Decision Tree Classification and Forecasting of Pricing Time Series Data

EMIL LUNDKVIST

KTH Electrical Engineering

Master's Degree Project
Stockholm, Sweden July 2014

TRITA-XR-EE-RT 2014:017

Abstract

Many companies today, in different fields of operations and sizes, have access to a vast amount of data which was not available only a couple of years ago. This situation gives rise to questions regarding how to organize and use the data in the best way possible.

In this thesis a large database of pricing data for products within various market segments is analysed. The pricing data is from both external and internal sources and is therefore confidential. Because of the confidentiality, the labels from the database are in this thesis substituted with generic ones and the company is not referred to by name, but the analysis is carried out on the real data set. The data is from the beginning unstructured and difficult to overlook. Therefore, it is first classified. This is performed by feeding some manual training data into an algorithm which builds a decision tree. The decision tree is used to divide the rest of the products in the database into classes. Then, for each class, a multivariate time series model is built and each product's future price within the class can be predicted. In order to interact with the classification and price prediction, a front end is also developed.

The results show that the classification algorithm both is fast enough to operate in real time and performs well. The time series analysis shows that it is possible to use the information within each class to do predictions, and a simple vector autoregressive model used to perform it shows good predictive results.

Acknowledgements

I would like to express my very great appreciation to Daniel Antonsson for giving the opportunity to work with his group and the valuable information he has shared.

I would also like to extend my thanks to Patricio Valenzuela who has acted as my supervisor at KTH and provided important knowledge and advice. My grateful thanks are also extended to Professor Cristian Rojas who is my examiner at KTH and has given his time to help me.

Finally I would like to thank my family for their support during the thesis.

Contents

List of Figures

List of Tables

Glossary

$G(X)$ The gain criterion of a test X. 14

$I(D)$ The entropy of the training data D. 15

$\gamma_X(r, s)$ The covariance function. 19

$\mu_X(t)$ The mean function. 19

θ The number of model parameters. 23

$\{X_t\}$ A time series. $t = 0, \ldots, n$. 19

$\{Z_t\}$ A time series, usually representing white noise. $t = 0, \ldots, n$. 20

p_v The p-value. 44

ACF, $\rho_X(h)$ The autocorrelation function. 21

ACVF, $\gamma_X(h)$ The autocovariance function. 20

AR(p) Autoregressive model of order p. 22

ARMA(p,q) Autoregressive moving average model of order $p + q$. 22

MA(q) Moving average model of order q. 22

PACF, $\alpha_X(h)$ The partial autocorrelation function. 21

VAR(p) Vector autoregressive model of order p. 22

WN$(0, \sigma^2)$ White noise with mean 0 and variance σ^2. 20

Chapter 1

Introduction

1.1 Background

Humans have always been and always will be curious about the future. We have for instance tried to predict small things such as the outcome of tomorrow's big soccer game or the upcoming election, and more complex scenarios like what the fate of our planet and the universe will be in the next couple of billion years.

A famous example of a successful prediction is by the German astronomer Johann Gottfried Galle (Figure 1.1), who predicted the existence of the previously unknown planet Neptune by calculations based on Sir Isaac Newton's law of gravity. A more recent example of the same kind is the prediction of the Higgs boson by François Englert and Peter Higgs, which only recently was measured to be correct [29].

Other predictions have become famous because of their extreme lack of accuracy. One example is when Thomas Watson, chairman of IBM, in 1943 (supposedly) made the following prediction: "I think there is a world market for maybe five computers." Another example is the reason why Decca Records Ltd. (Figure 1.2) rejected the Beatles in 1962: "We don't like their sound, and guitar music is on the way out."

The last example clearly highlights the value of predictions in economic situations. If a company always knew everything about the coming market trends and the competitors products, it would of course be easier to adapt to the market and optimize the company's strategy. Even if only partial information were available, it would be of high value.

Many companies sell products in a highly competitive market. Therefore, knowledge about the current pricing and predictions of future market prices in different segments are very important. With this information the companies can set the price of their own products in a manner that they sell at a price that is low enough to

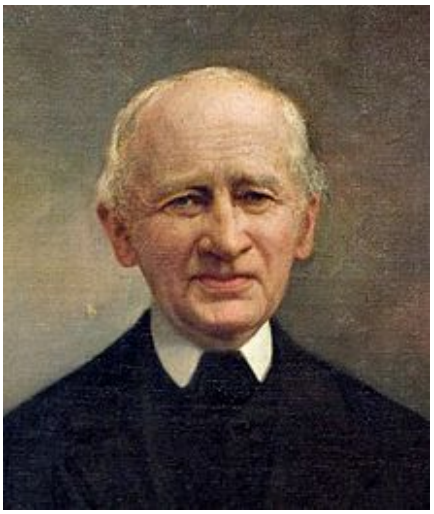

Figure 1.1: Johann Gottfried Galle - A great predictor.

©This logo is the property of Universal Music Group

Figure 1.2: Decca Records - A not so great predictor.

get a high sell-out, but at the same time is high enough to get a sufficient margin. This balancing act is made significantly easier if:

1. The current pricing information for the whole market is easy to retrieve and display.
2. Predictions about future prices in the market are available.

Previous projects at the company in this area have focused on building a back-end for collection, storage and distribution of current product pricing information. These projects have so far collected more than 2 million pricing data points that can be retrieved and displayed at request. This thesis is dedicated to develop a solution for the previously mentioned points by classifying the available pricing information and applying time series analysis to do predictions. The problem definition is further described in the next section.

1.2 Problem Definition and Requirements

As introduced in Section 1.1 it is of high importance for a company to be able to both get an understanding of current and future prices to position its products competitive with respect to similar products in the market. The current situation at the company is that the pricing information is available as a large database, which is difficult to analyse and visualise. This pricing information consists of a list of products, each with properties corresponding to the product, and also pricing

information for the product over time (the structure of the data is further explained in Section 2.1).

The main goal of this thesis is to make the available information more valuable to the employees that use it for pricing decisions. In order to do this, two main steps have been identified:

1. Classification of the products into logical classes.

2. Prediction of the price evolution of each product, in relation to its classes.

The need for classification of the products into classes comes from the fact that the employees analyse different market segments separately. All products in one group will have some properties that are common within the group. These groups could be manually assigned by the employees, but this would be very time consuming. Therefore, a classification algorithm that takes a few training examples and then sorts the rest of the products into suitable classes would be preferable. When the products are classified it is a simple task to display their pricing relative to the other products in the same class.

The classification is also important for the prediction of each product's future price. Within each class it is likely that the prices vary together. For instance, if one product in a class drops its price well below the competition, the probability that the other products in the same class adjust their prices accordingly is high, preventing loss of sell out and market share in the segment. If one would consider the whole market when predicting a single product's future price, the prediction would likely not be as accurate. For example, if one day the price of a specific toaster would drop, it is not very likely that the price of cameras will change accordingly. Even if on one occasion, a change in one toaster's price would affect a specific camera's price, it would most certainly be a coincidence and this information would not be valuable for future prediction of the camera's price.

The employees also need a way of interacting with the classification and forecasting systems by entering information and validating the results. This gives rise to the need of a front end application that can handle the tasks of entering and displaying data and analysis.

Figure 1.3 shows an overview of the whole process. The available data is unstructured from the start. With input from the employees and a classification algorithm the data is classified. Using information from each class and applying time series analysis, forecasts for each product in the class can be obtained. In the three following sections the requirements on the classification algorithm, the time series analysis and the front end application respectively are further described.

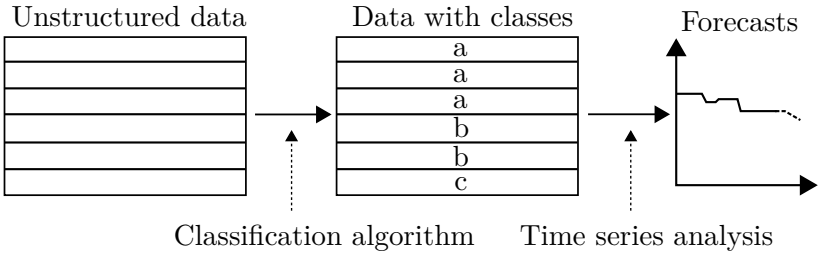

Figure 1.3: Overview of the process.

1.2.1 Requirements on the Classification Algorithm

As explained above, the most important advantage of using a classification algorithm is that it helps the employees classify the data faster and better than they would be able to do manually. The most important requirements of the algorithm are:

- It should be able to make use of training samples that employees enter to the algorithm.
- It should almost correctly classify all the products in the database, according to the employees that use the classification.
- It should be fast enough to work in real time so that the employees can update the classes and make new analysis when working with the program.
- It is preferable if the algorithm is easy for the employees to interpret, so the employees know why a specific product is classified in a certain class.

1.2.2 Requirements on the Time Series Analysis

The time series analysis aims at making predictions within each specified class from the classification to create a better insight of how the pricing will change in the market for the employees. The most important requirements of the analysis are:

- It should be able to make predictions of each product within a class, based on the price evolution of all the products in the same class.
- The prediction should contribute to a better understanding of the future price evolution of the products, than what is possible to achieve for an employee that simply looks at the data.
- It should be fast enough to work in real time so that the employees can analyse products in different classes when working with the program.

1.2.3 Requirements on the Front End Application

The front end enables the employees to interact with the classification algorithm and the time series analysis. The most important requirements of the front end application are:

- The employees should be able to enter classification training data.
- The employees should be able to view the outcome of the classification algorithm.
- The employees should be able to view predictions for a given class.
- The application should operate fast enough for the employees to be able to work in real time.

The last requirement corresponds to that no operations should take longer than 10 seconds, since that approximately is the time for keeping a users attention before being distracted by other tasks [28].

1.3 Previous Work

The previous work at the company in this area has mainly consisted of collecting the necessary data for further analysis. This has resulted in a database with more than 2 million unclassified pricing data points and the number grows for each day. Manual classification and analysis have then been carried out on a subset of the data.

Both in the field of classification and time series analysis there are a large number of results. Common methods in machine learning are decision tree learning, neural networks, support vector machines and Bayesian networks [4]. The difficulty of this project is to find and understand what information is relevant to solve the problem defined in Section 1.2, and then implement and evaluate it.

There are many software solutions that at different levels help users with both classification and time series analysis separately. Some examples for time series analysis are Zaitun Time Series [41], Strategico [40] and Ipredict [15]. Examples for classification include Salford Systems Cart [36] and the Attivo Classification Engine [5]. However, all of the found solutions handle very general data sets. This thesis aims at building a solution that is tightly integrated with the existing product information from the database and the work flow of the company's employees. It also tries to evaluate if this kind of analysis has value for the employees or if their ordinary work flow is better suited for the task.

1.4 Outline of the Report

In this first chapter an introduction to the thesis has been given. In Chapter 2, the theory needed for understanding the problem is presented. Then, the analysis and implementation are described in Chapter 3. The results for the classification, the time series analysis and the front end are shown in Chapter 4. Lastly, the conclusions and future work are presented in Chapter 5.

Chapter 2

Theory

In this chapter the theory needed for understanding the analysis, implementation and results is introduced. Only a brief overview of the used concepts are given. For a more detailed theory, please refer to the references given in the sections.

First, the data structure of the database is explained in Section 2.1. Then, in Section 2.2, the choice of classification method is motivated, and a theoretical foundation is presented. Section 2.3 begins by introducing time series analysis notations. Then, time series theory is presented and a short example of a time series is explained as an introduction to the field.

2.1 Data Structure

As mentioned in the introduction, a very large database with product and pricing information is available for this project. The structure of the data in this database is shown in Figure 2.1. As it can be seen in the figure, the database consists of two tables called *Product* and *PricePoint*.

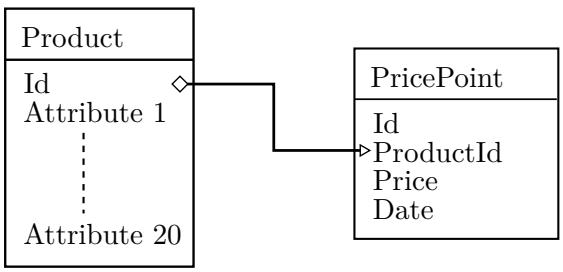

Figure 2.1: The structure of the data in the database.

A *table* in a database is a method to define the structure of the data stored in it. All the products stored in the database have the format of the Product table and all the price points have the format of the PricePoint table. The *Id* field, which both tables have, is a unique identifier for an entry in the database. This means that each time that a new product or price point is submitted to the database, it gets an Id that no other product has or will have, and keeps this Id as long as it is in the database. This particular database is of a type called a *relational database* [11]. A relational database enables relations between the tables. In Figure 2.1, one such relation is defined by the arrow between the Product's Id (*PK* - private key) and the PricePoint's ProductId (*FK* - foreign key). The relation is of the type *one to many*, which means that one Product can have a relation with many PricePoints. The foreign key field in the PricePoint table connects it to a specific product in such a way that the foreign key is the same as the private key of one product entry. In this way one product can have many price points and each price point is connected only to one product.

An example of a couple of entries in the database is shown in Table 2.1. In the example, there are two product entries. Each entry has a unique Id and some attributes which can be unique, but most often are not. There are four price point entries, each with a unique Id. However, all price points from Id 1 to 3 have the same ProductId. This is the relation that tells that all these price points belong to the same product, namely the one with Id 1. The reason for this structure is that each product can have many price points and each price point corresponds to the price at a specific time. In this example the product with Id 1 has three price points for three different dates, while the product with Id 2 only has one price point corresponding to one date.

Product		
Id	**Attribute 1**	**Attribute 2**
1	A	15
2	B	15

PricePoint			
Id	**ProductId**	**Price**	**Date**
1	1	2000	2014-06-24
2	1	2000	2014-06-25
3	1	1900	2014-06-26
4	2	3000	2014-06-26

Table 2.1: Example of a couple of entries in the database.

The fields of the tables Product and PricePoint in the database have a number of different data types. Table 2.2 lists these datatypes and gives a short comment on each one. The different kinds of data types set limitations on what sorts of classification algorithms can be used, which is discussed in Section 2.2.1.

Data Type	Comments
int	Integer value, whole numbers.
float	Floating-point value, decimal numbers.
bit	Boolean value, 0 or 1, `true` or `false`.
varchar	Variable character field, strings of text.
date	A date with year, month, day, hour and so on.

Table 2.2: Data types in the database.

2.2 Classification

The problem of classification is that of assigning classes to objects of unknown class. The set of possible classes is discrete in contrast to the case in parameter estimation. In the field of machine learning the problem can be further divided into *unsupervised* and *supervised* learning [4].

In *unsupervised learning* there is no prior information about which object should belong to which class. Sometimes not even the set or number of possible classes is known. It is entirely up to the classification algorithm to group similar objects together. Figure 2.2 shows an example of unsupervised learning, where a couple of objects are classified according to their geometrical similarity.

