the same size of the training data it is required to predict further into the future. Hence it may tend to overestimate the prediction error.

Many measures are available to assess the prediction quality. Apart from division by their application - for regression or classification tasks - every method has its pros and cons and picking the correct one is tightly dependent on the requirements of the task itself.

Among the candidates for evaluation of the regression task in our environment, these are the most relevant ones: Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) and Mean Normalized Absolute Error (MNAE). RMSE's disadvantage is that its value scales with the range of the predicted value. This range may change with different normalization and discretization methods, making it hard to compare their contribution. While MAPE mitigates this issue by normalizing using the target value, exactly because of this it gets into problems when the target values are 0, which is a possible case when the value expresses differences between subsequent observations. MNAE performs the normalization using a different value

extracted from the data - it may be an average of the target value or a value for each observation predicted by a selected baseline algorithm.

To judge the performance of the classification task, we have focused on the basic Accuracy measure, Weighted Mean Recall and Weighted Mean Precision. The major disadvantage of the Accuracy measure is its predisposition to display unjustified good results if the distribution of the target classes is unbalanced. Hence usage of weighted recall and precision to crosscheck the validity of results would be necessary. The next option would be to use the F Score, which is based on precision and recall - in fact, the F Score could utilize the weighted variants of both.

While synthetic measures provide enough information about the precision of given predictions, they do not show how the imprecisions could affect the practical application. In case of financial time series this means the trading decisions. To investigate this, an evaluation based on simulating trading using a certain trading strategy needs to be performed. The selection of the trading strategy is crucial in this process. It needs to provide a compromise between the real world of market trading and the influence of the predictor itself. If a very sophisticated strategy using e.g. pair trading methods to minimize an impact of incorrect decisions would be used, the weight of the prediction itself could be so small that distinguishing between slight difference of two prediction methods would be made impossible.

Hence reasonable, yet straightforward strategies seem the best choice for the evaluation process. A basic strategy for regression task could e.g. simply act on a certain percentile of best predictions (highest returns) and enter the market when such prediction occurs. Exiting the market could then be performed e.g. when a negative market return has been predicted.

It is apparent that the evaluation needs to take into account things like transaction fees, price slippages or eventually order delays and partial order fills.

Chapter 3

Design

October: This is one of the peculiarly dangerous months to speculate in stocks. The others are July, January, September, April, November, May, March, June, December, August and February.

Mark Twain

Based on objectives set out in section 1.2 we will design a testing environment for financial predictors that will allow to execute experiments with minimal effort, but still provide enough flexibility to allow altering the experiment in any manner.

Apart from the functional demands we have specified in the aforementioned section, there are two structural goals we need to keep in mind:

- Steps of the prediction process need to be designed as components which are truly independent of each other and will function regardless of previous processing steps.
- The process needs to be easily modifiable and understandable, so that any hypothesis can be quickly introduced into the process and verified.

In this chapter, we will decide which environments will be used to build the project, lay out the general structure of the project, shape the internals of individual components and conclude with description of the tools we have decided to use for each part during the design.

3.1 Software environments evaluation

In order to be able to commence the design of our testing environment, we first had to decide on the tools using which we would like to implement our project, because their selection will strongly influence the structure of the project.

As indicated in section 1.2 the project will be based on combination of two software environments - a computing one (such as R, MATLAB, Mathematica) and a machine learning one (such as RapidMiner, WEKA or Statistica). Since there were several possible candidates we have tested each one of them, reviewed its capabilities in the respective areas and integration abilities with possible counterparts.

For computational environments we explored capabilities in the areas of data acquisition and manipulation; financial time series preprocessing and quantitative trading. For machine learning tools we evaluated breadth of available ML, preprocessing and evaluation methods, support for time series and suitability of the modelling paradigm of the tool for our purpose.

The evaluation has shown that the best choice will be a combination of R computing environment [R Development Core Team, 2011] and RapidMiner [Mierswa et al., 2006]. R language has extensive record of use in the financial area and has a wide range of specialized packages for this purpose. RapidMiner has a sufficient collection of ML algorithms and allows intuitive construction of complex processes. A recent addition to RapidMiner - R Extension allows convenient integration with the R language.

3.2 General structure

The environment will provide choice from multiple methods or approaches in each data mining step. Every method also has parameters, which need to be centrally configurable in order to allow easy experimentation and also to facilitate the optimization process. These requirements lead to a completely componentized process, where inputs and outputs are the same regardless of the internal behavior of each component.

The main components of the process will be the following:

- **Data input** will provide the predefined datasets either from a local store or an online source.
- **Preprocessing** should perform cleansing and shaping of the data, add derived technical indicators and calculate the target value for each observation.
- **Prediction** is going to train the selected ML method on the training data and apply the learnt method to the testing data. It will also perform accuracy evaluation of the prediction.
- **Trading** will perform simulation of trading using a selected strategy based on the generated predictions. One of the statistics of the trading session will be used as the output performance.
- **Logging** will be responsible for storing the results into files. The results will contain not only all output data created by the process, but also all input settings necessary to reproduce the experiment.
- **Parameter loading** shall obtain settings either from an external input or will provide default settings. These will be propagated to all components and also logged with other results.

- **Optimization** will not be a component in the process itself, but a separate process encapsulating the main one. It will set the input parameters and evaluate the performance metrics provided on the output.

In addition, auxiliary code for settings propagation through the process and for interaction with the optimization will be necessary.

We expect to use the R language for data input and more complex parts of processing and trading, whereas RapidMiner will be used as the main development environment connecting the components and for the machine learning prediction itself.

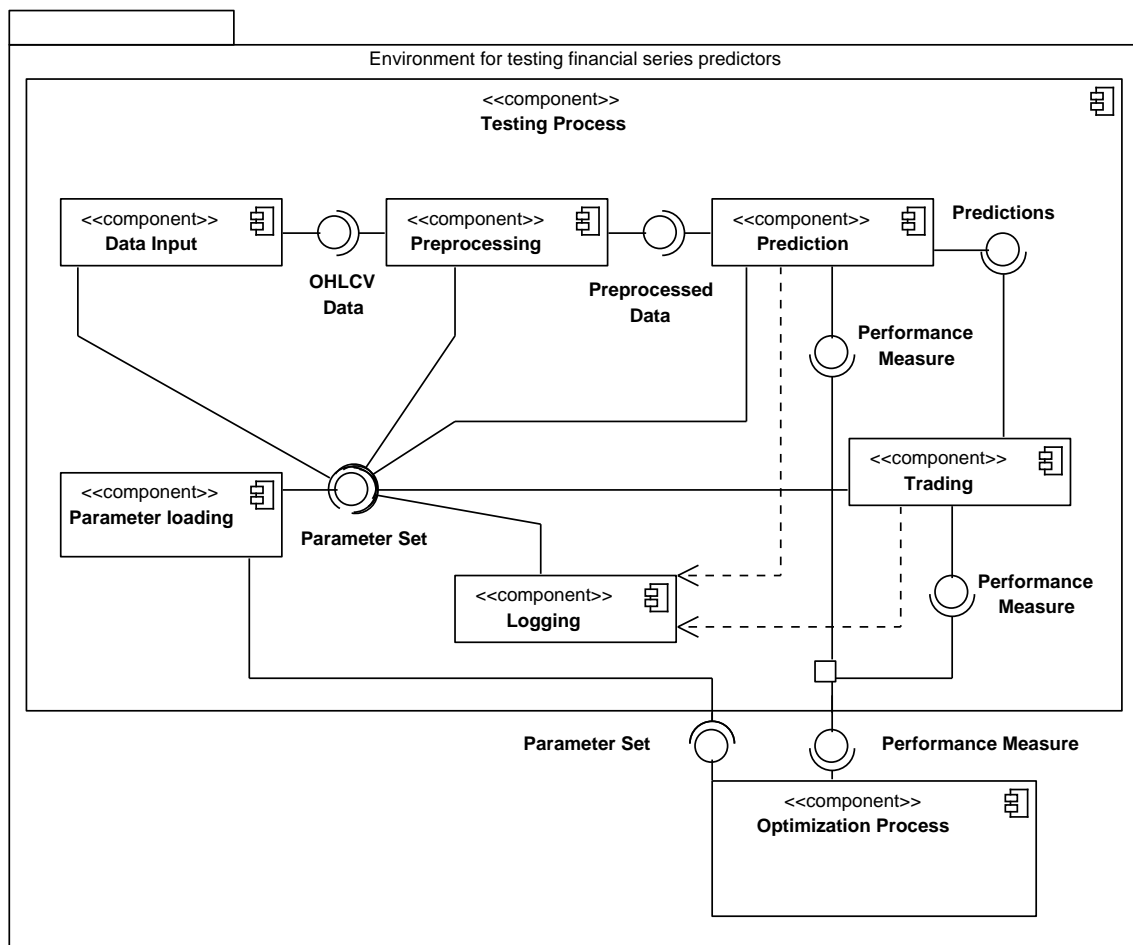


Figure 3.1: Component diagram of the prediction environment. The main prediction process has a performance metric output and a parameter set input. This allows the optimization process to attach itself and employ general optimization methods without the knowledge of internal workings of the prediction process.