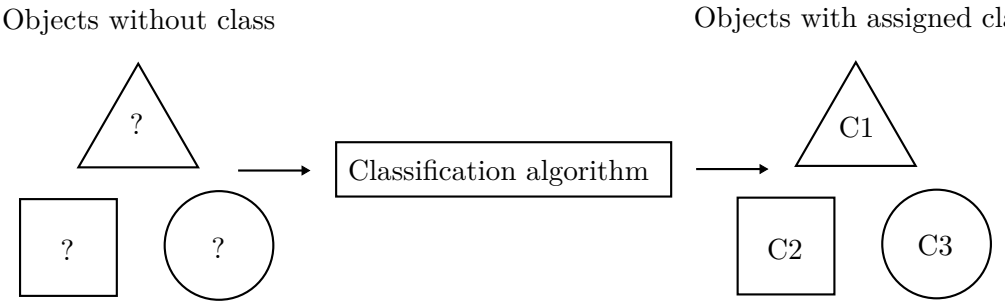

Figure 2.2: Unsupervised learning.

In *supervised learning* there exists a training set of objects, where the class is already known. This set of objects is called the *training data*. In this case the aim is to classify the rest of the given objects by looking at their attributes and comparing to the objects with an already known class. An example of supervised learning is shown in Figure 2.3, where the classification algorithm uses the training data to classify the geometrical shapes in the predefined classes.

There are many different classification algorithms and, as explained in Section 2.2.1, the *Decision Tree Classification* approach was chosen for this project. Section 2.2.2 provides the theory on how to do classification when one already has a tree and Section 2.2.3 explains how to build a decision tree from training data.

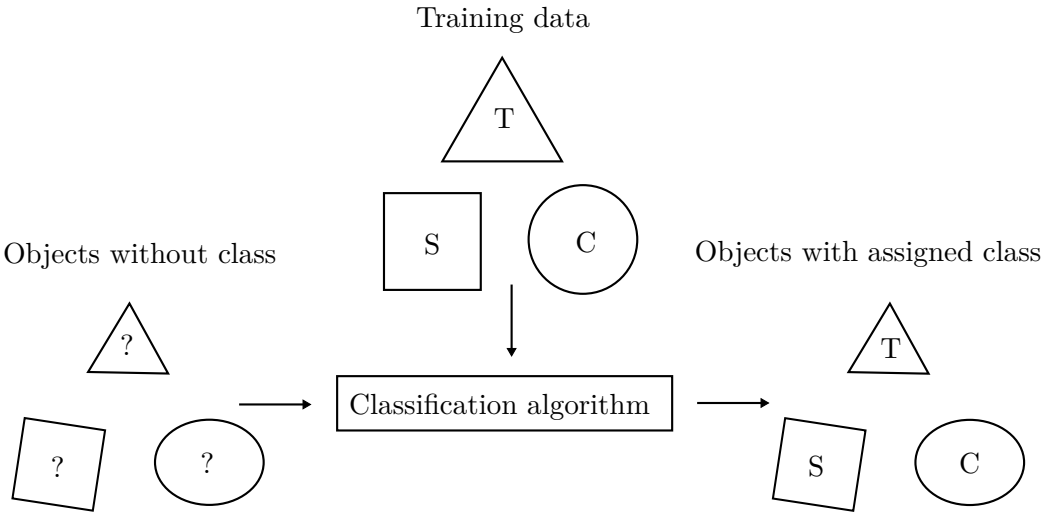

Figure 2.3: Supervised learning.

2.2.1 Choice of Method

There is a wide range of classification algorithms suitable for different kinds of classification problems. Table 2.3 holds an overview of some common classification algorithms and their characteristics. As discussed in Section 1.2.1, there are several requirements on the classification algorithm. The most important requirements are that it should be fast enough to work in real time and also that the employees should find the result satisfactory.

A technical constraint stems from the fact that the available data has a predefined set of data types (as explained in Section 2.1). These data types are both of continuous and discrete nature, and some are also categorical. A *categorical* variable has the property that it can only adopt a fixed set of values. These values may not have a measure of length, meaning that for instance size comparison between two variables is not possible. As an example of this, consider the integer numbers and compare with types of weather. We all know that 5 is larger than 3 and smaller than 6 and we can also define a distance between the two integers i_1 and i_2 as $d = |i_1 - i_2|$. On the other hand, we cannot say that rainy weather is larger than sunny weather or define a distance between the two (of course we can, but it would not make much sense). Table 2.4 holds a description of the different data type properties.

The existing data types in the database set the requirement of our classification method that it has to be able to handle both categorical and non-categorical predictors. This rules out the classification methods “Support Vector Machines” and “Discriminant Analysis” as suitable algorithms. “Naive Bayes” is not suitable either, because of the fact that it is only fast for small datasets and the dataset from the database is very large. That leaves “Decision Trees” and “Nearest Neighbour”

Algorithm	**Fitting Speed**	**Prediction Speed**	**Memory Usage**	**Easy to Interpret**	**Categorical Predictors**	**Predictive Accuracy**
Decision Trees	Fast	Fast	Low	Yes	Yes	Low
Support Vector Machines	Medium	Medium	Low for few support vectors	Easy for few support vectors	No	High
Naive Bayes	Fast for small datasets	Fast for small datasets	Low for small datasets	Yes	Yes	Low
Nearest Neighbour	Fast	Fast	High	No	Yes	Good in low dimensions
Discriminant Analysis	Fast	Fast	Low	Yes	No	Good when modelling assumptions are satisfied

Table 2.3: Overview of classification algorithms [14, 20].

Type	**Description**	**Example**
Continuous	A continuous variable (defined distance)	1.0, 1.2, 1.5
Discrete	A discrete variable (defined distance)	1, 2, 3
Categorical	Distinct and separate values (no defined distance)	Sunny, rainy and cloudy weather

Table 2.4: Description of data type properties.

as good alternatives as they are both fast and handle categorical predictors. There are two main differences between them which are that "Nearest Neighbour" has a high memory usage and is not easy to interpret while "Decision Trees" has a low memory usage and is easy to interpret. One of the requirements of the algorithm (from Section 1.2.1) is that it should give results that are easy to interpret, which makes the "Decision Tree" algorithm the best one for the given task. It is fast for fitting and prediction, handles categorical predictors, has a low memory usage and is easy to interpret. The only drawback is that it has relatively low predictive accuracy compared to the other algorithms. However, the requirement is that it should classify the data in a satisfactory way according to the employees. In the results in Section 4.1.4 it is shown that this actually is fulfilled.

With the motivation above, decision trees are chosen for the classification of the data

in this project. Some decision tree algorithms are presented in Table 2.5. Because of the large amount of information and intuitive design of the ID3 algorithm, this was the first algorithm considered. One very important limitation of the ID3 algorithm is that it does not handle discrete attributes [33], which is needed for the classification in this project. However, the C4.5 algorithm improves on this point [32] and it is also able to handle missing attribute values. Therefore, the C4.5 algorithm was chosen for the project.

Algorithm	**Comment**
ID3	Iterative Dichotomiser 3
C4.5	Successor of ID3
CART	Classification And Regression Tree
CHAID	CHi-squared Automatic Interaction Detector

Table 2.5: Decision tree algorithms.

2.2.2 Classifying With an Existing Decision Tree

A decision tree can be seen as a graph representation of a decision algorithm where each node holds a statement that is either `true` or `false`. From each node there are two directed edges where one correspond to `true`, and one to `false`. Depending on the value of the statement, the decision algorithm continues along the corresponding edge. The first node, where the decision algorithm starts, is called the *root* node and the last nodes, where no further decision can be made, are called *leaf* nodes.

A decision tree is used when classifying objects of unknown class. The decision algorithm starts at the root node and checks if this first statement is `true` or `false` for the object by checking its properties. Then, it continues until it reaches a leaf node. All leaf nodes have classes assigned, and this is the class that the algorithm assigns to the object.

An example of this process is shown in Figure 2.4. Consider you have an unknown geometrical shape and you want to determine if it is a red equilateral triangle, an equilateral triangle, a hexagon or something else. Then you can start at the root node checking if the statement "Does it have three sides of equal length?" is `true`. Continuing down in the tree you will reach one of the leaf nodes and be able to decide which class it belongs to. This procedure have to be repeated for each object that should be classified. However, this classification method requires an existing decision tree well built for the problem. The next section explains how to build such decision trees.

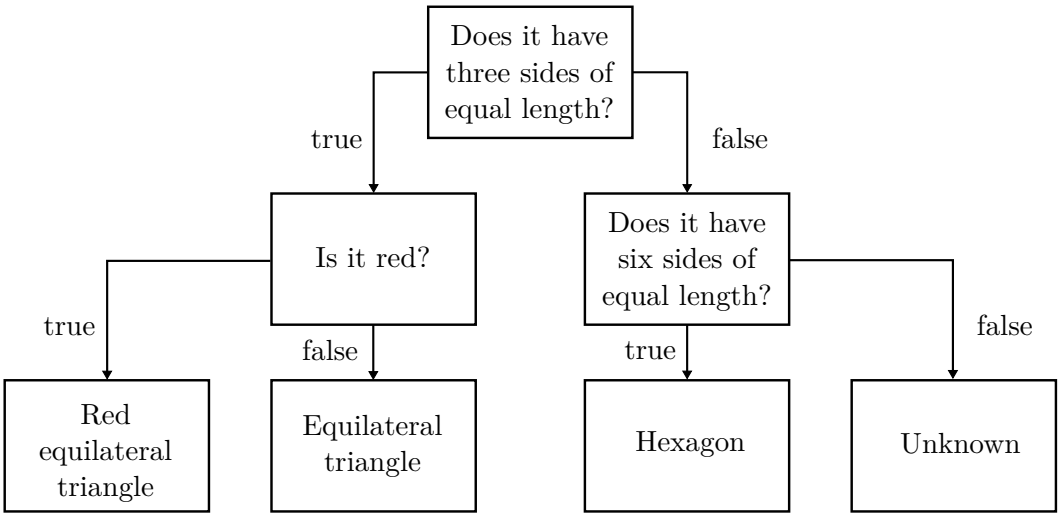

Figure 2.4: Decision tree example.

2.2.3 Building a Decision Tree

As explained in Section 2.2.1, the algorithm called *C4.5* [32] was chosen for this project. C4.5 uses a simple and elegant method with the concept of "divide and conquer" for building decision trees. The general idea of the algorithm is presented in Algorithm 1.

Algorithm 1: General algorithm for building decision trees [16].

Data: training data
Result: a decision tree
check for base cases;
let a' be the attribute to split on;
forall the *attributes* a **do**
    calculate measure $g(a)$ for how good a splits the training data;
    if $g(a)$ *better than* $g(a')$ **then**
        set a as new a';
    end
end
create a decision node that splits on a';
add the resulting data to the children of the node;
recursively enter the children to the algorithm;

Let the training data set for the algorithm be denoted by D and the classes by $\{C_1, C_2, \cdots, C_n\}$. In the example in Figure 2.3 the training data D would be the set of the triangle with class T, the square with class S and the circle with class C and the corresponding classes would be $\{T, S, C\}$. There are three base cases to

consider [32]:

1. **The training data in D belong to the same class C_i.** Then the decision tree for D simply is a single leaf with this class. For example: If there are only triangles labelled T in the training data D, there is no reason to build a decision tree, all data can be considered to belong to class T.

2. **There is no training data in D.** Then the resulting tree is a single leaf with the most frequent class in all the training data at the levels above in the decision tree.

3. **The training data in D belong to different classes.** This is the most interesting base case where a decision tree has to be built to be able to divide the tree for useful classification. In this base case a test is created (explained later), which divides the training data into two new sets depending on the outcome of the test. Each of these training data sets are then recursively entered into the algorithm again and further divided until they reach one of the above base cases (refer to Algorithm 1). For example: If there is one triangle labelled T and one square labelled S in the test data D, a test is needed to tell the two apart. A good test in this case would be to consider the number of edges of each object. The test "Does the object have more than 3 edges?" would divide the training data into two more test data sets which both would reach a case in the next recursion of the algorithm.

This recursive algorithm will first consider the whole training data set D and split it until each subset of it reaches one of the first two cases. One important part to understand is how the test in base case number 3 is created. In the older algorithm ID3 [33], this is performed with a statistic called the *gain criterion* $G(X)$ [32]. Let $f(C_i, D)$ be the number of cases in the training data D that belong to the class C_i, $|D|$ be the number of training cases in D, and X be a test under consideration. Then G is defined as

$$G(X) := I(D) - I_X(D), \tag{2.1}$$

where

$$I(D) := -\sum_{i=1}^{k} \frac{f(C_i, D)}{|D|} \log_2 \frac{f(C_i, D)}{|D|}, \tag{2.2}$$

and

$$I_X(D) := \sum_{i=1}^{n} \frac{|D_i|}{|D|} I(D_i). \tag{2.3}$$

$\frac{f(C_i,D)}{|D|}$ can be seen as the probability of randomly picking the class C_i from the training data D. Then, by using Shannon's general formula for uncertainty [37] we obtain the *entropy* $I(D)$ in equation (2.2). $I_X(D)$ is another entropy measure *after* the training data D has been tested with the test X and got n outcomes of new training data D_i. ID3 then selects the test that maximizes the gain criterion G in equation (2.1).

However, this approach favours tests that split the training data into many outcomes which can give an overfitted model [32]. In C4.5, this problem is overcome by instead maximizing the *gain ratio criterion*:

$$G_r(X) := \frac{G(X)}{A(X)}, \tag{2.4}$$

where

$$A(X) := -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}. \tag{2.5}$$

$A(X)$ acts as a normalizer that penalizes tests with many outcomes to balance the gain criterion. The next examples give an intuitive explanation of the measures (2.1) and (2.4).

Example 2.2.1 (Entropy)**.** Entropy can intuitively be seen as the amount of information in a distribution. A more general way of writing equation (2.2) is

$$I(D) = -\sum_{j=1}^{k} p_j \log_2 p_j \tag{2.6}$$

where p_j is the probability for the outcome j of the random variable D. Consider the following three tosses of a coin, with different probabilities for heads and tails coming up:

1. **A fair coin.** For a fair coin the probability of getting heads up and getting tails up is equal ($p(\text{heads}) = p(\text{tails}) = 0.5$). In this case the entropy is $I(D) = 1$ according to Equation (2.6). To simplify the notation we write $I(0.5, 0.5) = 1$.
2. **A non-fair coin.** For a coin that is not fair, the probability of getting heads up and getting tails up is not equal. If for example the probability for heads up is $p(\text{heads}) = 0.8$, the entropy is $I(0.8, 0.2) = 0.72$.
3. **A coin with two equal sides.** If the coin has heads on both sides, the probability of heads is $p(\text{heads}) = 1$ and the entropy is $I(1, 0) = 0$.

In the first case, where the coin is fair, we know nothing about if the outcome will be heads or tails up from a coin flip before the result is observed. After the coin is tossed, we gain a lot of information of the outcome (from having no idea of the outcome to know exactly) and therefore the information contained in this distribution is high. In contrast, in the third case where the coin has two equal sides, there is no information in the distribution at all. This is because we already know the outcome of the coin flip before it is carried out. After the coin is tossed and we observe the result, the knowledge of the outcome has not increased. In the case of a non-fair coin, we know something about the outcome and the entropy is therefore somewhere between the entropy of the fair coin and the coin with two equal sides. Figure 2.5 shows the entropy as a function of the probability p(heads) to get heads up for a coin. □

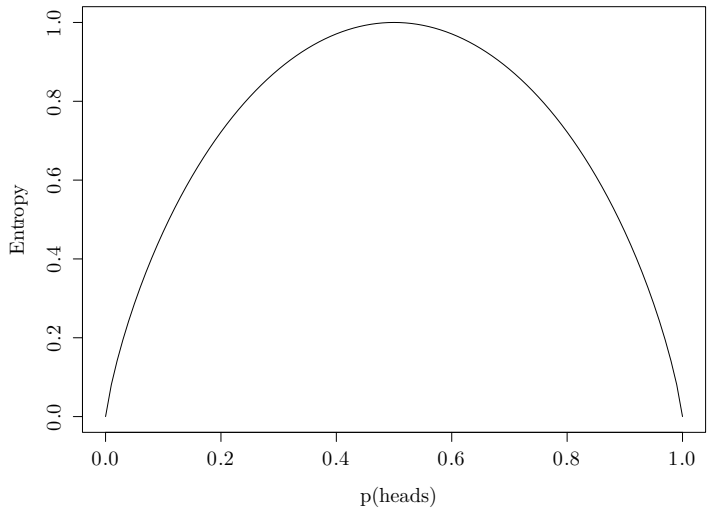

Figure 2.5: Entropy as a function of the probability of heads for a coin toss.

$I_X(D)$ and $A(X)$ are similar entropy measures as the entropy $I(D)$ in the previous example. $I_X(D)$ sums the weighted information contained in all the new partitions D_i, $i = 1, \ldots, n$, resulting from a specific split X from the training cases D. The information gain $G(X)$ is then the difference of the current information of the distribution and the information after the split. We want the information remaining in the distribution as low as possible after the split, because it means that we have more knowledge of the outcome (compare to the coin flip where the coin has two equal sides). Therefore, the information gain $G(X)$ should be maximized for a good split.

To get the gain ratio criterion $G_r(X)$ for a split, the normalizer $A(X)$ is used to normalize the gain criterion. $A(X)$ is larger for splits with many partitions D_i than

for few partitions. To see this, consider the extreme case for a split X_0 which does not split D. Then $\frac{|D_i|}{|D|} = 1$ and $A(X_0) = 0$. This makes $G_r(X_0)$ large. The other extreme is X_∞ which results in an infinite number of splits. Then $A(X_\infty)$ is large and $G_r(X_\infty)$ is small. In reality the splits are somewhere in between and $A(X)$ acts as a normalizer that penalizes many splits D_i, as previously mentioned. Example 2.2.2 illustrates these concepts with some training data.

Example 2.2.2 (C4.5)**.** This example illustrates some concepts from the C4.5 algorithm by considering the training data in Table 2.6. In the table, the attributes "Weather", "Have ice cream", "Have money" and "Age" are shown together with the decision if an ice cream should be eaten or not.

Weather	**Have ice cream**	**Have money**	**Age**	**Eat ice cream?**
Hot	Yes	Yes	Old	Eat
Hot	Yes	Yes	Young	Eat
Hot	Yes	No	Old	Eat
Hot	Yes	No	Young	Eat
Hot	No	Yes	Old	Eat
Hot	No	Yes	Young	Eat
Hot	No	No	Old	Don't eat
Hot	No	No	Young	Don't eat
Cold	Yes	Yes	Old	Don't eat
Cold	Yes	Yes	Young	Don't eat
Cold	Yes	No	Old	Don't eat
Cold	Yes	No	Young	Don't eat
Cold	No	Yes	Old	Don't eat
Cold	No	Yes	Young	Don't eat
Cold	No	No	Old	Don't eat
Cold	No	No	Young	Don't eat

Table 2.6: C4.5 example training data.

From equation (2.2) we get

$$I(D) = I\left(\frac{6}{16}, \frac{10}{16}\right) = -\frac{6}{16}\log_2\frac{6}{16} - \frac{10}{16}\log_2\frac{10}{16} = 0.95, \tag{2.7}$$

which is the entropy of the distribution. If we consider a split on "Weather", we get

$$I_{\text{Weather}}(D) = \frac{8}{16}I\left(\frac{6}{8}, \frac{2}{8}\right) + \frac{8}{16}I\left(\frac{8}{8}, \frac{0}{8}\right) = \frac{8}{16}\cdot 0.81 + \frac{8}{16}\cdot 0 = 0.41 \tag{2.8}$$

from equation (2.3), and $G(\text{Weather}) = 0.54$ from equation (2.1). The results from the other attributes are shown in Table 2.7. In this table, one can see that the

information gain is largest for "Weather", equal for "Have ice cream" and "Have money", and 0 for "Age". Therefore, the split is done on the attribute "Weather" in this simple example.

Split Attribute	$\mathbf{I(D)}$	$\mathbf{I_X(D)}$	$\mathbf{G(X)}$
Weather	0.95	0.41	0.54
Have ice cream	0.95	0.91	0.04
Have money	0.95	0.91	0.04
Age	0.95	0.95	0

Table 2.7: C4.5 example measures.

The decision tree from which the training data was generated can be seen in Figure 2.6. In that tree, the attribute "Age" does not exist, i.e. the attribute does not affect the outcome if we should eat ice cream or not. This is reflected in Table 2.7 where "Age" has the information gain $G(\text{Age}) = 0$. One can also observe that the attribute "Weather" is at the root of the tree and that if it is "Cold" we should never eat ice cream. In this sense, it is a good split variable which is reflected in its relatively high information gain.

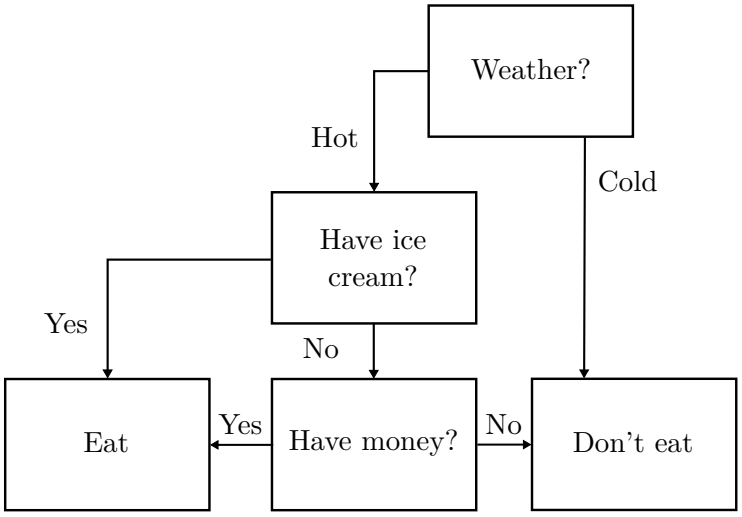

Figure 2.6: C4.5 example decision tree.

□

2.3 Time Series Analysis

There are many methods for predicting future values based on previous observations. Methods that have been considered for this thesis are artificial neural networks [42],

support vector machines [10], time series analysis [8, 18] and hidden Markov models [7]. Due to the limited amount of time for the thesis, not all methods could be tested on the data set. Time series analysis was chosen for further analysis because of the following reasons:

- Time series analysis is well known and used for many applications.
- There are many available tools for working with time series analysis.
- The available dataset is typical for time series analysis.

The fact that time series analysis is well known and used has led to large amounts of information available in the area. There are several papers and books written on the subject [18, 8] and software such as Matlab, R [34] and R.NET [35] have implementations that enable fast testing on the real data set. The other methods considered does not seem to have equally strong standardizations of the theoretical approach. Statements such as "While artificial neural networks provide a great deal of promise, they also embody much uncertainty" [42] and "To overcome the challenges in predicting time series with hidden Markov models some hybrid approaches have been applied" [7] are common when reading about the other methods. This gives the impression that they are not equally developed and time series analysis is a more robust choice in this sense.

2.3.1 Introduction

A *time series* $\{X_t\}$ is a collection of observations at different points in time, usually equally spaced. Time series analysis is the study of previously collected time series data with the hope that it could say something about the future outcome of the observed object.

In this section some important concepts from time series analysis needed for the understanding of the analysis, implementation and results, are introduced.

Stationarity

Stationarity is an important concept in time series analysis. First, the mean and covariance functions are introduced:

Definition 2.3.1 (Mean and Covariance)**.** For a time series $\{X_t\}$, the *mean function* $\mu_X(t)$ is defined as

$$\mu_X(t) = E[X_t], \tag{2.9}$$

and the *covariance function* $\gamma_X(r, s)$ as

$$\gamma_X(r,s) = \text{Cov}(X_r, X_s) = E[(X_r - \mu_X(r))(X_s - \mu_X(s))]. \tag{2.10}$$

□

Stationarity is then defined as:

Definition 2.3.2 (Stationarity)**.** A time series $\{X_t\}$ is *weakly stationary* if [8]:

$$\begin{aligned} &\text{(i)} \quad \mu_X(t) = \mu, \quad \forall t \in \mathbb{Z}, \\ &\text{(ii)} \quad \gamma_X(t+h,t) = \gamma_X(h), \quad \forall h, t \in \mathbb{Z}. \end{aligned} \tag{2.11}$$

□

Stationarity implies that the mean and autocovariance function of the time series do not change with time, making it possible to build time independent models for future behaviour of the time series. If the process considered is not stationary, it is first transformed to obtain these properties, please refer to Example 2.3.1 for details.

White Noise

When constructing time series models, white noise is an important building block. It is defined by:

Definition 2.3.3 (White noise)**.** A process $\{Z_t\}$ is *white noise* if [8]

$$\begin{aligned} &\text{(i)} \quad E[Z_t] = 0, \\ &\text{(ii)} \quad E[Z_t^2] = \sigma^2 \end{aligned} \tag{2.12}$$

and $\{Z_t\}$ is a sequence of uncorrelated random variables. If $\{Z_t\}$ is white noise with mean 0 and variance σ^2, it is written $\{Z_t\} \sim \text{WN}(0, \sigma^2)$. □

ACF and PACF

The autocovariance function and partial autocovariance function are important tools when working with time series data. If the time series $\{X_t\}$ is stationary, the time independent covariance function can be simplified:

Definition 2.3.4 (Autocovariance/autocorrelation function)**.** If $\{X_t\}$ is a stationary time series, the *autocovariance function* ACVF, $\gamma_X(h)$ is defined as [8]

$$\gamma_X(h) := \gamma_X(t+h,t), \tag{2.13}$$

and the *autocorrelation function* ACF, $\rho_X(h)$ as

$$\rho_X(h) := \frac{\gamma_X(h)}{\gamma_X(0)}. \tag{2.14}$$

□

The partial autocorrelation function is then defined by:

Definition 2.3.5 (Partial autocorrelation function)**.** If $\{X_t\}$ is a stationary time series, the *partial autocorrelation function* PACF, $\alpha_X(h)$ is defined as [8]

$$\begin{aligned} &\text{(i)} && \alpha_X(0) := 1, \\ &\text{(ii)} && \alpha_X(1) := \rho_X(1), \\ &\text{(iii)} && \alpha_X(h) := \rho_X(X_{h+1} - P_{X_2,\ldots,X_h}X_{h+1}, X_1 - P_{X_2,\ldots,X_h}X_1), \quad h \geq 2, \end{aligned} \tag{2.15}$$

where $P_{X_2,\ldots,X_h}X_n$ is the best linear predictor of X_n given $X_2, \ldots, X_h$. □

The sample versions, which can be calculated from real time series data, are

$$\hat{\gamma}_X(h) = \frac{1}{n}\sum_{t=1}^{n-|h|}(x_{t+|h|} - \bar{x})(x_t - \bar{x}), \quad -n < h < n, \tag{2.16}$$

$$\hat{\rho}_X(h) = \frac{\hat{\gamma}_X(h)}{\hat{\gamma}_X(0)}, \tag{2.17}$$

and

$$\begin{aligned} &\hat{\alpha}_X(0) = 1, \\ &\hat{\alpha}_X(1) = \hat{\rho}(1), \\ &\hat{\alpha}_X(h) = \hat{\rho}(X_{h+1} - P_{X_2,\ldots,X_h}X_{h+1}, X_1 - P_{X_2,\ldots,X_h}X_1), \quad h \geq 2. \end{aligned} \tag{2.18}$$

One important reason why they are good tools for time series analysis is that they have the properties that $\rho(h) = 0$ if $h > p$ for AR(p) models and $\alpha(h) = 0$ if $h > q$ for MA(q) models. Thus, they can be used for finding the order of those models from time series data by inspecting their sample autocorrelation functions. They are also useful for deciding if a time series is white noise or not. This is done by considering the bounds $\pm\frac{1.96}{\sqrt{n}}$ where n is the number of observations. If more than 95 % of the values of the ACF $\rho_X(h)$ for $h \neq 0$ fall within these bounds, $\{X_t\}$ can be considered white noise and no further modelling is possible [9].

Models

There are a couple of standard models for time series analysis and in this project the *vector autoregressive* (VAR) model was chosen for further analysis (the reason for this is explained in the end of this section). To give an understanding of the model, a couple of simpler models are first introduced [8].

AR(p) An AR(p) model is one of the simplest models for a weakly stationary process. In this model the future values of $\{X_t\}$ depends on the p previous values and the current noise realization:

$$X_t - \phi_1 X_{t-1} - \ldots - \phi_p X_{t-p} = Z_t, \quad Z_t \sim \mathrm{WN}(0, \sigma^2). \tag{2.19}$$

MA(q) A MA(q) model is similar to the AR model, but instead of depending on previous values of $\{X_t\}$ it depends on q previous values of the noise term:

$$X_t = Z_t + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q}, \quad Z_t \sim \mathrm{WN}(0, \sigma^2). \tag{2.20}$$

ARMA(p,q) An ARMA(p,q) model is a combination of an AR(p) model and a MA(q) model; it depends both on previous values of $\{X_t\}$ and of the noise $\{Z_t\}$:

$$X_t - \phi_1 X_{t-1} - \ldots - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q}, \quad Z_t \sim \mathrm{WN}(0, \sigma^2). \tag{2.21}$$

VAR(p) The VAR(p) model used for the implementation in this project is a multivariate version of the previously described AR model with structure

$$\mathbf{X}_t = \nu + \mathbf{A}_1 \mathbf{X}_{t-1} + \ldots + \mathbf{A}_p \mathbf{X}_{t-p} + \mathbf{U}_t, \tag{2.22}$$

where $\mathbf{X_t} = (X_{1,t}, \ldots, X_{n,t})^\top$, ν is a $n \times 1$ vector holding model constants, $\mathbf{A}_i$ are $n \times n$ square matrices with model parameters, $E[\mathbf{U}_t \mathbf{U}_t^\top] = \mathbf{\Sigma}$, $E[\mathbf{U}_t \mathbf{U}_{t-k}^\top] = 0$ for $k \neq 0$, and $\mathbf{\Sigma}$ is a positive definite covariance matrix [18, 31]. All VAR(p) models can be rewritten as a VAR(1) model with the notation [18]

$$\mathbf{X} = \mathbf{B}\mathbf{Z} + \mathbf{U}, \tag{2.23}$$

where

$$\begin{aligned}
&\mathbf{X} := [\mathbf{X}_1, \ldots, \mathbf{X}_t], \\
&\mathbf{B} := [\nu, \mathbf{A}_1, \ldots, \mathbf{A}_t], \\
&\mathbf{Z}_t := [1, \mathbf{X}_t, \ldots, \mathbf{X}_{t-p+1}]^\top, \\
&\mathbf{Z} := [\mathbf{Z}_0, \ldots, \mathbf{Z}_{t-1}], \\
&\mathbf{U} := [\mathbf{U}_1, \ldots, \mathbf{U}_t].
\end{aligned} \tag{2.24}$$

There are many other more complex models within time series analysis [18]. Examples are *vector autoregressive moving average* (VARMA), *autoregressive conditional heteroskedasticity* (ARCH), *generalized autoregressive conditional heteroskedasticity* (GARCH), *nonlinear generalized autoregressive conditional heteroskedasticity* (MGARCH) among others. VAR models are the only considered models for this thesis because of their simplicity and that they give good results for the available data set. If they would not have been sufficient for the task and time allowed, more complex models would have to be tested. However, as it can be seen in the results in Section 4.2, the VAR models give good predictive accuracy.