3.3 Data input

The only data input to the process (except parameters) will be the dataset of market prices. The shape of the data always remain the same, regardless of the type of the market - be it company stock, forex instruments or commodity futures. The data input component should materialize the dataset either from a local store or an online service and provide it for processing.

3.3.1 OHLCV representation

The common representation of the data has a constant time frequency (i.e. minutely, hourly, daily) and includes five basic statistics for each observation at this frequency. These statistics, commonly abbreviated as OHLCV, are:

- **Open** is the price at the beginning of the current observation.
- **High** determines the highest price during the course of this observation.
- **Low** is the lowest price during the course of this observation.
- **Close** is the price at end of the current observation.
- **Volume** is the amount of items (e.g. stock, futures contracts) that have exchanged hands during the observation.

The OHLCV data are not the real representation of how the exchange markets work, but rather an extract of its behavior adapted for constant frequency. Data representing the precise events of the market operation - a listing of individual ask/bid orders being placed - are also available. However, this data isn't available freely, it is quite voluminous and in order to be able to take advantages of patterns extracted at this time scale, large volume transactions need to be executed with very short time delays. We also believe that operating on this scale is not particularly suitable for machine learning methods we would like currently to focus on.

Therefore, the format of datasets used by our environment will be OHLCV.

3.3.2 Datasets

The component will contain several predefined datasets. Selection of a dataset will be performed by a parameter from process' parameter set.

The probable composition of prepared datasets will be: multiple daily stock markets (probably of technology companies), at least one minutely observations dataset and some commodity futures instrument. Datasets should contain rather larger number of instances. If less data would be preferable in the learning process, it can be filtered using mechanisms in the Preprocessing component.

Offline datasets should be stored either in a database, standard file format such as Comma Separated Value or in RapidMiner's (RM) repository infrastructure. Since all of these options are very easily accessible from RM, we should evaluate ease of deployment and any possible advantages during the implementation phase.

Online data can be obtained using HTTP protocol from services such as Google Finance or Yahoo! Finance. Interface to these service is provided by `quanstrat`[Ryan, 2011] R package described in subsection 3.11.2.

There are several particulars for each class of instruments that require additional transformation steps to be performed on the dataset values in order to give them as unified meaning as possible. For stock markets data, one needs to consider two events: stock splits and dividend payouts. A stock split occurs whenever the number of shares on the market is increased. Since the price on the market is expressed per share and the company value remains the same, the price jumps when the split is made. In order to avoid major disproportion in comparison to prices before the split, the prices are adjusted based on the ratio of the new stock. The second event is a dividend payout. Dividends are paid to the owner of the stock, therefore stock's price gradually raises until the pay day. This distortion also needs to be adjusted for.

For futures trading, the situation is slightly more complicated. Futures markets usually express the market movements in points rather than a currency. Hence it is suitable to perform conversion into prices. The second issue is the manner in which futures contracts are traded - by using leverage. When a trader buys a contract, he doesn't pay the whole price of the contract, but deposits a margin to his broker company. However, when he closes the transaction, he receives the difference in price for the whole contract. In order to make trading stocks and futures comparable in our environment, we have decided that our simulation of trading will not use the leverage mechanism, but will allow to buy an arbitrary fraction of a futures contract for the appropriate fraction of the price.

3.4 Preprocessing

This component together with prediction is the centre of the testing environment. It will perform all the steps between loading the data and learning the predictor. This includes: initial shaping, addition of secondary data, normalization, transformation into instance space and several others. The initial parts of the processing will be executed in R language, while the instance space transformation and onward steps will be in RapidMiner.

3.4.1 Initial shaping

The first step with the dataset from the Data input component is to reduce its size based on a designated parameter. This may be desirable for those machine learning algorithms, whose performance varies notably depending on the amount of the learning data.

The next step is to perform detrending of the data, that is extracting relative values from the absolute prices. The common choice is calculating an arithmetic return (subtracting the previous observation's value for the feature from the current value) or a logarithmic return. For the volume value, a rate of change between observations or division by a moving average are the options.

3.4.2 Indicators

Although we will not exercise any technical trading methods manually, we expect that if a phenomenon based on the technical analysis will be showing prominently enough in the time series, the prediction algorithms should capture it in their prediction. In order to facilitate this, we need to provide the necessary data, in which the phenomenon is supposed to be easily discoverable. In the case of technical analysis this data are the technical indicators. Following are some of the well-known indicators, which should be included in the dataset:

- **Average Directional Index** (ADX) is supposed to express the strength of a trend in the market and its assumed direction.
- **Moving Average Convergence Divergence** (MACD) is also used for spotting trends' direction as well as momentum.
- **Relative Strength Index** (RSI) estimates the market's strength based on the ratio of up and down observations. Based on this ratio, the proximity of a turning point in the market's behavior may be estimated.
- **Bollinger Bands** (BBands) is a straightforward indicator, which creates a corridor around the prices based on Moving Average. This corridor should provide relative measure of fast rise or decline.

Since most of the indicators are relying on a certain amount of previous values, some of the features cannot be calculated for observations at the beginning of the dataset. While replacement by values from similar observations is sometimes practised, we have decided to simply remove any observation which contains an NA value.

3.4.3 Target value

Determining what should constitute a target value (The value which will be predicted) is one of the most crucial decisions. The first choice in this matter is regression versus classification. One may perform better than the other depending on other parameters and the data itself. While an exact value may not be necessary for trading, the real number continuity may allow for significant nuances to be seen by certain prediction algorithms. On the other hand, a nominal target value in classification clearly determines what each training observation means and allows the learning mechanism to strictly shape the prediction method.

For the regression task's target value, we plan to use the arithmetic return of Close price in a predefined number of observations, i.e. the price change in e.g. 5 days from the current observation. By using a higher number we hope to eliminate some of the noise of the series. We expect this target horizon to be in direct proportion to the number of previous observations, from which the prediction needs to be made. Exploring the relation between the target horizon and the embedding dimension (subsection 3.4.5) is something we would like to focus on during the testing of the environment.

In the classification task we will have 3 categories:

- **Up** - The price will substantially increase.
- **Neutral** - The price will not change relevantly in neither direction.
- **Down** - The price will substantially decrease.

Several options present themselves for defining what will be considered a substantial change. One is local extrema (i.e. peaks and valleys) detection on the absolute Close price values. The detection will probably need to use a threshold of minimum change from the previous value to consider an observation extremum as the movements of the series are quite erratic and we would like to limit the number of observations in up/down categories. The second option is to assign the categories based on quantiles of the return values distribution. The Up category would be assigned to 20% of highest return values, while the Down category will be assigned to 40% lowest return values. Alternatively, the Down category could be assigned to any negative return value.

By this stage of preprocessing the dataset will have all features we have decided to include and the target value will have been assigned to all observations.

3.4.4 Normalization / Discretization

Normalization is a standard step in most data mining efforts. It is necessary to unify the scale and usually also the mean of features in a dataset in order to obtain reasonable performance from many if not all ML methods. Some of the normalization methods are purposely non-linear, to enhance small but relevant differences in values and downplay outliers' impact, which may be important for our series of returns. Although any normalization method will help to improve the prediction performance and differences between them are often marginal, we would like to support multiple methods and in future analyze their influence on performance. We have selected the following popular normalization methods to include in our environment:

- **Softmax using logistic sigmoid** normalization, which is increasingly non-linear from the center towards the ends. It normalizes values into $(0; 1)$ interval.
- **Softmax using hyperbolic tangent** normalization, which is also increasingly non-linear. It normalizes values into $(-1; 1)$ interval.
- **Min-Max** normalization is linear and normalizes values into an arbitrary interval. We will probably use $(0; 1)$.

Although discretization is usually used to transform numeric values into nominal ones, it can be also applied in order to remove noise and help an algorithm to spot certain patterns more easily. We will include the following two discretization methods to experiment with:

- **Equal frequency** discretization will create intervals, whose width will be adjusted based on the distribution of values.

- **Equal width** discretization creates intervals of equal width within the range of values.

Both normalization and discretization will be optional - the settings will allow not to use them at all.

3.4.5 Instance space conversion

Conversion into instances is done by creating delay vectors from the series (often called windowing). Currently, we will use a fixed delay of 1 observation, i.e. a delay vector will contain every single previous observation up to the number specified by the embedding dimension. The embedding dimension will be parametrized and will probably be one of the frequently iterated parameters during optimization.

Since our time series will have many features (variates) after the preprocessing step, we will need to be able to specify, for each feature, whether it should be included in the delay vector (E.g. preprocessing adds moving average of stock's volume to every observation for later instance filtering purposes. We do not wish this value to be present for every previous observation in the delay vector). This will likely be solved by creating a feature naming convention that will allow to distinguish such features.

As has been mentioned above, we will support filtering after the instances have been created. Currently, the only supported filtering will be based on moving average of the volume. This filter will help to remove low liquidity periods in the market time series - these periods do not provide representative market behavior. In addition, we will filter delay vectors of observations from the beginning of the time series, which contain NA values due to insufficient number of previous observations in the series.

3.4.6 Component analysis

One of the more sophisticated groups of statistic methods used in data mining is component analysis. These methods allow to transform the existing features into a lower number of new, highly uncorrelated ones while minimizing the loss of information. They can help the data mining process by reducing the instance dimension and emphasizing the major differences between instances in the new features. We would like to include the following two representatives of the method:

- **Principal Component Analysis (PCA)**
- **Independent component analysis (ICA)**

3.5 Prediction

Once the data is completely prepared, it will be separated into training and testing partitions and transferred into the Prediction component. At present, a split validation with linear sampling will be used as the splitting/validation mechanism.

This component will execute training of a machine learning method selected using the corresponding parameter of the environment. Once the training is completed, the learnt representation of the method (model) is used for prediction on the testing data.

Although there are very many appealing ML methods, we have selected just few of the best known to be initially included in the environment. As emphasised throughout the whole design of the process and by virtue of RapidMiner's architecture, adding new methods will be very easy. The methods that will be in the process now are:

- **Support Vector Machine**(SVM) actually denotes multiple methods for classification and regression (Support Vector Regression) based on common theory. SVMs produce very good results in many ML applications, require only few parameters' tuning and have fast training. For more details, see section 2.4.
- **Artificial Neural Network**(ANN) is a classic method in ML, which can be used for both types of the prediction task. However, it requires considerable amount of parameter tuning and experimentation with unwarranted results. More details are also in section 2.4.
- **k-Nearest Neighbors**(kNN) is a lazy-learner method, i.e. it does not require any learning and the training data are not used until a prediction for a specific instance is requested. It has only few parameters for tuning and provides relatively good results. Detailed description is presented in section 2.3.
- **Input Clustering + group of ANNs** is an experimental method we set out to experiment with. It combines clustering of the input dataset and multiple neural networks. Details are provided in section 3.10
- **Generic R method**. In order to allow easy integration of prediction methods created in the R language, we will prepare a template structure, where the only required step will be to implement the algorithm in R into a predefined function.

The methods that can support both regression and classification should determine the required task automatically and behave accordingly. However, there are methods that are designed only for one of these tasks and these, if applied to the incorrect type of task, should show an error describing the situation.

Evaluation of prediction performance using standard accuracy measures is carried out within the Prediction component as well. Again, the appropriate metric needs to be selected depending on the type of the task. Details of used metrics are in section 3.7.

Output of the prediction component is the testing partition with a prediction assigned to each instance and the accuracy metric calculated from the predictions.

3.6 Trading

This component will perform a trading simulation on the testing data. The trading will be governed by a strategy, which will base its decisions on the predicted val-

ues. Actions yielded by the strategy are virtual market orders, which are evaluated approximately as they would be by the market (i.e. a brokerage company). This evaluation is projected into a virtual trading account.

The simulation will iterate through the data's time period and execute the strategy on every observation. A standard trading strategy consists of signals and rules. Signal is a definition of a condition, which determines whether the signal should be triggered for the current observation. In technical analysis, such condition would be usually linked to an indicator. In our case, the signals will be attached to the predicted value. Strategy's rules act based on the generated signal and decide the type of the issued market order and its values.

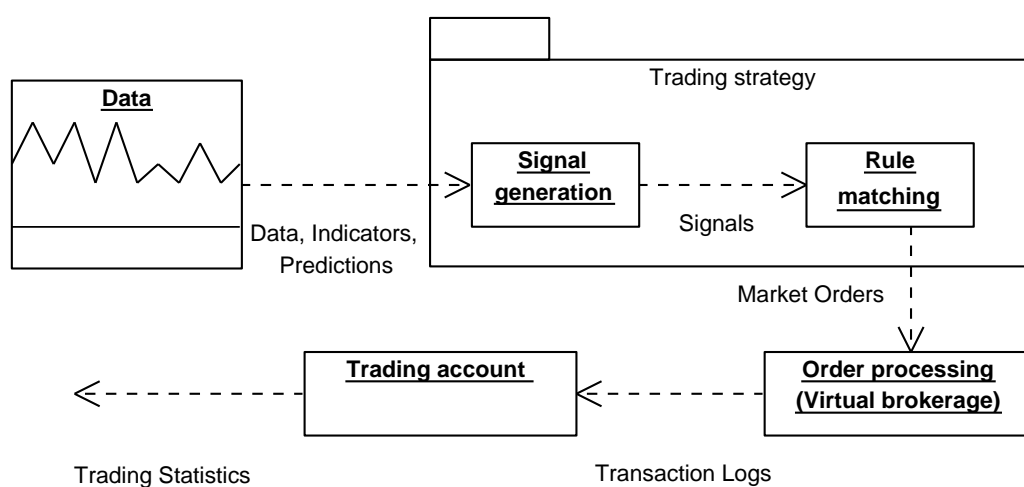


Figure 3.2: Diagram of the framework for trading strategy simulation. Information in the trading account will be used to generate the trading performance statistics after the simulation.

Different signals will be necessary for regression/classification task. For the sake of simplicity, we will create independent strategies, each with a specific signal triggering logic.

Currently, our environment will contain one testing strategy per prediction task and a baseline strategy:

- **Simple signal-based regression strategy** will have signals based on minimal and maximal thresholds. Values of these thresholds will be derived from the target value distribution in the training data using quantiles. The quantile values will be parameters of the process. A buy signal will be emitted whenever a prediction is over the max threshold and the sell signal whenever below the min threshold. Rules will react to these buy/sell signals, but will be subject to a limit of maximal amount of invested capital. This limit will also be parametrized.
- **Simple signal-based classification strategy** will have its signals directly linked to the predicted class. A buy signal will be triggered whenever the target class is "Up" and likewise, a sell signal will be generated by the target class "Down". Rules act in the same manner as in the previous strategy.

- **Buy and Hold strategy** this strategy serves only for benchmarking purposes. It doesn't make use of predictions at all, but issues a buy signal on the first day and a sell signal on the last day of the trading period. The issued buy order is for maximum allowed amount of invested capital.

Whenever an order for entering the market is being issued, the strategy needs to decide the quantity of the order - how much stock, futures contracts, etc. will be bought. We intentionally will not use any advanced method of order sizing and will make all orders have a constant price. This price will be a parameter of the process and the quantity of the order will be calculated simply as

constant_order_price/instrument_price

The trading component will produce several types of data and multiple statistics that will be recorded by the Logging component. Its only output returned to the process will be a selected performance metric.

3.7 Performance evaluation

Evaluation of performance will not be unified into one dedicated component, but performed within the Prediction and Trading components. The reason is that calculation of the metrics is tightly bound with the prediction and trading simulation processes.

RapidMiner is rich as to the available accuracy and error measures for prediction evaluation. Likewise, the R environment and used packages provide many useful metrics. We would like to use the following measures in our environment (N is the number of observations, O_k are P_k values of k^{th} observation and prediction, respectively):

- **Root Mean Squared Error (RMSE)** is a standard non-normalized error measure defined as $(\frac{1}{N} \sum_{k=1}^N (O_k - P_k)^2)^{\frac{1}{2}}$
- **Mean Absolute Percentage Error (MAPE)** is an error measure based on normalizing the error by the observed value: $\frac{1}{N} \sum_{k=1}^N (100 \frac{O_k - P_k}{O_k})$
- **Normalized Absolute Error (NAE)** is the total absolute error normalized by the total error if an average would be predicted: $\frac{\sum_{k=1}^N (O_k - P_k)}{\sum_{k=1}^N (O_k - \bar{O})}$
- **Accuracy** for classification task, which expresses the ratio of correctly classified instances.
- **Mean recall and precision** for every class, which will allow us to judge the success on individual classes (directions of prediction).
- **Several statistics of the trading simulation** that are common when assessing trading performance: net profit of the trading strategy, median profit/loss of a trade, percentage of successful trades, profit factor of trading, maximum drawdown during the trading session, average ratio of winning and losing trades.