AIC

AIC is short for *Akaike's information criterion* and is a very useful measure for deciding which model to use. It is defined as [3, 17]:

$$V_{\text{AIC}} = 2\theta - 2\log(L) \tag{2.25}$$

where θ is the number of model parameters and L is the maximum value of the model's likelihood function for a given data set. The model structure that minimizes the AIC is often a good choice of model, since it has a good balance between few parameters (low value of θ) and a good representation of the data (high value of L).

Model Estimation

By using the general VAR(1) model from equation (2.23), the estimated model parameters $\hat{\mathbf{B}}$ can be derived to be [18]:

$$\hat{\mathbf{B}} = \mathbf{X}\mathbf{Z}^\top(\mathbf{Z}\mathbf{Z}^\top)^{-1} \tag{2.26}$$

As seen in the equation, estimation can be carried out by only considering the past observations of the variables contained in $\mathbf{X}$ and $\mathbf{Z}$.

Prediction

The best linear predictor $\hat{Y}$ of Y in terms of $X_1, X_2, \ldots, X_n$ is [12]:

$$\hat{Y} = a_0 + a_1 X_1 + \ldots + a_n X_n, \tag{2.27}$$

with constants $a_0, a_1, \ldots, a_n$ chosen such that the mean square error $E[(Y - \hat{Y})^2]$ is minimized. The predictor is determined by

$$\mathrm{Cov}(\hat{Y} - Y, X_i) = 0, \quad i = 1, \ldots, n. \tag{2.28}$$

In the case of the VAR processes (equation (2.22)) used in this project, the best linear predictor fulfilling this requirement is [18]

$$\hat{\mathbf{X}}_\mathbf{t} = \mathbf{A}_1 \mathbf{X}_{t-1} + \ldots + \mathbf{A}_p \mathbf{X}_{t-p} \tag{2.29}$$

This is intuitive to understand, since the best predictor is the same as our model with estimated parameters and with the noise term disregarded.

Granger Causality

The Granger causality test is a good tool for determining which time series could be useful for prediction of other time series [13]. One possible approach to define Granger Causality is the following [38]:

Definition 2.3.6 (Granger Causality)**.** Let $F = \{X_t, Y_t, X_{t-1}, Y_{t-1}, \ldots, X_1, Y_1\}$ where $\{X_t\}$ and $\{Y_t\}$ are time series. Then, if the variance of $\hat{Y}_{t+h}$ based on F is smaller than the variance of $\hat{Y}_{t+h}$ based on $\{Y_t, Y_{t-1}, \ldots\}$ for any positive lag h, X_t is Granger causal for Y_t with relation to F. □

In the case of VAR(1) models with only two variables a test for this causality is simple to perform. If the VAR model from equation (2.22) is written as

$$\mathbf{X}_t = \begin{bmatrix} X_{1,t} \\ X_{2,t} \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} \begin{bmatrix} X_{1,t-1} \\ X_{2,t-1} \end{bmatrix} + \mathbf{U}_t, \tag{2.30}$$

the Granger test can be performed by looking at the α values. The null hypothesis that $X_{1,t}$ does not Granger cause $X_{2,t}$ is then true if $\alpha_{21} = 0$ [39].

When testing models with $K > 2$ variables, $\mathbf{X}_t$ is split in the two parts

$$\mathbf{X}_t = \begin{bmatrix} \mathbf{X}_{1,t} \\ \mathbf{X}_{2,t} \end{bmatrix}, \tag{2.31}$$

such that $\mathbf{X}_{1,t}$ has K_1 variables, $\mathbf{X}_{2,t}$ has K_2 variables and $K_1 + K_2 = K$. Then, the VAR(p) can then be rewritten as [18, 31]:

$$\begin{bmatrix} \mathbf{X}_{1,t} \\ \mathbf{X}_{2,t} \end{bmatrix} = \sum_{i=1}^{p} \begin{bmatrix} \alpha'_{11,i} & \alpha'_{12,i} \\ \alpha'_{21,i} & \alpha'_{22,i} \end{bmatrix} \begin{bmatrix} \mathbf{X}_{1,t} \\ \mathbf{X}_{2,t} \end{bmatrix} + \mathbf{U}_t. \tag{2.32}$$

The null hypothesis that $\mathbf{X}_{1,t}$ does not Granger cause $\mathbf{X}_{2,t}$ is in this case defined as [31]

$$\alpha'_{21,i} = 0 \quad \forall i = 1, 2, \ldots, p. \tag{2.33}$$

Therefore, we test if there exists any $\alpha'_{21,i} \neq 0$ for $i = 1, 2, \ldots, p$. This statistic has the F-distribution $F(pK_1K_2, nK - p)$ [18], which is used in later analysis.

2.3.2 A Classic Approach

A classic method [8] for time series analysis is as follows:

1. Find the main features of the time series such as trends, seasonal components and outliers.
2. Remove the trend and seasonal components to get a stationary, detrended and deseasonalised time series.
3. Choose a model that fits the stationary time series.
4. Use the model for forecasting of the model residuals.
5. Add the previously removed trends and seasonal components to the prediction to get a forecast of the original data.

Consider the following example which illustrates the process:

Example 2.3.1. Figure 2.7 shows the average monthly atmospheric carbon dioxide levels measured at Mauna Loa, Hawaii, since 1958. A polynomial trend is plotted in blue. There is also an evident yearly seasonal component, and another seasonal component with a period of approximately 30 years. Figure 2.8 shows the data when the trend and the two seasonal components are removed. This data is used to build an AR(9) model and the sample autocorrelation of the residuals for one-step prediction of this model is shown in Figure 2.9. The sample autocorrelation of the residuals from the model resembles white noise, which means that no further

time series models are possible with this approach. The AR model is then used to forecast the residuals, and then the trend and seasons are added to get the forecast result in Figure 2.10. □

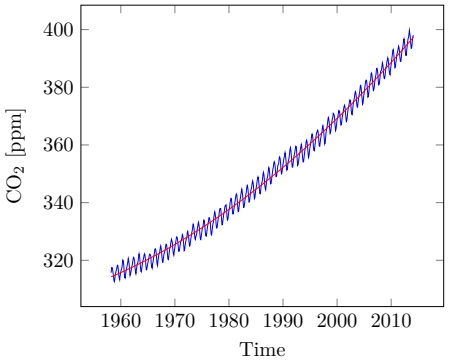

Figure 2.7: Original data and trend.

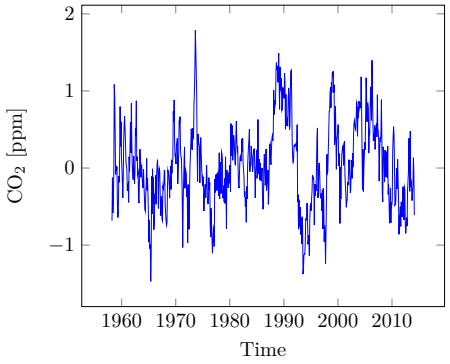

Figure 2.8: Data without trend and seasonal components.

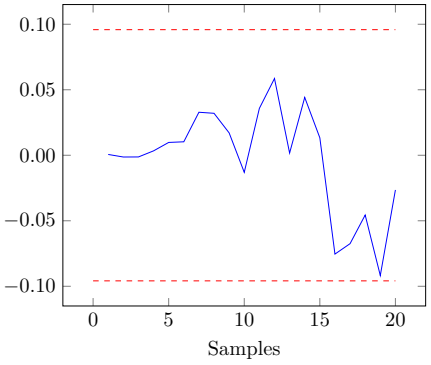

Figure 2.9: Autocorrelation of residuals for an AR(9) model.

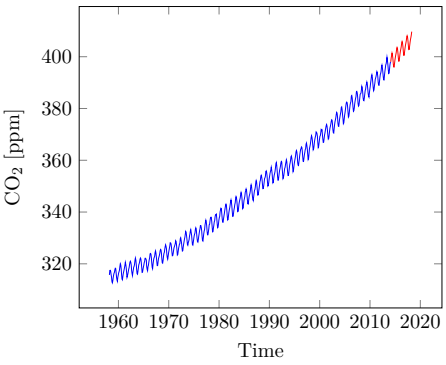

Figure 2.10: Forecast.

Chapter 3

Analysis and Implementation

In this chapter an analysis of the problem defined in Section 1.2 is presented. The implementation of the solution is also explained. The implemetation tools are first described in Section 3.1. Then, in Section 3.2, the classification part is detailed and the time series analysis can be found in Section 3.3.

Figure 3.1 gives an overview of the implementation and its three sub parts. The user interacts with a front end, which in turn handles the information flow to the classification and times series algorithms, as well as interacting directly with the database. Among other things, the user can display pricing data in raw numbers and in graphs. He can also enter classes manually or automatically with help from the classification algorithm. The time series analysis is used when the user requests a prediction of a specific class in the database.

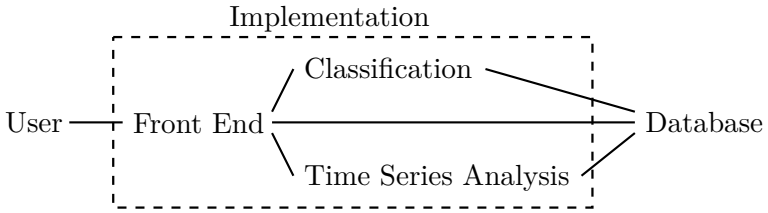

Figure 3.1: Implementation overview.

3.1 Implementation Tools

At the company as well as at many other workplaces the operating system in use is one of the systems from the Microsoft® Windows family. The database with product and price information described in Section 2.1 is of the type *Microsoft® SQL Server* [23]. There are many programming languages to choose from that

would be suitable for the implementation, but since the users and the database only use Microsoft® products, the programming language *C#* [25] was chosen. C# has many similarities with C, C++, Java and other object oriented programming languages. *Microsoft® Visual Studio* [26] was chosen as *integrated development environment* (IDE) because of its tight integration with C#, its excellent debugging and performance tools and also the simplicity of working with databases within the program. To have good control of the source code *Microsoft® Team Foundation Server* [24] was configured and installed on an external server. It was used both for work tracking and source control.

3.2 Classification

To reuse existing code for the C4.5 classification algorithm the scientific computing framework *Accord.NET* [2] was used. The full source code of the framework is available and the implementation of the C4.5 algorithm is also public [1]. The available code is a direct implementation of the algorithm described in Section 2.2.3.

The integration of the C4.5 algorithm consists of the following parts:

1. A method for creating a decision tree from a selection of products.
2. A method for storing a decision tree in the database.
3. A method for classifying a selection of products with a decision tree.

Each of these parts are explained in more detail below.

3.2.1 Creating a Decision Tree

As explained in Section 2.2, a set of training data is needed to build a decision tree that can be used for classifying other data. The data that needs to be classified is each product in the database with attributes of the data types shown in Table 2.2 on page 9. The idea of creating this decision tree is to simplify the classification work of the employees. The training data for building the tree is a couple of products in each class, which the employee has manually labelled.

In Algorithm 2, a high level overview of the implementation is shown. In the real implementation there are many help methods and data conversion algorithms used, but they are not relevant for understanding the structure. The algorithm starts by retrieving a list of product ids and a corresponding class to every id. This list has been created by an employee who has picked out a collection of products and manually labelled them in the front end application. The algorithm then gets all the products from the database from the list of ids and sets the corresponding class of each product.

Algorithm 2: Creating a decision tree.

Data: list of product ids and list of manually set classes
Result: a decision tree
database products ← all products from the database with id from input list;
set the class of all database products from input classes;
set the flag 'manually classified' on all database products;
update database;
codebook ← created codebook from all database products;
decision attributes ← set decision attributes to consider;
input ← all decision attributes from all database products;
output ← all manually set classes from all database products;
decision tree ← decision tree generated from input and output;

Each product in the database also has a boolean field called *manually classified* which is set to `true` for all the selected products. The purpose of this field is to not reclassify these products later on when the classification described in Section 3.2.3 takes place and also to label them as training data. This reclassification could otherwise happen if an employee for instance classified a single product in a class with products with very different attributes. In the creation of the decision tree this product would be considered an outlier and then it could get another class when classifying it at another instance. By labelling it as manually classified and ignoring it in further classification, this unwanted behaviour is avoided. When the database products are updated with a class and label, the changes are submitted to the database.

The implementation of the C4.5 algorithm used is not able to handle other than integer and float values. Integer values represent both discrete and categorical predictors (distinguished with a flag in the code), and float values represent continuous predictors. A *codebook* [2] is used to convert the other data types, for instance strings of text and boolean values. The principle of the codebook is very simple and shown in Figure 3.2. To the left in the figure, attributes with values that are not integers are shown. A codebook is created where each unique attribute value is assigned an integer code. The codebook is then used to change the attributes to integers so that they can be used in the algorithm. The same codebook is needed later in the implementation (refer to Section 3.2.3) to translate the integer codes back to their original values.

Decision attributes are the attributes of the products that the algorithm will consider when building the decision tree. Some attributes may be known to have no impact on which class the product belongs to. Examples of such attributes are the Id (as each product has a unique Id) and the previously mentioned flag that the class is manually assigned.

The input values are then all the chosen decision attributes from all the database products. The output values are the manually assigned classes that are given from

Codebook

Attributes	Value
A1	A
A2	B
A3	C
A4	A

→

Value	Code
A	0
B	1
C	2

→

Attributes	Value
A1	0
A2	1
A3	2
A4	0

Figure 3.2: Codebook example.

the parameter of the method. The input and the output values are then entered in the C4.5 algorithm which builds and returns a decision tree object.

3.2.2 Storing a Decision Tree

When new products are submitted to the database there is a need of automatically assigning them a class in the back end application. It would be very time consuming to rebuild the tree each time a new product should be classified, as many new products are added continuously. This gives rise to the need for storing the tree so it is easily accessible from multiple applications running on different computers.

Since a Microsoft® SQL Server database is already available a new table is added in order to store the trees. The simple design of this table is shown in Figure 3.3. It has an id field, which is unique for each entry, and two binary fields. The binary fields can store binary data with a specified length of bytes, i.e. a binary file. The first binary field is for storing the actual tree and the second to store the corresponding codebook. To be able to store the tree and the codebook in the database, they are first *serialized*. Serialization is a way to convert objects in the program to binary files [22]. The binary files are then stored in the database.

DecisionTree
Id Binary Tree Binary Codebook

Figure 3.3: Table to store decision trees in the database.