All metrics are logged during the course of an experiment. However, a single main metric has to be selected as the output of the process. This output measure will be used by the optimization process to judge the quality of tested parameters. The main metric is selected by a parameter of the process.

3.8 Logging

Several files need to be created with every executed experiment. These files contain all the necessary parameters to reproduce an experiment and all output data of the process. This makes it possible to assess the quality of the process setup without having to execute it again.

All parameters of the process should be stored in a readable format. From the Prediction component, we should store the performance metrics calculated from the prediction as well as the generated predictions. The Trading component will log all trading account statistics and create a graph visualising the trading, including cumulated profit. Also, since most of the Trading component will be in the R environment, we should store the inputs - the prediction data and parameters - in the native R format. Later on, we can load this data and experiment with the trading strategy only, without the need to generate the predictions again.

3.9 Optimization

Optimization of the prediction process parameters will probably be the most frequent use case of our environment. We have decided to structure it as a separate process of RapidMiner. This process will contain the optimization algorithm, which will in turn execute the prediction process and supplies it with a complete parameter set. This parameter set will be used inside the main process instead of the default values.

The user has to be able to choose which parameters should be optimized and what values should the algorithm try.

Every iteration of the optimizer loop will yield the tested parameter set and the resulting performance. These values should be logged into a single table and stored into a file, which will afterwards give a clear view of individual parameters' influence.

Possible choices for the optimization process are Genetic Algorithm or Grid Search.

3.10 Clustering + Neural Networks group

During the research for this thesis, we came up with a machine learning setup we were interested in investigating. In order to illustrate how a custom prediction method can be easily created in the environment, we will add this method into the initial version.

The hypothesis of this method is that an Artificial Neural Network (ANN) may perhaps give better predictions if all the instances it has to deal with were more alike. This means both the training data and also the instances for which a prediction is requested.

Therefore, all the data could be divided into clusters and a separate ANN assigned to each cluster. A network would be trained only on data from one cluster and then predict only on instances assigned to this cluster.

The clustering algorithm would be executed on the training data. The resulting clusters should then remain fixed and the testing data will be assigned to these fixed clusters. Not all clustering algorithms are capable of operating in such a way, but e.g. k-Means is able to accomplish this.

Since every instance is predicted by only one of the networks, there is no combining of multiple results. The prediction is just drawn from the corresponding network.

Although similar to bagging and boosting, this method separates the instances before the actual training begins and the separation is based on instances' similarity.

Currently, the method will use a fixed number of clusters and networks. Should testing show any promising results, a variable number of clusters is the apparent first step in extension of this method.

3.11 Platform

As described in section 3.1 we will build our project on a combination of RapidMiner data mining tool and the R computing environment.

Both the main prediction process and the optimization encapsulating process will be designed in RapidMiner. Operators in the process will largely correspond to the division into components in this chapter.

The R environment will be used for the online data input, preprocessing and trading and will be embedded as a subcomponent - it won't interact directly with the other main components.

3.11.1 Rapidminer

RapidMiner [Mierswa et al., 2006] is a comprehensive machine learning and data mining software. The main concept of the design process in RapidMiner is an operator. Every processing unit, be it a data input, predictor or a performance evaluator is an operator. Every operator can have one or more inputs and outputs, which connect to other operators. In most cases, the input and output is a dataset, eventually it can be a trained machine learning model, a performance vector or one of the several other less used data types. The ease of design thanks to this concept is, apart from the broad repertoire of ML methods, the main advantage of this software.

Several extension have been developed for RapidMiner. The most important for this project is a recently created R Extension. This extension is what allows

RapidMiner's integration with R environment. The extension introduces a new operator - R script. This operator can contain arbitrary R code, which can process inputs (which are transparently marshalled into R structures) given to the operator. Once the R script is finished, its results are propagated to the output of the operator (again, transparently marshalled to RapidMiner data structures).

3.11.2 R environment and packages

The R language and environment [R Development Core Team, 2011] is a popular statistical and data analysis platform. The design of the language and broad base functionality along with large number of additional packages created by a strong community around the tool make it a perfect choice for financial data analysis and experimentation.

There are several packages from the financial analysis area we will be using. We will focus only on the functionality used by our project, although some of the packages provide much more than mentioned.

Package "quantmod" [Ryan, 2011] adds a set of methods for managing and manipulating with symbols - individual instruments and their OHLCV time series. It allows to download symbols' historical data from online sources, which we make use of in the Data input component. "TTR" [Ulrich, 2011] package contains many technical indicators, some of which we add during the preprocessing step. "dprep" [Acuna et al., 2009] package contains several normalization and discretization methods we optionally apply during preprocessing. "blotter" [Carl and Peterson, 2011] package provides the basic infrastructure for financial simulations, it introduces the notion of a portfolio, trading account and trading transaction. "quantstrat" [Carl et al., 2011] adds trading simulation using strategy as described in section 3.6

Chapter 4

Implementation

In most fields of research, when someone makes an important finding, they publish it. In the case of prices, they set up a firm and sell advice about their discovery.

Benoît Mandelbrot

Based on our detailed design charted out in the previous chapter, we have implemented the testing environment. In the following sections we will describe relevant implementation aspects of each part of the project.

Most of the components depend on one or more configuration parameters. Names of these parameters are provided in the text. However, for a complete overview of all parameters including possible values and precise behavior for each value, see section 4.9 at the end of this chapter.

In order to facilitate central configuration of process parameters, we had to devise a method to propagate these parameters through the RapidMiner(RM) process and into the R code. Inside RM we have resolved this using macro support. Macros are set up in a single operator and then can be referenced from any other operator in the process. The propagation into the R code was resolved by creating a key/value dataset, which is then sent into the R functions.

Although the RM and R portions of the project interact closely, we have tried to keep them as independent as possible. Specifically, we have structured the R code to depend minimally on RM process. This allowed us to implement the R preprocessing and trading code without use of RM and then simply link this code from R Script operator of RM process. Linking of the R code in the operator is done by including the main R file of the project, calling a necessary conversion function and then invoking the desired function (e.g. strategy simulation).

4.1 Data input

This component outputs OHLCV dataset of one of the available instruments depending on the value of *input_dataset* parameter. Two modes of operations had to

be implemented:

- **Local storage of dataset** for datasets that are included with the environment. We have decided to import the original data into RapidMiner(RM) repository and use a Retrieve operator to load this data.
- **Download of symbol data from an online service** will use a method from "quantmod" R package to download historical daily data from Yahoo! Finance web. We have also implemented a caching mechanism that stores the downloaded data within the environment's data directory. The length of the downloaded data is set to 10 years (approximately 2500 observations).

Regardless of the selected option, this component outputs a dataset with identical structure.

4.2 Preprocessing

The dataset on input of Preprocessing operator is immediately forwarded to an R Script and initial preprocessing is done within the R environment. Here, the following steps are performed:

- The dataset size is trimmed based on the *max_dataset_items* parameter - only that many most recent observations are kept.
- Technical indicators ADX, MACD, RSI and Bolinger Bands are added using the "TTR" package.
- Open, High, Low, Close columns are converted into returns (difference from the previous observation). Volume is converted into series of rate of change quotients.
- The target value is assigned based on *target_value* - numeric calculation for regression or one of the category assignment methods for classification.
- Normalization and/or Discretization are applied depending on *normalization*, *discretization* and *discretization_num_values*.

Afterwards, the preprocessed dataset is returned back into the RM process and several more operations are executed:

- The observations are transformed into delay vectors using the Window operator, which depends on the *window_size* parameter.
- Optional filtering based on moving average of Volume is applied (governed by parameters *filtering*, *filtering_volume_ma*)
- The instance dataset is separated into training and testing partitions using linear sampling with 0.8/0.2 split.
- Optional component analysis (either PCA or ICA) is applied as per *feature_set_reduction* and *num_components*.

The final dataset, which may now vary vastly in the number of features, value ranges and even in the type of the target value column, is forwarded into the Prediction component.

4.3 Prediction

The Prediction component executes one of the five available machine learning methods depending on *prediction* parameter . So far, all used implementations support both regression and classification. The present methods are:

- **SVM** was added as an operator included in RM. During the course of creation of the environment we have experimented with multiple kernels of SVM - linear, radial and polynomial. For polynomial kernels, parameter *svm.kernel.degree* specifies the degree of the polynomial. However, in the last stages of implementation, we have been receiving consistently better results using dot product (linear kernel). This phenomenon is something a more extensive future testing will have to shed a light on.
- **ANN** is also an RM operator. We have set the feed-forward network to have two hidden layers, whose size automatically adjusts to the number of inputs according to a popular rule of thumb formula $(num_input + num_output) / 2 + 1$. Activation function is sigmoid (linear for regression).
- **kNN** operator is a standard Nearest Neighbor algorithm. The number of nearest neighbors is governed by *knn.num.neighbors*, weight of neighbors is relative to their distance, for which the Euclidean metric is used.
- **R predictor stub** is a set up framework for possible generation of predictions in R environment. The user only needs to implement the prediction method into function "prediction1" in source file "prediction.r".
- **Clustering + ANNs** - is our experimental setup, more details in section 4.8.

In RM the learning method's operator trains a model and then outputs it for future usage. To generate a prediction, the model has to be input along with the testing data into an operator for model application.

Once the predictions are generated, their quality is evaluated. This is done using Performance Evaluation operators that calculate all the metrics described in section 3.7

4.4 Trading

The trading simulation is performed entirely in the R environment. The evaluation function "eval_strategy1" receives the original dataset, the transformed dataset including predictions and settings. It returns the selected trading performance metric. Most of the financial-related steps are accomplished by functionality from packages "quantstrat" and "blotter". The steps taken in this component are:

- Merge the prediction into the testing period of the original data, since the trading framework needs the original prices and volume.
- Set up the correct strategy (controlled by *strategy* parameter and the type of the prediction task).

- Execute the trading simulation using the specified strategy.
- Update virtual account and portfolio with the results of the simulation and generate statistics.
- Create multiple logs based on the simulation. The logs are stored as files with a single timestamp inside the environment's predefined logs directory.
 - Create an R structure containing: the original data, transformed data with predictions, trading transactions, statistics and parameters of the experiment. Store it in a native R format, which can be easily loaded in the future.
 - Plot a graph of trading visualising the trading prices, moments of buy/sell orders and cumulated profit.
 - Store the trading statistics and parameters in a plain-text format for easier reading and possibly automatic extraction.

Finally, a metric is chosen according to *strategy_eval_metric* from trading stats and returned into the RM process

4.5 Performance evaluation

Since all metrics were calculated earlier, the only remaining task in performance evaluation is to decide whether the prediction metric or the trade metric should be used as output of the whole process. This again depends on the *strategy_eval_metric* parameter.

4.6 Logging

Similarly in logging, most necessary operations have been already performed in Prediction and Trading components. The last useful information that should be stored is the text representation of the transformed dataset with predictions. Although it has been already stored as an R structure, it is desirable to also have it in easily readable format. Therefore, the dataset is stored as a CSV file and the same naming convention with timestamp is used.

4.7 Optimization

Optimization is implemented as an independent RapidMiner process internally executing the main process. It is smaller than the prediction process and consists of only few general blocks.

Central configuration of parameters and their propagation through the process is done using macros as in the main process.

The optimization operator encapsulates most of the setup. We have used the Grid Search optimizer, since we will mostly want to go through all parameter combinations we specify rather than some of them. The optimization operator contains a separate macro operator for each parameter of the main process. When executing the optimization, the user selects which macros (parameters) should be tested and with what values. The main process is invoked through Execute Process operator, which also specifies all the macros that will be propagated.

A performance metric output from each execution of the prediction process is stored into a log and saved into a CSV file at the end of the optimization. This table then gives a very good overview of effects of the tested parameters on performance.

4.8 Clustering + Neural Networks group

As one of the prediction options we have implemented our experimental prediction setup consisting of clustering and multiple neural networks. In the current implementation the number of clusters and, in effect, the neural networks is fixed and its value is 4.

For clustering we used the common k-Means operator. The reason is that it allows to use the fixed clustering model created by the initial clustering on the training data to assign clusters to the testing data (This is not exactly online clustering, since the model doesn't modify itself when assigning clusters to the testing data instances).

Based on the assigned cluster, the instances are separated into smaller datasets and fed to individual networks. The neural networks have the same configuration as the single network used for prediction (section 4.3).

Sets of instances with predictions from networks are then merged back into a single dataset and this is the result of this prediction method.

4.9 Environment parameters

As could be noticed throughout previous chapters, the parameters are an important part of the process. In this section, we compiled descriptions of all available parameters with explanation of their every possible value.

- *input_dataset* - Determines which predefined dataset should be used. (default 3)
 - 1 - E-mini S&P 500 Futures minutely data traded on CME. The data starts at midnight of 2008/09/03 and ends on midnight of 2009/03/12. The dataset consists of 180941 observations.
 - 2 - IBM (International Business Machines Corp.) stock daily data traded on NYSE. The data is dynamically loaded on first request. The downloaded data represents 10 years ending with the current day. Therefore, the dataset contains around 2500 observations.
 - 3 - YHOO (Yahoo) stock daily data traded on NASDAQ. The data is dynamically loaded on first request. The downloaded data represents 10 years ending with the current day. Therefore, the dataset contains around 2500 observations.
 - 4 - Gold 100 oz. Futures daily data traded on CBOT. The data starts at midnight of 2004/12/29 and ends on midnight of 2011/05/04. The dataset consists of 1602 observations.
 - 5 - MSFT (Microsoft Corp.) stock daily data traded on NASDAQ. The data is dynamically loaded on first request. The downloaded data represents 10 years ending with the current day. Therefore, the dataset contains around 2500 observations.
- *max_dataset_observations* - Limit the maximum number of observations used in the process. Value of -1 means all observations are used, any other value X keeps only the X most recent observations in the dataset. (default -1)
- *target_value* - Determines the nature of the target value (the value that will be predicted). (default 1)
 - 1 - The target value will be difference (return) between the current price and the future price. The task will, therefore, be regression of returns in X time steps (e.g. days, minutes, etc.). The X - distance between the current and the future observation is determined by parameter *target_horizont*.
 - 2 - The target value will be a class based on detection of peaks/valleys in upcoming time steps. Three classes are present: "1" - a peak will occur in X days, "0" - neither peak or valley will occur in X days, "-1" - a valley will occur in X days. The X is determined by parameter *target_horizont*. The peak/valleys detections have a threshold - the peak/valley must differ at least by value Y/Z from surrounding observations. Y/Z is determined by parameters *target_peak_thresh/target_valley_thresh*, respectively.

- 3 - The target value will be a class based on detection of major difference (return) between current price and future price. This task is similar to $target_value = 1$, but will be performed as classification. Three classes are present: "1" - major positive return in X days, "0" - no major positive or negative return, "-1" - major negative return in X days. The X is specified by parameter *target_horizont*. The sensitivity, i.e. the minimum required return value is determined using quantiles taken from the returns distribution. These quantiles are given by parameters *target_min_quantile* and *target_max_quantile*.
- *target_horizont* - Determines the number of observations between the current and the future one (from which the target value is calculated). How the future value is calculated is determined by *target_value*. (default 3)
- *target_peak_thresh* - The minimal threshold for peak detection. For an observation to be considered a local maximum, it must differ at least by this value. Used with $target_value = 2$. Note that this value is very specific to the used dataset. (default 0.2)
- *target_valley_thresh* - The minimal threshold for valley detection. For an observation to be considered a local minimum, it must differ at least by this value. Used with $target_value = 2$. Note that this value is very specific to the used dataset. (default -0.3)
- *target_max_quantile* - This value determines the quantile from the distribution of the target value in the training data. The value corresponding to the specified quantile will be the minimal threshold for an observation to be considered a major positive return when determining the class for training instances (with $target_value = 3$). When the target value is numeric, i.e. the task is regression ($target_value = 1$), this value is used as a threshold for a buy signal in the trading strategy (see parameter *strategy* = 1). (default 0.8)
- *target_min_quantile* - This value determines quantile from the distribution of the target value in the training data. The value corresponding to the specified quantile will be the maximal threshold for an observation to be considered a major negative return. Use -1 to set the threshold to 0, i.e. any negative return will be considered a major negative return. Used with $target_value = 3$. (default -1)
- *normalization* - Specifies the type of normalization applied to features. (default 1)
 - 1 - No normalization is applied.
 - 2 - Softmax normalization is applied.
 - 3 - Sigmoidal normalization is applied.
 - 4 - Min-Max normalization is applied.
- *discretization* - Determines what kind of value discretization is applied to features. (default 1)
 - 1 - No discretization of values if performed.