3.2.3 Classifying With a Decision Tree

When an employee has entered training data and a decision tree has been created (Section 3.2.1) and stored (Section 3.2.2) the next step is to use this tree to classify

other products. An overview of the algorithm for classification of products is shown in Algorithm 3.

Algorithm 3: Classification of products.

Data: list of ids of products to classify
Result: classified products
database products ← all products from the database with id from input list;
codebook ← gets codebook from the database with highest id;
binary decision tree ← gets decision tree from the database with highest id;
compiled decision tree ← compiles the binary decision tree;
forall the *database products p* **do**
    coded attributes ← converts p's attributes with codebook;
    coded class ← enter coded attributes in compiled decision tree;
    class for p ← convert the coded class with codebook;
end
update database;

The algorithm takes in a list of product ids of products to classify. The corresponding entries in the database (database products) are then retrieved. To be able to classify these products the last codebook and decision tree are also fetched. Because of the high priority on the speed of the classification due to the large amount of data processed, the binary classification tree from the database is first serialized and then compiled into executable code. Then each product's arguments are converted with the codebook and then fed into the compiled decision tree. The tree outputs a coded class which is coded back with the codebook to get the right class. When all products are classified, the changes are submitted to the database.

3.3 Time Series Analysis

This section holds an analysis and a description of the implementation of the time series analysis. In the analysis part, an example is presented to introduce the methods used. Then, the same methods are used with real data from the database. The implementation part explains which programming tools were used and also gives pseudo code for the algorithm.

3.3.1 Analysis

In this section an analysis of the structure of the available data is presented, motivating the implementation in Section 3.3.2. First, we consider a simplified example to get an insight in the methods used:

Example 3.3.1. In Figure 3.4 the price evolution of three example products is considered for approximately 40 days. These three products have the exact same price evolution except that Product 2 is shifted one day in relation to Product 1, and Product 3 is shifted two days in relation to Product 1. One could say that in this class of products, Product 1 is the market leader and sets the price that the other products are following. This is a behaviour that is frequent in real data. However, in real data it has much higher complexity in the sense that there could be more than one market leader in a class, the products don't follow each other exactly one day after, etc.

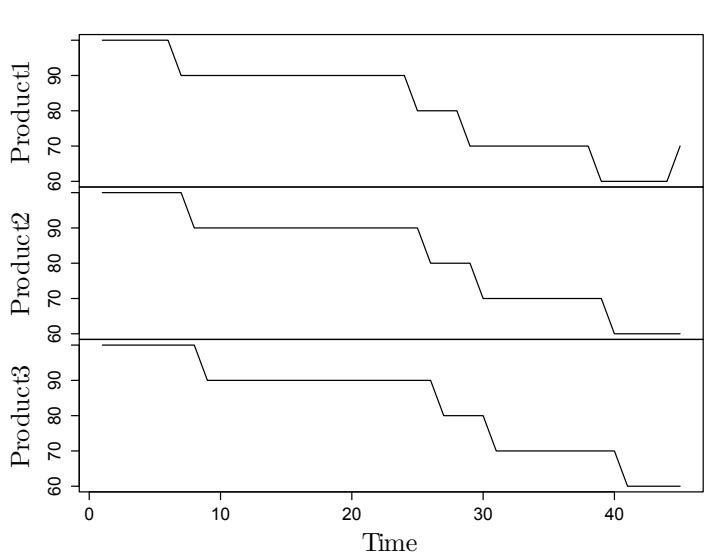

Figure 3.4: Example price evolution for three products.

In Figure 3.5, the ACFs for the differentiated price evolutions of the example products are shown. The three diagonal sub plots are the sample autocorrelation with the product itself, and the other sub plots are the sample crosscorrelation between each two of the products. Notice how the three diagonal plots satisfy the hypothesis of white noise sequences discussed in Section 2.3. This essentially tells us that the price evolution for each product when considered separately seems random and it is not possible to build a model that predicts the future price evolution in a reliable way. The best that can be done in this case with the presented theory is to estimate the trend and stop there. However, the interesting information is in the three sub plots called `Prd2 & Prd1`, `Prd3 & Prd1` and `Prd3 & Prd2`. Notice the single high peak at lag -1, -2 and -1 in each plot respectively. This corresponds to that Product 2 is a 1-day delayed version of Product 1, Product 3 is a 2-day delayed version of Product 1 and Product 3 is a 1-day delayed version of Product 2. This is not entirely a surprise, since it is exactly how the example was built. However,

this means that it is possible to build a relevant VAR model of this time series.

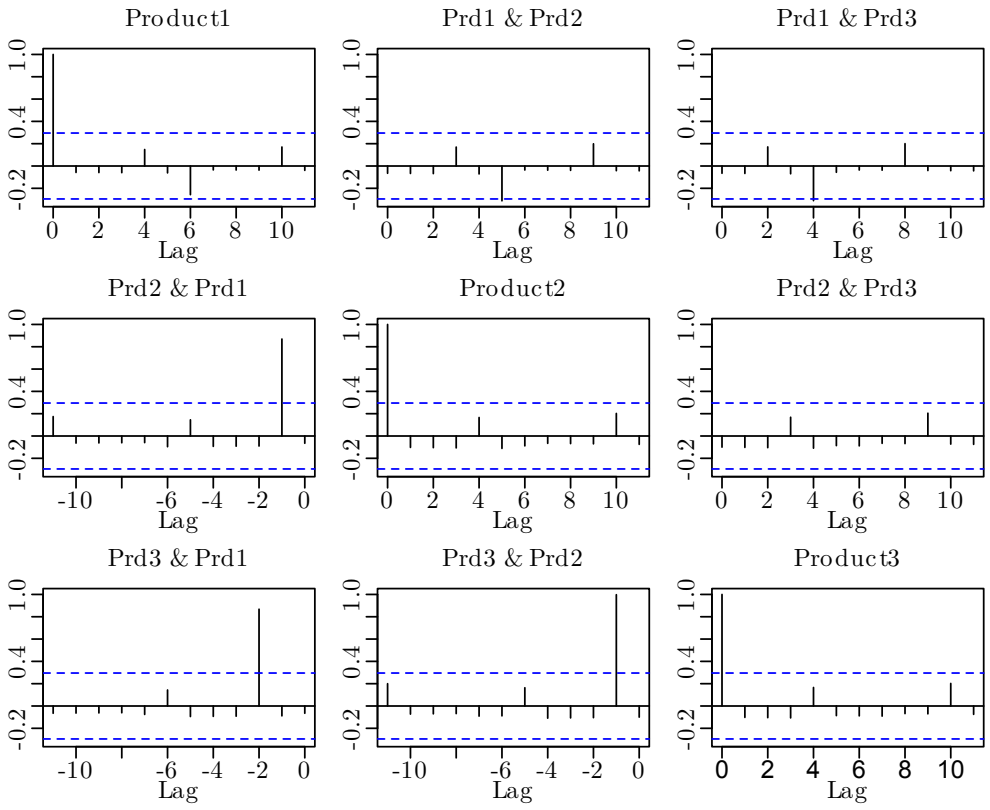

Figure 3.5: The ACF of the differentiated example products.

Figure 3.6 shows the result when making a prediction of each product from the estimated VAR model. The blue line is the best prediction and the red lines show 95 % confidence bounds for the prediction. The forecast for Product 1 is simply that it will follow the estimated trend and may also have some weak dependence on the earlier behaviour. The forecast for Product 2 on the first forecasted day is that the price will go up exactly as it did for Product 1 the day before, which makes a lot of sense. Notice how the confidence bounds follow the same behaviour as the best prediction. This stems from the fact that Product 2 has always in the past followed Product 1 exactly one day after. Therefore, according to our model, it will continue doing that. However, the prediction for days after the first one will have a higher uncertainty and be equal to the prediction for Product 1. The forecast for Product 3 follows the same pattern where during the first two days it will first have constant price, then follow Product 1 and 2 in the price increase and then also have the same prediction as Product 1 had the first day. □

When working with real data the situation is more complex as mentioned previously,

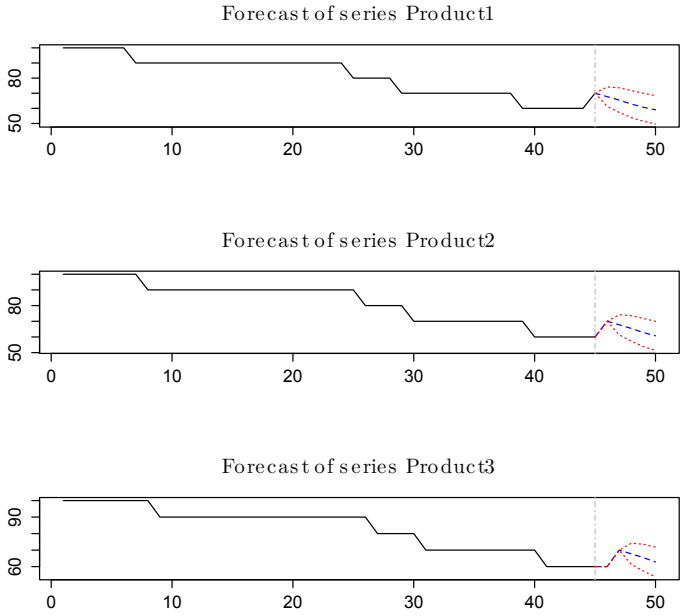

Figure 3.6: Prediction of the example products.

but it still follows the same idea as the idealized example. In the implementation all products are modelled with all other products in the same class, but for the analysis in this section the mean price in the class is chosen instead to limit the number of plots. Figure 3.7 shows the price evolution for a product and the mean price in the class the product belongs to. In the figure it can be seen that they follow each other a bit, which is a common behaviour also for products of the same class.

Figure 3.8 shows the ACF of the differentiated products and mean. By studying the figure one can see that, as in the simple example, the product and the mean by themselves satisfy the white noise hypothesis and it is not possible to build models with good prediction capabilities using general time series analysis. The sub plot `Product & Mean` shows lack of correlation, which gives the information that the product does not follow the mean price in the class. The interesting information can be gained in the sub plot `Mean & Product` where it can be seen that there is some correlation approximately for lags from -1 to -7. This means that the mean price actually follows the product's price and it is possible to build a VAR model for this behaviour. In this example it is the mean price that follows the product's price, but in the real implementation correlation is found between different products in the same class. In this way one can see which products in the class are setting the price and which products that follow others. Some products can also be completely uncorrelated with all other products in the class and then the model will not give better predictions than simply estimating the trend.

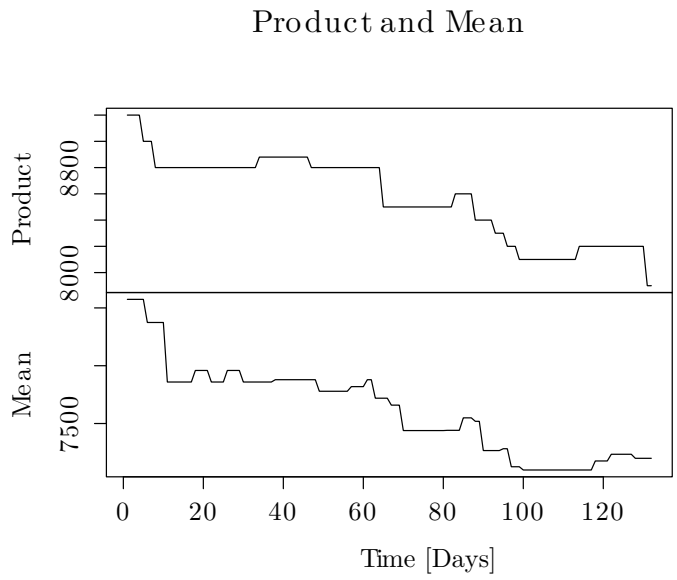

Figure 3.7: Price evolution for product and mean.

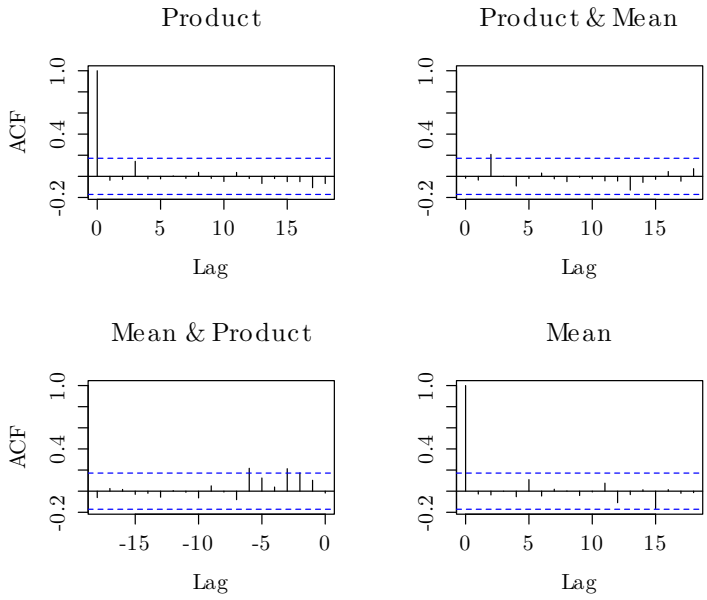

Figure 3.8: The ACF of the differentiated products and mean.

In Figure 3.9, the price prediction is shown for both the product and the mean, just after a change of the product's price. As inferred by the earlier correlation,

the prediction for the mean price is that it will also go down in the next couple of days, as it can be seen in the figure. However, the product's price evolution is more difficult to predict, since it is not correlated with previous values of itself or with the mean price, as previously discussed. Therefore, the best prediction is mostly based on the estimated trend and it has a relatively large confidence interval, implying that the prediction has low accuracy.

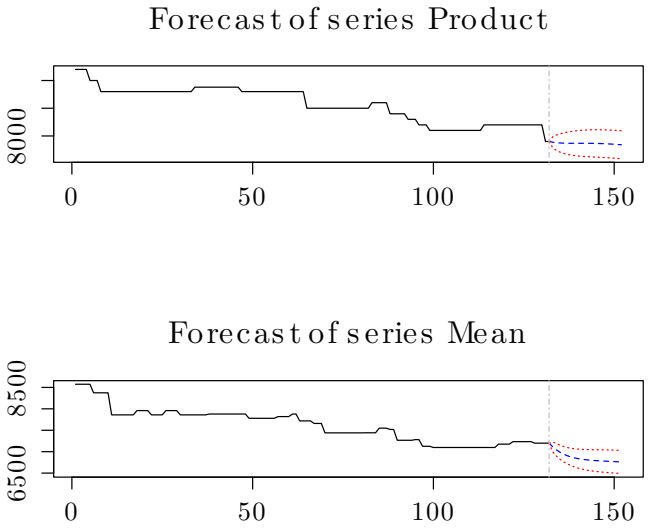

Figure 3.9: Prediction of the product and mean.

3.3.2 Implementation

There are many software solutions for working with time series analysis. Matlab and R are both two popular alternatives. Since R is open software [34] and also integrates well with C#, it was chosen for this project's implementation. The integration with C# is handled with the library *R.NET* [35]. This library enables R code directly into the source of the project and also makes it simple to convert between different data types.