- 2 - Discretization based on intervals with equal frequencies of values. The number of intervals is determined by parameter *discretization_num_values*.
 - 3 - Discretization based on intervals of equal width. The number of intervals is detected automatically using Scott's formula.
- *discretization_num_values* - The number of bins to be used for discretization. Used with *discretization* = 2. (default 5)
- *window_size* - This parameter determines the length of the delay vector. A delay vector is produced when converting the time series features into instances. The length, therefore, specifies how many historical values for each feature will be in a single instance. Value of this parameter significantly influences the prediction algorithm's success. (default 10)
- *filtering* - Allows to filter instances (delay vectors) which would negatively affect the performance. (default 1)
 - 1 - No filtering is done.
 - 2 - Instances are filtered based on moving average of Volume at the time of the last observation in the instance. This allows to remove low liquidity regions of the time series. The threshold for Volume MA is determined by parameter *filtering_volume_ma*.
- *filtering_volume_ma* - Defines the cut-off threshold for Volume-based filtering. Used with *filtering* = 2. (default 300)
- *feature_set_reduction* - This parameters allows to reduce the number of instance features, either by selection or transformation. Currently, there are no options for mere selection of features. Transformation reduces the dimensionality by replacing the existing features with a smaller set of new ones, which are more informative. (default 1)
 - 1 - No feature reduction is performed.
 - 2 - Principal Component Analysis is performed on instances and a predefined number of the most significant components is extracted. The number of components is determined by parameter *num_components*.
 - 3 - Independent Component Analysis is performed on instances and a predefined number of the most significant components is extracted. The number of components is determined by parameter *num_components*.
- *num_components* - Defines the number of components to be yielded by feature set reduction. Used with *feature_set_reduction* = 2 and *feature_set_reduction* = 3. (default 5)
- *prediction* - Selects the machine learning algorithm used to predict the target value. Currently, all algorithms can process both tasks - classification and regression. The task is determined by RapidMiner algorithms based on the type of the target feature, but should the need arise, could be determined by the *target_value* parameter. Note that some of the algorithms perform parameter optimization by themselves, therefore the processing time may vary vastly. (default 3)

- 1 - Support Vector Machine is used for prediction.
 - 2 - Artificial Neural Network is used. The network is a feed-forward, with two hidden layers and is trained by backpropagation. The number of neurons in hidden layers is computed automatically based on the number of input features.
 - 3 - Nearest Neighbor (kNN) algorithm is used. The predicted value is calculated from the target value of k nearest neighbors. Number of neighbors k is set by parameter *knn_num_neighbors*. The distance is measured by the Euclidean metric. The influence of each neighbor is weighted, i.e. determined by its distance from the predicted instance.
 - 4 - Predict using R. This choice allows prediction to take place in R and then marshals the results back into the RapidMiner process. The prediction implementation is to be put into method "prediction1" in the environment's R file "prediction.r".
- *svm_kernel_degree* - Specifies the degree of SVM's polynomial kernel. Used with *prediction* = 1. (default 5)
 - *knn_num_neighbors* - Determines the number of nearest neighbors involved in prediction using kNN. Used with *prediction* = 3. (default 10)
 - *txn_fee_size* - This parameter sets the cost for individual buy/sell orders. It is utilized when a strategy is executed on predicted values. (default -15)
 - *order_sum* - Specifies the size of each order when entering the market. The value is dollar volume of the order, i.e. if a stock costs \$10 and *order_sum* is 100, the order quantity will be 10.
 - *strategy* - Specifies the trading strategy executed on the testing data. The strategy is defined by rules executed based on signals extracted from the predicted values. (default 1)
 - 1 - Use plain buy on positive signal strategy, sell on negative. When the task is regression, buy/sell signals are based on return thresholds determined by quantiles in parameters *target_max_quantile* (for buy signal) and *target_min_quantile* (for sell signal). For classification tasks, buy signals are observations with class "1" and sell signals observations with class "-1". The order sizing and limits are governed by parameters *order_sum* and *limit_to_order_sum*.
 - 2 - Buy and Hold strategy. This strategy is independent on the actual predictions and serves only as a baseline benchmark. A single buy order is placed at the beginning of the trading period and kept for the whole duration of the trading. Every strategy should perform significantly better than this one in order to be relevant for further examination.
 - *limit_to_order_sum* - Determines whether the strategy is limited to buy at most *order_sum* worth of the traded instrument. (default 1)
 - 1 - A new order is placed only if no stock (or other instrument) is currently owned.

- 2 - A new order is placed whenever a buy signal occurs, regardless of currently owned amount of stock.
- *strategy_eval_metric* - This parameter defines which metric is returned from the prediction environment. The yielded metric is especially relevant when tuning parameters in the optimization environment. Available metrics are either synthetic accuracy measures of the prediction algorithm or statistics of the trading strategy application. (default 1)
 - 1 - The metric is based directly on the prediction algorithm accuracy. For regression task it is currently normalized absolute error (NAE) and for classification task it is accuracy.
 - 2 - The metric is the net trading profit/loss of the trading strategy.
 - 3 - The metric is the median trade profit/loss.
 - 4 - The metric is the percentage of positive (profitable) trades.
 - 5 - The metric is the profit factor. Profit factor is the sum of profits of positive trades divided by the sum of losses of negative trades.
 - 6 - The metric is the maximum drawdown (MDD). MDD is the highest drop in the accumulated profit through the trading period.
 - 7 - The metric is the average win/loss ratio. This ratio is calculated as the average winning trade size divided by the average losing trade size.

Chapter 5

Testing

*In this business if you're good,
you're right six times out of ten.
You're never going to be right nine
times out of ten.*

Peter Lynch

In this chapter we will demonstrate the functionality of our environment by putting it to practical use. We will perform rough exploration of the parameter space and analyze its results - compare the success of individual machine learning methods, assess the influence of the optimized parameters on the performance and evaluate the trading results.

5.1 General comparison

In order to get a notion of the quality of individual predictors and to see how the parameters affect the prediction, we tested combinations of all predictors and of those parameters that we believe are the most influential.

The first part of Figure 5.1 shows values of parameters that were not optimized during this testing and remain the same through the whole test. Their values have been decided based on previous experimentation. The lower section of Figure 5.1 shows optimized parameters and values that were tested.

We have chosen a rather "hard" dataset of Yahoo Inc. stock for this testing. This stock hasn't had any major positive trends during past years and its performance isn't very good. Furthermore, in recent years, which fall into the testing period of the dataset, it has seen several sudden drops and rises, on which the prediction method's resilience can be examined. Market movements in this time period can be seen in Figure 5.3. We believe that choosing more complicated data is superior to selecting a time series with strong positive trends, which might yield better performance and trading results, but whose significance, however, isn't very high.

The results of the testing are presented in Figure 5.2. Total of 145 parameter combinations were evaluated.

Parameter name	Value	Note
input_dataset	3	Yahoo! Inc. daily stock (2514 observations from 2001-05-09 to 2011-05-06)
max_dataset_items	-1	All 2514 observations were used
test/train split	0.8/0.2	Although this value is fixed in the process, it is noteworthy. This split applied to Yahoo! stock means 8 years of training data and the last 2 years used for testing data.
target_value	1	Arithmetic return in <i>target_horizon</i> days
target_max_quantile	0.7	Prediction above 70% percentile will be a buy signal
target_min_quantile	-1	Any negative prediction will be a sell signal
discretization	1	No discretization will be used.
filtering	1	No filtering will be used.
num_components	10	If component analysis is used, it will return the 10 most relevant principal variables.
knn_num_neighbors	8	The 8 nearest neighbors will be used in kNN prediction.
txn_fee_size	-15	Cost per transaction will be \$15.
order_sum	10000	Every buy order will have a value of \$10 000.
limit_to_order_sum	1	At most <i>order_sum</i> will be invested at one moment.
prediction	(1,2,3,5)	Tested machine learning methods will be SVM, ANN, kNN and our Clustering + ANN group
target_horizon	(2, 5, 10)	Returns in 2, 5 or 10 days.
normalization	(1, 4)	No normalization or Min-max>Returns in 2 or 5 days.
window_size	(5, 10, 20)	Embedding dimension 5, 10 or 20 days.
feature_set_reduction	(1, 2)	Either no component analysis or PCA

Figure 5.1: The upper section of the table shows the constant parameters of the experiment. Optimized parameters and the tested values are in the lower section.

No.	NAE	Trading Profit (\$)	Prediction Method	Prediction Horizon	Normalization	Embedding Dimension	Comp. Analysis
1	1.070	6541	SVM	5	None	20	None
2	1.303	5798	Cl. ANN	2	None	10	PCA
3	1.307	4113	kNN	2	None	20	PCA
4	1.410	3876	SVM	10	None	5	None
5	1.897	3663	Cl. ANN	2	None	5	None
6	1.152	3457	SVM	10	None	10	None
7	6.777	3399	SVM	10	Min-max	5	PCA
8	1.469	3387	kNN	10	None	20	PCA
9	1.075	3366	SVM	5	None	5	None
10	1.310	3215	kNN	5	None	20	PCA
11	1.420	3205	kNN	10	None	5	PCA
12	1.310	2854	kNN	5	None	20	None
13	1.202	2832	kNN	2	None	20	None
14	1.277	2762	kNN	2	None	5	PCA
15	1.607	2641	Cl. ANN	2	Min-max	10	None
16	1.298	2459	kNN	10	None	10	None
17	1.275	2294	Cl. ANN	5	Min-max	10	None
18	1.698	2242	Cl. ANN	5	None	20	PCA
19	2.148	2224	Cl. ANN	2	Min-max	5	None
20	1.824	2220	Cl. ANN	10	None	10	PCA
21	2.179	2172	SVM	2	None	20	None
22	1.149	2167	ANN	2	Min-max	5	None
23	1.241	2167	kNN	2	Min-max	20	None
24	5.317	2167	SVM	2	Min-max	20	None
25	1.104	2018	Cl. ANN	5	Min-max	5	PCA
26	4.663	1999	SVM	10	Min-max	10	PCA
27	1.528	1888	kNN	5	Min-max	10	PCA
28	1.343	1832	kNN	5	Min-max	20	None
29	4.704	1832	SVM	5	Min-max	10	None
30	2.119	1776	Cl. ANN	2	Min-max	10	PCA

Figure 5.2: The table contains parameter configurations of the 30 most profitable test runs ordered by profit. Column NAE contains value of Normalized Absolute Error. Prediction method Cl. ANN denotes our experimental method described in section 3.10

The results show that most of the configurations have rather sub par values of accuracy. The outcomes of trading give more promise for some parameter combinations. As can be seen, the accuracy measure is not a prerequisite for successful trading. Examination of the generated prediction data has shown that even if the predictions differ significantly from the target values, it is more important whether they deviate enough before buy/sell opportunities to be picked up by the trading

strategy.

Proportion of individual machine learning methods in the top 30 results is mostly balanced (SVM: 9, kNN: 11, Clustering+ANN: 9, ANN: 1) except the standalone Artificial Neural Network. This may be due to insufficient amount of experimentation usually necessary to obtain reasonable results from ANNs. Furthermore, the best result of ANN (No. 22) is achieved by just one transaction - a buy order at the beginning of the training period, see Figure A.3. It is, in fact, almost identical to the benchmark Buy-And-Hold strategy and hence should be taken as a baseline for a reasonable strategy.

Surprisingly, our method of clustering and using group of ANNs did cope quite well, although it uses ANNs too. The differences from the standalone ANN are that each network in our method is trained using less training data (since each network receives only one cluster) and higher uniformity of the training data (by virtue of clustering). These distinctions may make the difference necessary to give markedly better predictions.

Some known characteristics of each method are visible in the results. The SVM performs well with high dimensional instances - the best configuration is an SVM with embedding dimension of 20. This dimension combined with all features added during preprocessing results in instance vectors of dimension 280. Conversely, kNN operates best on limited number of highly informational features - hence most of kNN configurations in the results make use of Principal Component Analysis, which outputs instances with 10 features.

Unexpectedly, majority of the best configurations did not use normalization. This may mean that Min-max isn't the best choice normalization method for this task.

Most entries also exhibit an expectable relation between the prediction horizon and the embedding dimension (the number of available previous observations) - that the horizon is much shorter than the embedding dimension, usually less than half.

Figure 5.3 provides visualisation of the No. 1 configuration from the previous table. It is clearly visible that the prediction method is consistently able to foresee large price movements. Recall that the testing data are out-of-sample - the SVM was not trained on the testing period. For visualisation of the best configurations of the other trading methods, see Appendix A

5.2 Conclusion

The tested configurations of prediction methods did not yield very good absolute error-based accuracy results. However, some of the more successful configurations were able to regularly provide good indications of major market movements. Even our basic trading strategy was able to capture these indications and trade quite profitably. These trading results show that even without long fine-tuning, it is possible to adjust the prediction processes to learn from the market behavior well enough to provide a reasonable performance. This provides encouragement to conduct further research in optimization of the prediction process.



Figure 5.3: Visualisation of trading of the best configuration in the testing. Notice that the strategy executes rather a small number of high profit trades.

Chapter 6

Future work

Harder!

Sasha Grey

Our future work shall evolve in two main directions: expanding the testing environment and continually exploring combinations and parameters leading to an increasing performance of the prediction processes.

In the direction of expansion of project's features, we see the following candidates:

- Extended support of feature selection - possibility to individually select which indicators are added during preprocessing. This process should also be optimizable.
- Add more technical indicators.
- Sophisticated prediction approaches, such as predicting two values - e.g. an expected low and high of a certain time period.
- More advanced methods for assigning classes to instances in classification task.
- Possibility to combine multiple prediction methods using ensembles. Support for bagging and boosting.
- Alternative evaluation using validation data (i.e. separate the dataset into training, testing, validation).
- More predictors, such as Markov Models, Decision Trees or Inductive Logic Programming based methods.
- Alternative trading strategies.

We believe that increasing quality of predictions and trading will be possible only through persistent effort in experimenting, possibly with help of some of the future features mentioned above - such as the meta-algorithms for combination of multiple predictors.

Chapter 7

Conclusion

Rule No.1: Never lose money.

Rule No.2: Never forget rule No.1.

Warren Edward Buffett

In this work we have provided background on major machine learning methods by creating a review of existing literature with focus on the practical application to financial time series prediction. The review has shown that there is an active research in the area with interesting results.

Based on the gained knowledge we have designed a testing environment for experimenting with predictors on financial series. We have also introduced our experimental prediction method based on clustering and group of Neural Networks.

The testing environment was implemented according to the design using RapidMiner and R software tools. The environment is configurable and supports optimization of its parameters, which will be crucial for future usage.

To show that the implemented project serves well the purpose it was designed for and also to get a basic overview of the parameter space's influence on the quality of prediction and performance of trading, we have performed several optimization experiments and presented its results. This testing also included our experimental prediction method.

We believe that all of these goals have been accomplished successfully.

This work gave a lot to the author - initial insight into the area of financial predictive analysis and behavior of Machine Learning predictors, substantially expanded experience with R language and RapidMiner and last but not least, the testing environment itself, which he hopes to put into much use in future research.

References

- A. Abraham, N. S. Philip, and P. Saratchandran. Modeling chaotic behavior of stock indices using intelligent paradigms. In *International Journal of Neural, Parallel & Scientific Computations, USA, Volume 11, Issue*, pages 143–160, 2003.
- E. Acuna, , members of the CASTLE group at UPR-Mayaguez, and P. Rico. *dprep: Data preprocessing and visualization functions for classification*, 2009. URL <http://CRAN.R-project.org/package=dprep>. R package version 2.1.
- R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Proceeding FODO '93 Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, pages 69–84. Springer Verlag, 1993.
- J. Arroyo. Forecasting candlesticks time series with locally weighted learning methods. In H. Locarek-Junge and C. Weihs, editors, *Classification as a Tool for Research*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 603–611. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-10745-0.
- G. S. Atsalakis and K. P. Valavanis. Surveying stock market forecasting techniques - part ii: Soft computing methods. *Expert Systems with Applications*, 36(3, Part 2):5932 – 5941, 2009. ISSN 0957-4174.
- N. Baba and H. Suto. Utilization of artificial neural networks and the td-learning method for constructing intelligent decision support systems. *European Journal of Operational Research*, 122(2):501–508, April 2000.
- R. Barbosa and O. Belo. Autonomous forex trading agents. In P. Perner, editor, *Advances in Data Mining. Medical Applications, E-Commerce, Marketing, and Theoretical Aspects*, volume 5077 of *Lecture Notes in Computer Science*, pages 389–403. Springer Berlin / Heidelberg, 2008.
- G. Box, G. M. Jenkins, and G. Reinsel. *Time Series Analysis: Forecasting & Control* (3rd Edition). Feb. 1994.
- P. Carl and B. G. Peterson. *blotter: Tools for transaction-oriented trading systems development.*, 2011. URL <http://R-Forge.R-project.org/projects/blotter/>. R package version 0.8/r566.