The time series algorithm implementation has a single yet difficult task: it should be able to receive a number of products with their temporal pricing information, and for each product it should predict the future price evolution. An overview of the implementation is shown in Algorithm 4.

Algorithm 4: Prediction of products.

```
Data: list of products to predict and prediction length
Result: list of products with prediction
if prediction length larger than zero then
    global first and last price point date ← 0;
    forall the database products p do
        find the local first and last price point date for p;
        if any price point missing in between, linearly interpolate;
    end
    global first price point ← last local first price point date;
    global last price point ← last local last price point date;
    start R engine with required packages;
    convert all product price point data to R format;
    build a univariate time series for each product;
    build a multivariate time series consisting of all products;
    create VAR model that minimizes AIC and considers trend;
    predict each univariate time series with the VAR model;
    convert prediction data back from R format and add to products;
    return products with prediction;
else
    return products without prediction;
end
```

The purpose of the algorithm is that it should be able to receive a set of products with corresponding price information and do a price prediction on each product according to the analysis in Section 3.3.1.

If the prediction length is larger than zero, meaning that a prediction should take place, it starts by going through all the products that should be predicted and finds the first and the last recorded price for each product. For some of the products there is missing price information for a couple of dates between the first and last points recorded. Then the price is linearly interpolated to get a full set of data. When all the individual first and last recorded price point dates have been found, a global first and last date are set. It is between these two dates that the time series model is built. Figure 3.10 shows the resulting time window for this method.

Then the R engine is started with R.NET [35] and the required packages are loaded. Since the database formats differ from the ones used by R.NET, some conversions take place (they are not explained here due to space limitations). Then, the method outlined in Section 3.3.1 is implemented in R code. First, a univariate time series structure is built for each product's price points. Then, all these univariate models are put together into a single multivariate time series structure. From this multivariate time series the VAR model that minimizes the AIC and also considers trends is estimated according to the theory in Section 2.3. This model is then used

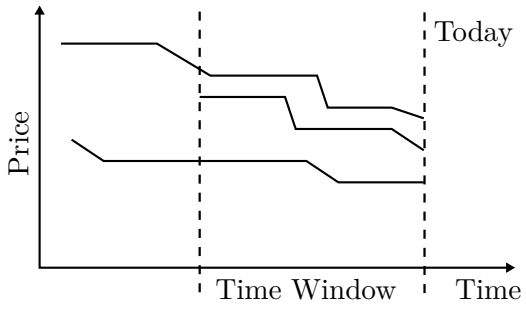

Figure 3.10: Time window selection.

for prediction of each univariate time series. The prediction data is converted to a suitable format and then added to each product. Now the prediction is done and the products are returned with the included predictions.

3.4 Front End

When working with C# the preferred framework for writing Graphical User Interfaces (GUIs) is *Windows Presentation Foundation (WPF)* [27]. WPF uses the *Extensible Application Markup Language* (XAML) to structure design elements, in a similar way as HTML is used for websites. Instead of writing the XAML code manually, Blend [21] was used for designing the GUI.

During the development of the front end, several meetings with the employees that use the GUI was held. During these meetings the current state of the program was displayed and feedback was retrieved. It was an iterative process where several different design approaches were implemented and tested between the meetings.

One important implementation detail is that each new action in the program, for instance the creation of a decision tree, was started in a separate thread from the GUI, so that the interface always stayed responsive. A progress bar and cancel button were also important tools for letting the user know if the program was busy or not.

The resulting front end application is further described in Section 4.3.

Chapter 4

Results

This section presents the results from the classification, time series analysis and the front end which can be found in Section 4.1, Section 4.2 and Section 4.3 respectively.

4.1 Classification

There are mainly two important requirements from Section 1.2.1 when considering the creation of the decision tree: (i) it should be fast enough to build and use the tree in real time and (ii) the quality should be good enough to replace manual classification. The speed results for creating and classifying with the tree are presented in Section 4.1.1 and Section 4.1.3 respectively, and the quality results in Section 4.1.4.

4.1.1 Creating a Decision Tree

Figure 4.1 shows how the speed of creating the decision tree with the implemented algorithm (from Section 3.2.1) varies with the number of products that are manually classified. Each data point is the result of the average of 10 runs through the decision tree creation algorithm and the distance between each data point is 10 number of products (which means that $10 \cdot \frac{1000}{10} \cdot 2 = 2000$ decision trees were created for this result). One can see that even for up to 1000 manually classified products with five decision variables, the time for creating the decision tree is less than 400 [ms]. In real applications the amount of manually assigned classes will be far less, and the time for creating the decision tree will often be less than 100 [ms]. One can also see that the dependence between the number of products and the time for creating the tree is approximately linear. The test was performed on an HP EliteBook Revolve 810 laptop with specifications shown in Table 4.1. This laptop was chosen because

Component	Specification
Processor	Intel Core i5 3437U, up to 2.9 [GHz]
RAM	4 [GB], 1600 [MHz]
Hard Drive	Samsung MZMPC128HBFU, 128 [GB] mSATA SSD

Table 4.1: Test computer specifications.

it has the same or lower specifications than the employees using the software have, which means that algorithm will be even faster when they use it.

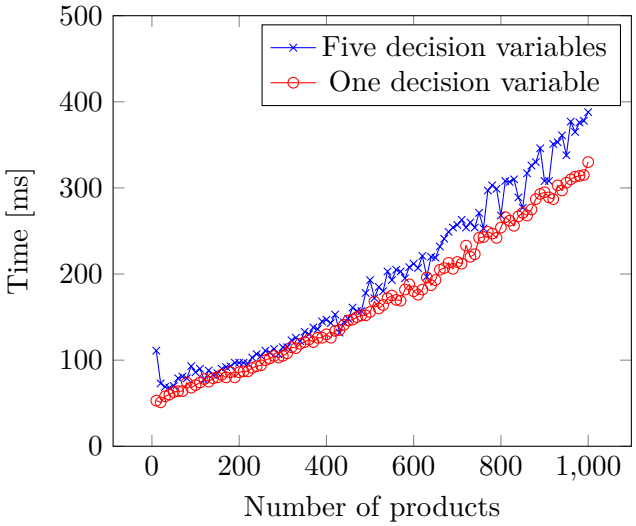

Figure 4.1: Tree build time for different number of products.

In Figure 4.1 the result is shown for one and five decision variables. One can see that it takes slightly longer time when there are five decision variables. Figure 4.2 highlights the effect on the building time when increasing the number of variables. Each data point is the average of a 1000 created decision trees with the specified amount of decision variables. As in the case of increasing number of products, the increasing number of decision variables also seems to increase the classification time linearly. However, the dependence is mild and if the number of decision variables increases with one, the algorithm approximately only takes 15 [ms] longer.

4.1.2 Storing a Decision Tree

The time for storing the created decision tree on the server highly depends on what the server is doing at the moment as well as how much traffic there is on the network at the time. Therefore, no results of this time consumption are presented. However,

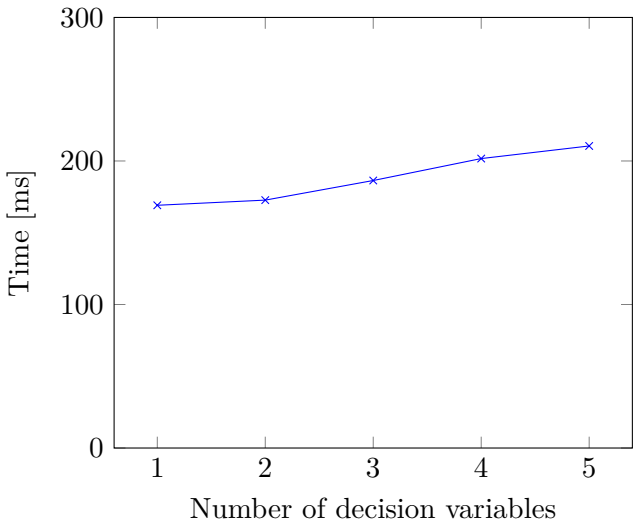

Figure 4.2: Tree build time for different number of decision variables.

the time consumed on that is generally a few seconds since the size of the tree is at most a couple of megabytes.

4.1.3 Classifying With a Decision Tree

The only requirement on the algorithm for classification of new products with an already existing decision tree is that it should be fast enough so that the employees do not have to wait long for the results. Figure 4.3 shows the classification time for different number of products to classify. In this test a tree with a specified number of decision variables is created from a set of 50 products. The number of products that the tree is created from does not vary in this test, because it should not alter the classification time. Then, from 10 to 1000 products are entered in the decision tree and get a class assigned according to the theory in Section 2.2. Each data point is an average of 10 such measurements.

In Figure 4.3 the result is shown for one and five decision variables. One can see that there is no clear trend that products with more variables will take longer time to classify. In Figure 4.4 it is even more clear. In that figure the result is shown for one to five decision variables. Each data point is an average of a 1000 product classifications with the specified amount of decision variables. One can see that the time consumed only varies a couple of milliseconds and can for these values of decision variables be considered constant.

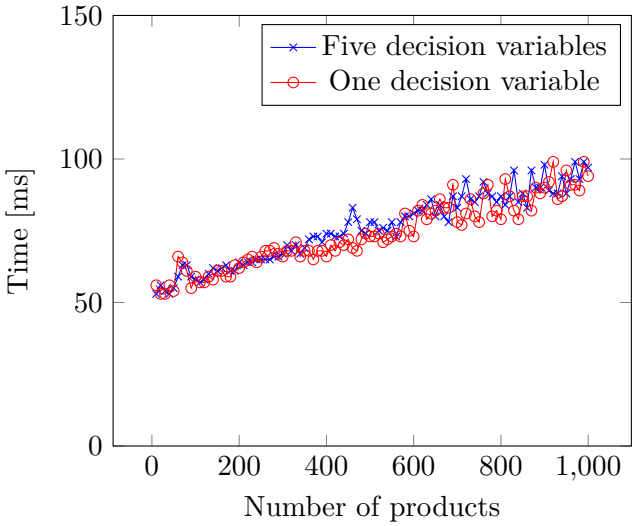

Figure 4.3: Classification time for different number of classification products.

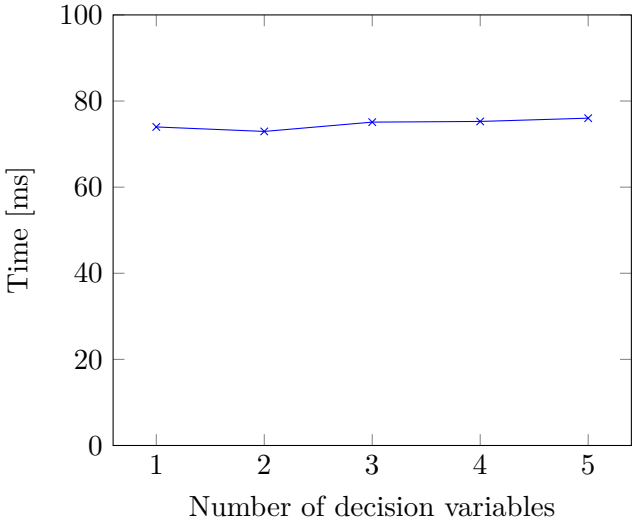

Figure 4.4: Classification time for different number of decision variables.

4.1.4 Quality of the Classification

In contrast to the speed of the classification algorithms, the quality of the result is very difficult to measure. It is the employees using the tool who decide if it is good enough. In short, one can say that if the result from the classification algorithm is close to the same that an employee would manually classify, it has a high quality.

Correct	Almost Correct	Incorrect
44	7	9

Table 4.2: Result of the classification quality test.

A simple test for measuring the quality according to one employee was performed. The test procedure was as follows:

1. The employee manually classified 30 random products at one vendor with approximately 200 products.
2. The manually classified products were used as training data for the classification algorithm. The algorithm then classified all other products at the vendor.
3. 60 classified products from the vendor, not belonging to the training data, were randomly chosen as validation data. The employee then gave a verdict on each product classified with the classification algorithm. The verdict could be that the classification was correct, almost correct or incorrect.

The result of the test is shown in Table 4.2. In this test, 73 % of the products were correctly classified according to the employee. The 12 % of the products that were almost correct, were all given this verdict because the attributes of the classified product were close to the given class, even though it was incorrect. 15 % of the products had completely wrong classes.

The test presented, although simple and only tested with one employee, clearly shows that the classification algorithm can satisfactorily classify the products in a similar way as an employee would. The main purpose of the algorithm is to make the classification faster, and the test shows that by only classifying 30 products, approximately all other products at the same vendor (and other) can be classified with a 70 % accuracy. This allows the employee to iteratively increase the quality of the classification by first giving a few training classes, let the algorithm classify and then quickly look through the results and make corrections. The training algorithm is then applied again, reaching a better classification. This work flow is much faster than manually classifying all of the products.

4.2 Time Series Analysis

First, we consider the results of the method from the example in Section 3.3.1 where the price evolution of a product and the mean price in a specific class were considered. A method for analysing how well a model fits the data is to divide the data into a model and a validation part (referred to as *cross-validation* [8]). The

model is then estimated with the model and validated with the validation data. In Figure 4.5, these two parts for the product and mean price are shown.

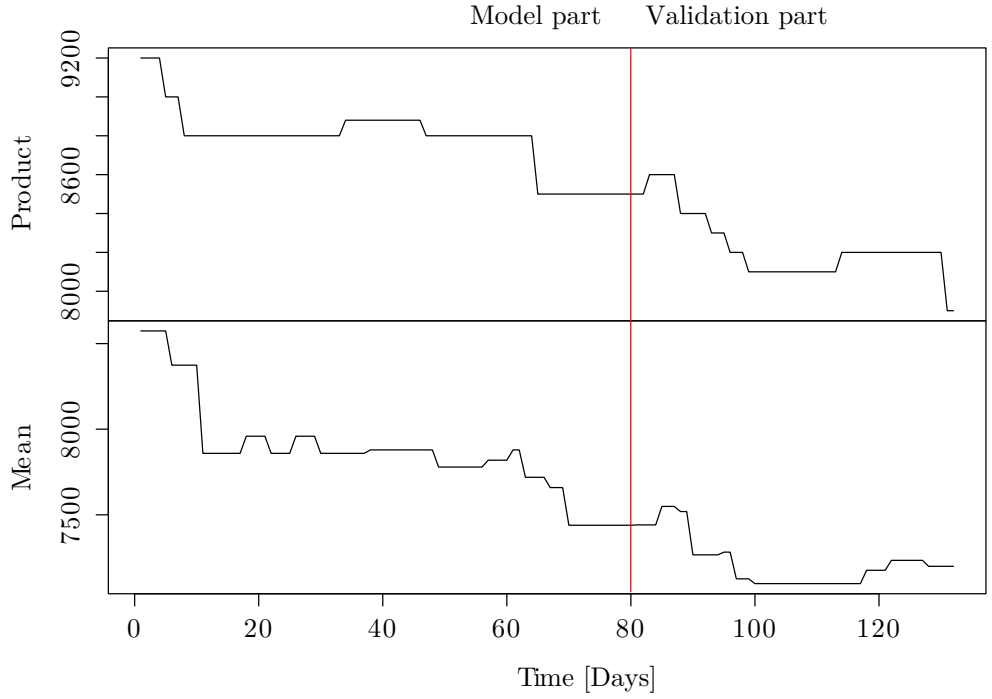

Figure 4.5: Model and validation part for product and mean price evolution.

As described in Section 3.3.1, the specific product considered is the market leader in this example, in the sense that it is useful for predicting future values of the mean price in the class. The Granger causality test gives a p-value of $p_v = 1.894 \cdot 10^{-5}$, which implies that this is indeed the case. The interesting residuals to look at are the ones for the mean price when estimating the VAR model from the model part and validating it on the validation part. These residuals are shown in Figure 4.6. To set them in context, the residuals from using a very simple model are also shown. This model simply predicts the next price to be the same as the day before.

There are several interesting observations from these figures. We first notice that the simple prediction is accurate for almost all days, leading to zero residuals for these days. This is because the price for most times stays constant. However, at every day where the price changes, the simple model fails and the residual is exactly the difference in today's price in comparison with yesterday's one. Therefore, it is not a good predictor for days when the price really changes, which of course are the changes that are interesting to model. The residuals for the VAR model look a bit

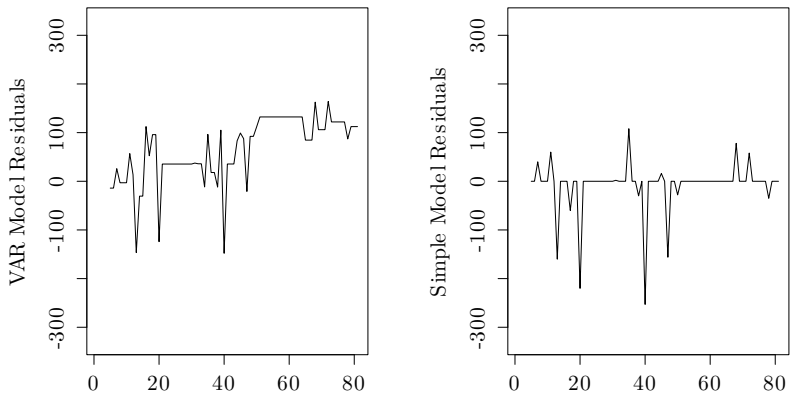

Figure 4.6: Residuals for the mean price with the VAR and simple model.

different in a couple of ways. They are more fluctuating because they also consider the price of the product. This makes it a worse predictor than the simple model for all days when the price doesn't change. This is because the simple model will have full accuracy on these days, whereas the VAR model has small residuals, in this example in the range of 0-100. However, we notice that at the days when the price changes, the VAR model has noticeably smaller residuals than the simple model. This could give information that the VAR model indeed predicts those changes better than the simple model.

Another, less good, feature of the VAR model can also be observed in in Figure 4.6. That is the VAR model's residuals drift from zero the further the prediction is in the validation data. This also explains a bit why it has lower residuals than the simple model, because it is gradually shifted upwards and the residuals are mainly negative. This comes from the fact that the trend estimated in the model part is no longer valid for the last data points. The problem could in part be solved by choosing a more complex model with a higher degree for the trend polynomial, for instance a second degree polynomial. However, this problem does not occur when the real algorithm without a model and validation part is used. In that case the trend is estimated on the whole data set and the prediction only takes place a few steps ahead, so this behaviour does not impact the final result very much which is shown later in this section.

Figure 4.7 shows another analysis of the modelling. In this figure a "moving model" have been used, which better illustrates how the results will perform in real situations. The "moving model" part expands for each time step such as at each time instant, the model is estimated from all the observed data up to this step. Then a prediction is done and compared with the validation data and generates the resid-

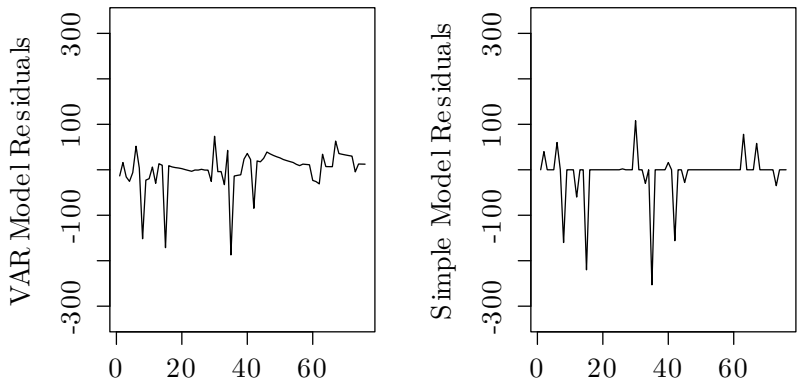

Figure 4.7: Residuals for the mean price with the VAR and simple model using a moving model part.

uals for each step. This corresponds to the situation when one at each day predicts the price of a product a couple of days ahead, with knowledge of all the previous prices for all products. The first thing to notice when comparing Figure 4.6 and Figure 4.7 is that the problem of the drift from zero in the residuals for larger values is gone. Another thing to notice is that the difference between the VAR model and the simple model, when considering the peaks of the residuals, is smaller. The reason for this is that the upwards trend is removed, as discussed briefly in the previous paragraph.

Figure 4.8 shows the histogram of the absolute residuals for the VAR and the simple model when using the moving model part. There, the previous observations can be seen more clearly: the VAR model has slightly better predictions, especially in the sense that it has fewer large residuals. The corresponding root mean square error is 42.1 for the VAR model and 50.7 for the simple model, which again shows that the VAR model is a better choice in general.

All of the observations above are from a single example from the database, even one with only three months of recorded data. This example was chosen because it has properties that can be seen throughout the whole database within many of the classes. To show that this is the case, a program was written to perform the Granger causality test on a part of the database. The program considers 12 randomly chosen classes and in each class calculates the Granger causality test for 5 randomly chosen products within each class. For each product the price data from six months were used. As explained in Section 2.3, we can discard the null hypothesis that the product do not Granger-cause any of the other products in the class if the p-value

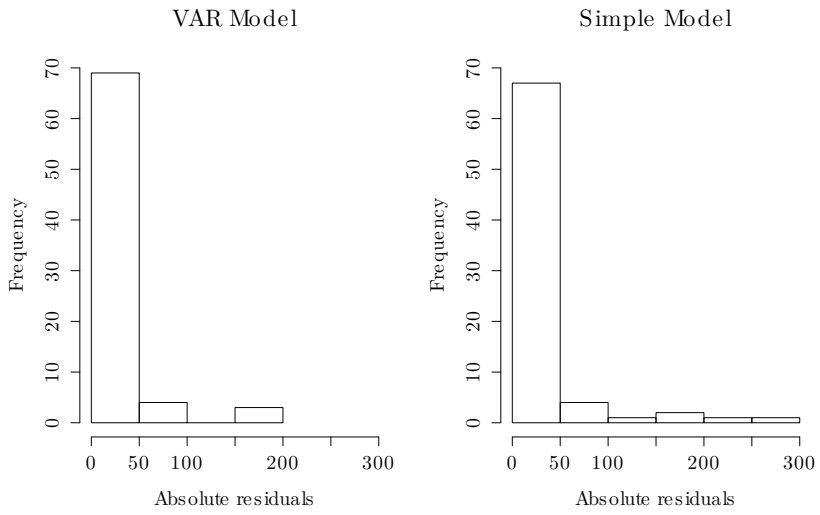

Figure 4.8: Histogram of the absolute residuals for the VAR and simple model.

is small. A common value to choose for p_v is $p_v = 0.05$.

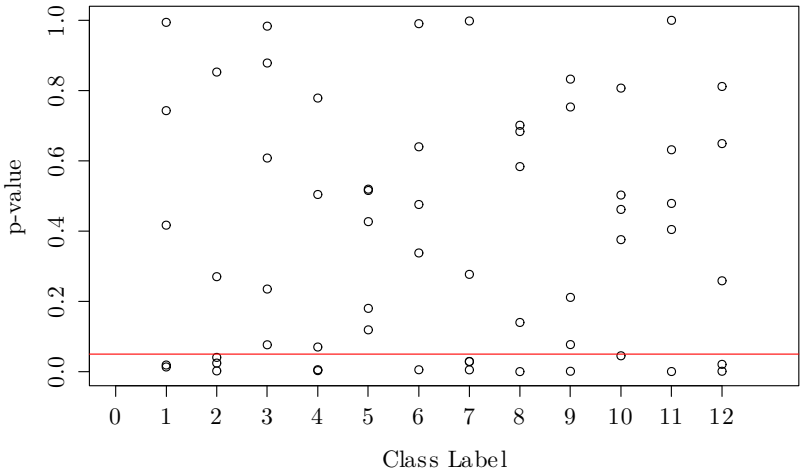

Figure 4.9: Resulting p-values for the Granger causality test for 12 test classes with 5 products in each class.

Figure 4.9 shows the results of this test. Each of the 12 test classes are shown with the p-values for the 5 products. The p-value $p_v = 0.05$ is also shown with a line in the figure. The interesting result here is that in all classes (except for class 3 and

class 5) there is at least one product where the null hypothesis can be discarded, i.e. the product is useful for predicting other products in the same class. Since the products tested only are a subset of the products in each class it is possible that another product in the same class can also be used for prediction.

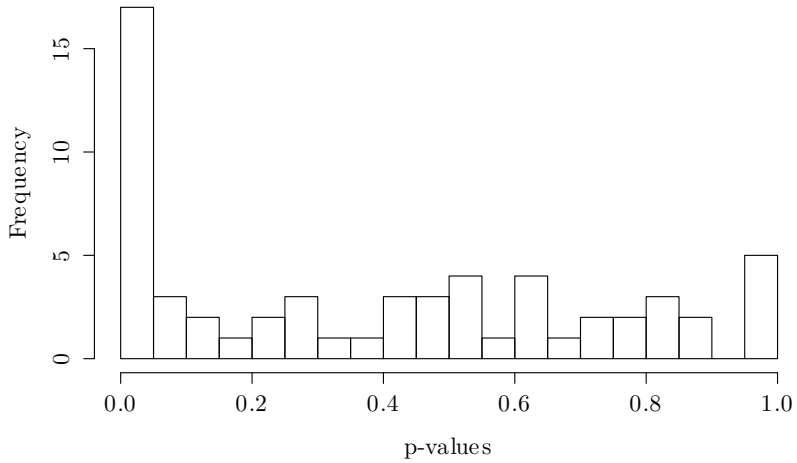

Figure 4.10: Resulting p-value histogram for the Granger causality test for 12 test classes with 5 products in each class.

Figure 4.10 shows a histogram of the p-values of all the Granger causality tests for the classes and products from the database. The histogram has a large peak for p-values close to zero and the rest of the histogram looks uniformly distributed. Each bin in the histogram is 0.05 wide, which means that the large peak close to zero corresponds to p-values less than 0.05. The products with p-values in this bin represent 28.3 % of the total number of tested products. This shows that close to a third of all the products considered are, according to the Granger test, useful for predicting future values of the other products within the same class.

In conclusion: the results in this section first gave a specific example of a product and the mean price in the same class to explain how the results look like for a typical prediction. Then, to motivate why the results are similar for the rest of the data, a set of Granger tests were performed on a subset of the database. The results of this test show that there often exists at least one product in each class which is useful for predicting the price evolution of the other products in the same class.

4.3 Front End

The final front end consists of two main tabs. The first is called *Pricing Data* and this tab is for displaying product information and for sorting them into classes for further analysis. The second tab is called *Plot Data* and this is where the price information for the products and the predictions can be viewed. Each tab is explained in the following two sections and a user manual for the front end can be found in Appendix A.

4.3.1 Pricing Data

The tab Pricing Data is shown in Figure 4.11. Some elements in the Figure are blurred out due to confidentiality. To the left in the tab there are tools for selecting and manipulating data. To the right in the tab there is a data grid which can show data and also handle classification input. The functionalities in this tab are:

- Display product pricing and properties for all products at selected vendors or a single product at a selected date.
- Display and edit product classes.
- Update the classes in the database manually or automatically with the implemented classification algorithm.
- Clear all or selected classes for products in the database.

4.3.2 Plot Data

The tab Plot Data is shown in Figure 4.12. Some elements in the Figure are blurred out due to confidentiality. To the left in the tab there are tools for selecting what data to display. To the right in the tab there is a result window where graphs are plotted with the open source library Oxyplot [30]. The values of the X-axis of the graph are dates and the Y-axis values are prices. The functionalities in this tab are:

- Plot a class of products with a selected prediction length.
- Add a single product to the graph.
- Clear the graph.

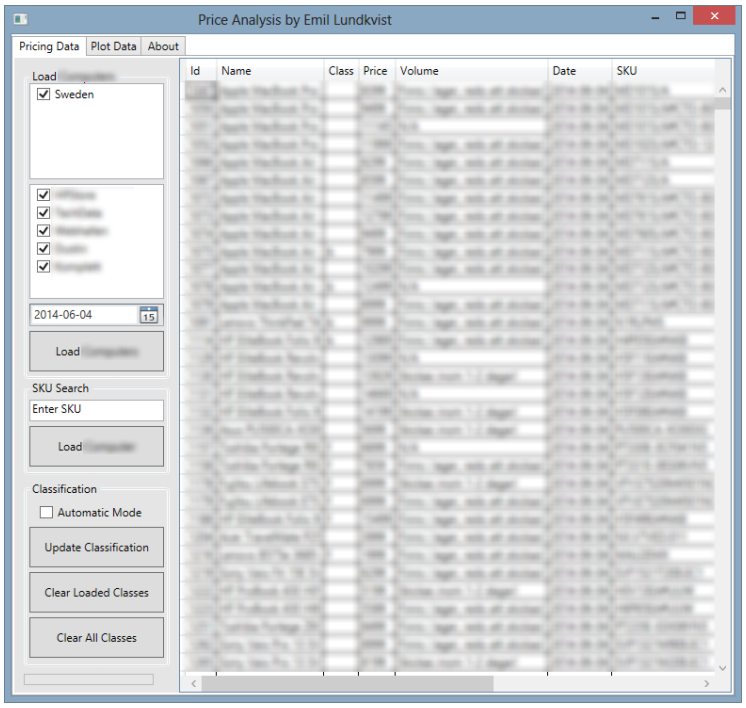

Figure 4.11: Front end pricing data.

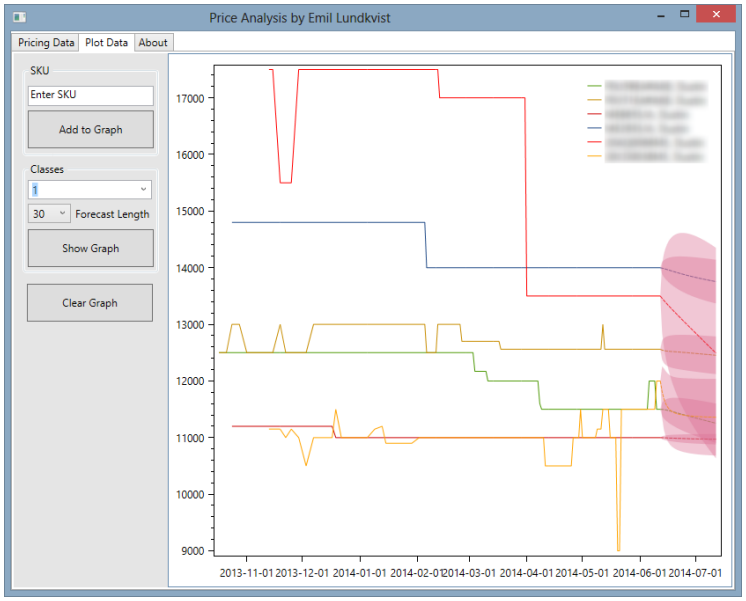

Figure 4.12: Front end plot data.