- P. Carl, D. Eddelbuettel, B. G. Peterson, J. A. Ryan, and J. Ulrich. *quantstrat: Quantitative Strategy Model Framework*, 2011. URL <http://R-Forge.R-project.org/projects/blotter/>. R package version 0.4.0/r594.
- M. Casdagli. Chaos and Deterministic versus Stochastic Non-Linear Modelling. *Journal of the Royal Statistical Society. Series B (Methodological)*, 54(2):303–328, 1992.
- F. K.-P. Chan, A. W. chee Fu, and C. Yu. Haar wavelets for efficient similarity search of time-series: With and without time warping. *IEEE Transactions on Knowledge and Data Engineering*, 15:686–705, 2003. ISSN 1041-4347.
- K. P. Chan and W. C. Fu. Efficient Time Series Matching by Wavelets. *Data Engineering, International Conference on*, 0:126+, 1999. ISSN 1063-6382.
- P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-470-7.
- J. G. Cleary and L. E. Trigg. K*: An instance-based learner using an entropic distance measure. In *In Proceedings of the 12th International Conference on Machine Learning*, pages 108–114. Morgan Kaufmann, 1995.
- G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2:303–314, 1989. ISSN 0932-4194.
- M. R. El Shazly and H. E. El Shazly. Forecasting currency prices using a genetically evolved neural network architecture. *International Review of Financial Analysis*, 8(1):67–82, 1999.
- A. W.-c. Fu, P. M.-s. Chan, Y.-L. Cheung, and Y. S. Moon. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. *The VLDB Journal*, 9:154–173, July 2000. ISSN 1066-8888.
- W. Huang, K. K. Lai, and Y. Nakamori. Forecasting foreign exchange rates with artificial neural networks: A review. *International Journal of Information Technology & Decision Making*, 3, 2004.
- W. Huang, Y. Nakamori, and S.-Y. Wang. Forecasting stock market movement direction with support vector machine. *Comput. Oper. Res.*, 32:2513–2522, October 2005. ISSN 0305-0548.
- E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping to massive datasets. pages 1–11. Springer, 1999.
- V. Kodogiannis and A. Lolis. Forecasting Financial Time Series using Neural Network and Fuzzy System-based Techniques. *Neural Computing & Applications*, 11(2):90–102, Oct. 2002.

- J. W. Lee, J. Park, O. Jangmin, J. Lee, and E. Hong. A multiagent approach to Q-Learning for daily stock trading. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 37(6):864–877, 2007.
- M. T. Leung, A.-S. Chen, and H. Daouk. Forecasting exchange rates using general regression neural networks. *Computers & Operations Research*, 27(11-12):1093 – 1110, 2000. ISSN 0305-0548.
- W. Liu and L. Shao. Research of sax in distance measuring for financial time series data. In *Proceedings of the 2009 First IEEE International Conference on Information Science and Engineering, ICISE '09*, pages 935–937, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3887-7.
- M. Maggini, C. L. Giles, and B. Horne. Financial time series forecasting using k-nearest neighbors classification. In *NONLINEAR FINANCIAL FORECASTING*, pages 169–181, 1997.
- L. Martinez, D. da Hora, J. de M. Palotti, W. Meira, and G. Pappa. From an artificial neural network to a stock market day-trading system: A case study on the bm-fbovespa. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 2006 –2013, 2009. doi: 10.1109/IJCNN.2009.5179050.
- I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. Yale: Rapid prototyping for complex data mining tasks. In L. Ungar, M. Craven, D. Gunopulos, and T. Eliassi-Rad, editors, *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940, New York, NY, USA, August 2006. ACM. ISBN 1-59593-339-5.
- S. Mitra and A. Mitra. Modeling exchange rates using wavelet decomposed genetic neural networks. *Statistical Methodology*, 3(2):103 – 124, 2006. ISSN 1572-3127.
- Y. nei Law and C. Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. In *In PKDD*, pages 108–120. Springer, 2005.
- Nguyen and D. T. Anh. Combining SAX and Piecewise Linear Approximation to Improve Similarity Search on Financial Time Series. In *Information Technology Convergence, 2007. ISITC 2007. International Symposium on*, pages 58–62, 2007.
- R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL <http://www.R-project.org>. ISBN 3-900051-07-0.
- V. M. Rivas, J. J. Merelo, P. A. Castillo, M. G. Arenas, and J. G. Castellano. Evolving rbf neural networks for time-series forecasting with evrbf. *Inf. Sci. Inf. Comput. Sci.*, 165:207–220, October 2004. ISSN 0020-0255.
- F. F. Rodríguez, S. S. Rivero, and J. A. Félix. Exchange-rate forecasts with simultaneous nearest-neighbour methods: evidence from the EMS. *International Journal of Forecasting*, 15(4):383–392, 1999.

- J. A. Ryan. *quantmod: Quantitative Financial Modelling Framework*, 2011. URL <http://R-Forge.R-project.org/projects/quantmod/>. R package version 0.3-16/r561.
- E. W. Saad, D. V. Wunsch, and D. C. Li. Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE Transactions on Neural Networks*, 9(6):1456–1470, 1998.
- F. E. H. Tay and L. Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309 – 317, 2001. ISSN 0305-0483.
- J. Ulrich. *TTR: Technical Trading Rules*, 2011. URL <http://R-Forge.R-project.org/projects/ttr/>. R package version 0.20-3/r107.
- K. Yang and C. Shahabi. An efficient k nearest neighbor search for multivariate time series. *Information and Computation*, 205(1):65 – 98, 2007. ISSN 0890-5401. Special Issue: TIME 2005.
- J. Yao and C. L. Tan. A case study on using neural networks to perform technical forecasting of forex. *Neurocomputing*, 34(1-4):79 – 98, 2000. ISSN 0925-2312.
- J. Yao, C. L. Tan, and H. Lee Poh. Neural networks for technical analysis: A study on klcj, 1999.
- C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish. Indexing the Distance: An Efficient Method to KNN Processing. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 421–430, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1558608044.
- L. YU, S. WANG, W. HUANG, and K. K. LAI. Are foreign exchange rates predictable? a survey from artificial neural networks perspective. *Scientific Inquiry*, 8, 2007.
- L. Yu, K. K. Lai, and S. Wang. Multistage rbf neural network ensemble learning for exchange rates forecasting. *Neurocomputing*, 71(16-18):3295 – 3302, 2008. ISSN 0925-2312. Advances in Neural Information Processing (ICONIP 2006) / Brazilian Symposium on Neural Networks (SBRN 2006).
- M. S. Yümlü, F. S. Gürgen, and N. Okay. Financial time series prediction using mixture of experts. In *ISCIS*, pages 553–560, 2003.
- S. Zemke. Data mining for prediction. financial series case, doctoral thesis, the royal. 2003.
- G. Zhang, B. E. Patuwo, and M. Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, 14(1):35 – 62, 1998. ISSN 0169-2070.

- P. Zufiria and P. Campoy. Estimating the embedding dimension distribution of time series with somos. In J. Cabestany, F. Sandoval, A. Prieto, and J. Corchado, editors, *Bio-Inspired Systems: Computational and Ambient Intelligence*, volume 5517 of *Lecture Notes in Computer Science*, pages 1168–1175. Springer Berlin / Heidelberg, 2009.

Appendix A

Notable trading strategies

This appendix contains trading visualisations of best configurations for k-Nearest Neighbor, our Cluster+ANN method and standalone ANN that were found during the testing described in chapter 5. For illustration, we have also included one of the worst performing strategies we have found during the testing.



Figure A.1: This is visualisation of the most successful Clustering+ANN group configuration. This strategy is able to spot smaller trading opportunities and tends skip the large ones. This is illustrated by performance in the first year of trading, which is notably worse than e.g. kNN (next page). However, it was able to operate much better in the more erratic movements of the second year.



Figure A.2: This is visualisation of the most successful k-Nearest Neighbors configuration. Although several good trading decisions can be seen in the major rises, the strategy wasn't able to avoid losing in some of the subsequent declines (e.g. in 08/2010) and did not prosper in frequently changing market (end of the trading period)



Figure A.3: This is visualisation of the most successful standalone Artificial Neural Network configuration. It generates only the first promising opportunity, but then never creates an exit signal. Therefore, it can be used as an equivalent of a Buy-and-Hold strategy - a baseline comparison.



Figure A.4: This prediction process was based on SVM with polynomial kernel and as can be seen, the strategy behaves almost inversely to what is desirable.

Appendix B

CD directory structure

The attached CD has following directory structure:

```
\PredEnv - Directory containing the whole project with its
           required directory structure.
\PredEnv\repository - Directory with RapidMiner's
                    repository, which contains the main process(file Main.
                    rmp), the optimization process (file Optimization
                    Environment.rmp) and the offline time series datasets.
\Thesis.pdf - Text of this work.
```