Chapter 5

Conclusions

The work described in this thesis has covered the analysis and implementation of a classification and forecasting program using a large database containing product pricing information. Different methods and algorithms for solving these problems have been discussed, implemented and tested.

For the classification problem, a decision tree approach was chosen because of its ability to handle categorical predictors and the ease of interpreting the results. To make forecasts of the product prices within each class, methods of time series analysis were applied and VAR structures were chosen as a model for the data.

The results from the classification implementation show that it was indeed a good approach, because of the measured speed of both the construction of the tree from manual training data and the classification. The quality of the classification was also good enough to save time in comparison with manual classification work. The results from the time series analysis focused on a single example showed that in this specific case the applied model could be used for prediction. It was also shown that the same method can be used on larger sets of the database. A front end was also developed to enable users to classify products, view pricing information and make forecasts of the future prices within each class.

5.1 Future Work

Although the tool described in this thesis is already distributed in the company and works as it was intended, there are of course many improvements that can be made. The most important ones are described in the following sections.

5.1.1 Classification

One of the main reasons for choosing a decision tree for classification was that it can be easily interpreted by the end user. While that is true, there is currently no function in the front end for doing this visualisation. The only way to see the decision trees is to extract each decision tree from the database and manually interpret it directly in the code. While this has been good enough for debugging during the implementation, it is not sufficient for comparing trees or analysing many at the same time. It would be preferable to have a dynamic graphical view of the tree in the front end of the application, looking something like the example in Figure 2.4 on page 13. This would not be difficult to achieve since the structure of the tree object in the database is very simple. Another feature that would be very useful in this tool is to enable editing of the trees. If the end user sees a decision in the tree that he thinks is a bit wrong for his classification it would be good if he could click the node and change it to his liking.

The tree learning algorithm C4.5 has a tendency to create unnecessarily large decision trees that have small errors when classifying the training data, but are less good at classifying new combinations of parameters. This is an example of over-fitting of the model. *Pruning* can be used to reduce the size of the decision tree and hence also the over-fitting. Methods for pruning exist [32] and would not be difficult to implement and test. However, this modification would most probably only result in a minor improvement of the general classification, as it's already good enough for the current usage.

5.1.2 Time Series Analysis

There are many aspects of the time series analysis that could be improved. The most important would be to test many other models for representing the data, for example the previously mentioned models VARMA, ARCH, GARCH and MGARCH [18], refer to Section 2.3. It would be interesting to compare the results and conclude which model that best explains the data.

Another interesting feature that would be useful for the end users would be to see how a change in a single product's price would affect the price of the other products in the same class. One could for example increase the price of one particular product by a fixed amount and see which other products that would follow and approximately by how much. This is possible with *Impulse Response Analysis* [18]. Methods for this kind of analysis are included in the R package "vars" [31]. It would be a very useful feature for the employees and not very difficult to implement.

In the current implementation it is only possible to do predictions for all products in a single class. It would be of interest to see also how the classes interact. One could for example study the mean prices of each class and see how they change

in relation to other classes to be able to make predictions on a larger scale. This analysis would require a large amount of code to be written, but the same theory as in the single product case could be applied. It would also be of interest to build and evaluate models considering external economical factors such as interest and exchange rates.

Another issue is that when an end user creates a class and gets a forecast, there is no way for him to see what the prediction is based on. This is an important tool for the end user as he combines the tool's prediction with his own experienced analysis. If he knew which products influenced a single product's price, this merge of predictions would be much easier. This would be a pure front end improvement and no more than the current theory would be needed. All the information is held in the estimated VAR model's parameters, so the implementation would simply consist of displaying this data in a satisfactory manner.

To linearly interpolate missing prices (described in Section 3.3.2) may not be the best way to get data at all points in time. One could try other methods for this, such as *multiple imputation* [6]. These methods could be implemented and tested to see which one that has the best predictive properties for the data set.

In the results in Section 4.2 two different validation methods were used. The first had a fix model and validation part, whereas the other had a moving boundary between them. However, in both methods the whole parts were used and information from long ago was used to build the model. In the case of pricing data, it could be useful to use *recursive estimation* algorithms [19, 17] which gives less weight to observations in the past.

Appendix A

Front End Manual

A.1 Introduction

This manual describes the interface and functions of the program *Price Analysis*. The main purpose of the program is to display and interact with the existing database with price information at the company.

Section A.2 describes the interface of the program and Section A.3 gives instructions to the included functions.

A.2 Interface

The interface consists of two main tabs, *Pricing Data* and *Plot Data*, which are detailed in Section A.2.1 and Section A.2.2 respectively.

A.2.1 Pricing Data

Figure A.1 shows the Pricing Data tab. To the left in the tab there are several control elements, which are described in Table A.1. Each control element in the table has a number corresponding to the red numbers in Figure A.1 for easy recognition. To the right in the tab there is a results view. The columns in this view are described in Table A.2.

The following main actions can be performed in this tab:

- Load product price data and display in the results view.
- Update or clear the product classification.
- Export product price data to Microsoft Excel.

Refer to Section A.3.1 for a complete description of the functions in this tab.

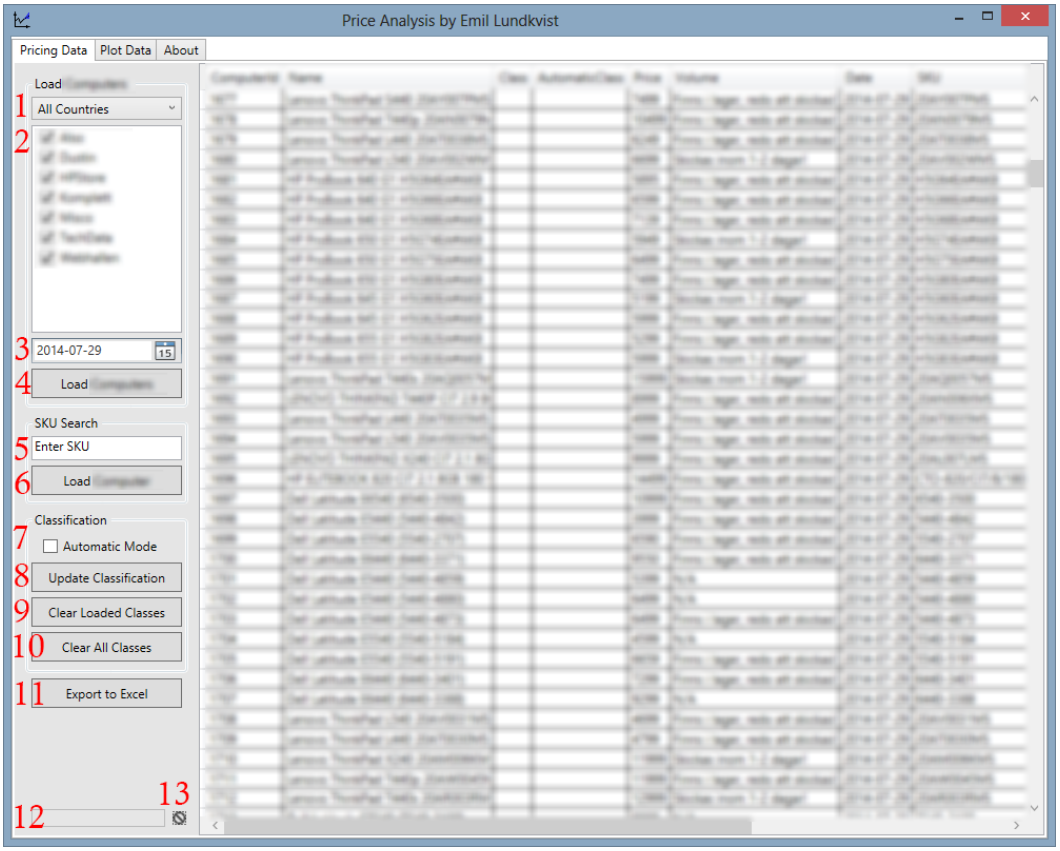

Figure A.1: Pricing Data tab.

Number	Name	Description
1	Country Combo Box.	A combo box for selecting which countries to load data from.
2	Vendor Check Boxes.	Check boxes for selecting which vendors to load data from.
3	Date Picker.	A date picker for selecting which date to load data from.
4	Load Products Button.	If this button is pressed, the products from the selected countries and vendors are loaded at the selected date.
5	SKU Text Box.	A text box for entering a SKU for a product to load.
6	Load Product Button.	If this button is pressed, the product with the SKU in the SKU Text Box is loaded for all countries and vendors where it exists.
7	Automatic Mode Check Box.	If this check box is selected when the Update Classification Button is pressed, the program will fill in the AutomaticClass column in the results view with classes.
8	Update Classification Button.	If this button is pressed, the classes manually set in the Class column in the results view will be saved to the database.
9	Clear Loaded Classes Button.	If this button is pressed, all the products loaded in the results view will have their classes cleared.
10	Clear All Classes Button.	If this button is pressed, all product in the database will have their classes cleared.
11	Export to Excel Button.	If this button is pressed, all data in the results view is exported to an Excel spreadsheet.
12	Progress Bar.	This progress bar shows when the program is busy and gives an indication of how long the action will take.
13	Cancel Action Button.	When this button is red it is possible to cancel the current action by pressing it. If it is grey, there is no action to cancel.

Table A.1: Description of control elements in the Pricing Data tab.

Column	**Description**
Attribute 1.	Description 1.
Attribute 2.	Description 2.
Attribute 3.	Description 3.
Attribute 4.	Description 4.
Attribute 5.	Description 5.
Attribute 6.	Description 6.
Attribute 7.	Description 7.
Attribute 8.	Description 8.
Attribute 9.	Description 9.
Attribute 10.	Description 10.

Table A.2: Column information in the results view.

A.2.2 Plot Data

Figure A.2 shows the Plot Data tab. To the left in the tab there are several control elements and to the right in the tab there is a graph view. The control elements are described in Table A.3 and information about how to use the graph view in Section A.2.2.

The following main actions can be performed in this tab:

- Plot product pricing information for SKUs or classes.
- Plot product pricing forecasts for classes.

Refer to Section A.3.2 for a complete description of the functions in this tab.

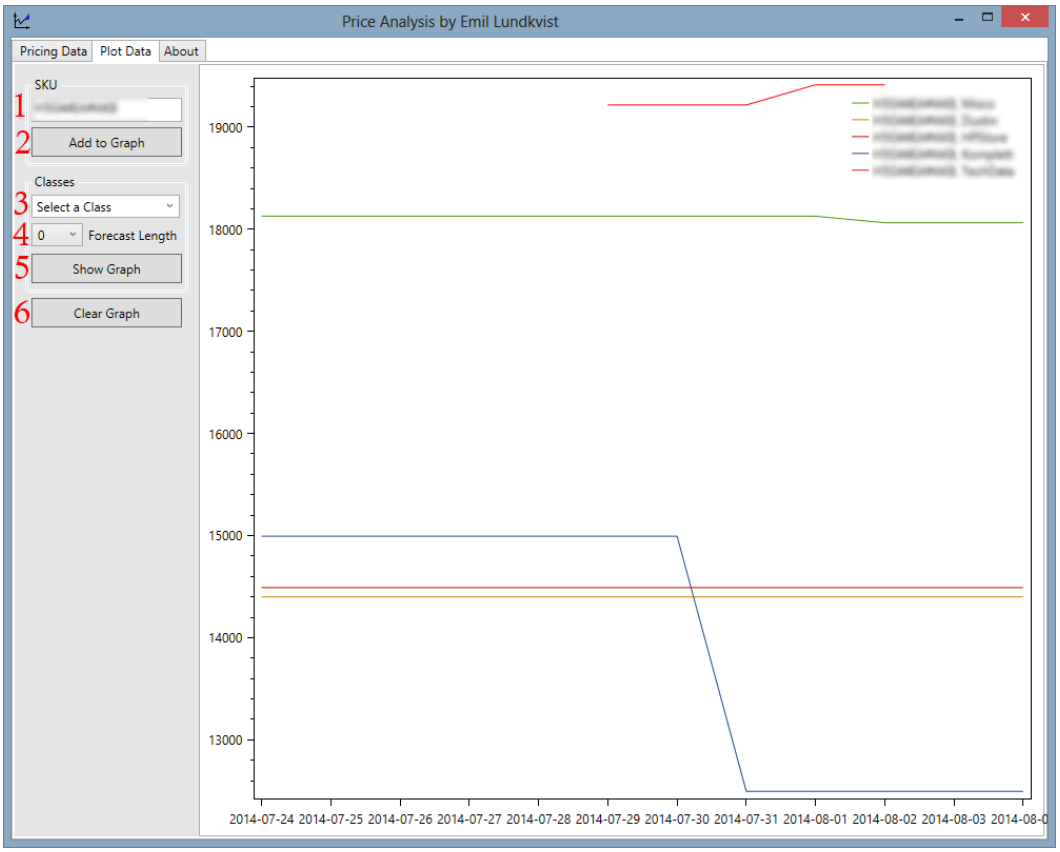

Figure A.2: Plot Data tab.

Number	Name	Description
1	SKU Text Box.	A text box for entering a SKU for a product to plot.
2	Add to Graph Button.	If this button is pressed, the product with the SKU in the SKU Text Box is plotted for all vendors where it exists.
3	Class Combo Box.	A combo box for selecting which class to plot pricing data from.
4	Forecast Length Combo Box.	A combo box for selecting the length of the forecast. If it is set to `0`, no forecast is performed.
5	Show Graph Button.	If this button is pressed, pricing data is plotted for the class selected in the Class Combo Box.
6	Clear Graph Button.	If this button is pressed, the graph view is completely cleared.

Table A.3: Description of control elements in the Plot Data tab.

Graph View

The graph view holds an xy-plot where the x-axis represents time and the y-axis represents price. Table A.4 holds information on how to navigate the view.

When product price information is plotted (refer to Section A.3.2), an info-box can be seen in the upper right corner. This info-box shows:

- The line color for the plotted product.
- The SKU for the plotted product.
- The vendor for the plotted product.
- A percentage for how important the plotted product is for the pricing in the predicted class. This number is only shown if a forecast has taken place.

Navigation	Mouse	Keyboard
Zoom	Scroll up to zoom in, scroll down to zoom out.	Press + to zoom in, − to zoom out.
Select Area	Hold down scroll wheel, select an area and release.	Not available.
Translate	Hold right mouse button and move the mouse.	Use the arrow keys.
Show Info	Hold left mouse button on a price line to get detailed info.	Not available.

Table A.4: Navigation in the graph view.

A.3 Functions

A.3.1 Pricing Data

Load All Products From Selected Vendors

1. Select "All Countries" from the Country Combo Box to load products from all countries, or select a specific country.
2. Select which vendors to load products among the Vendor Check Boxes.
3. Select which date to load pricing information from with the Date Packer.
4. Press the Load Products Button to display the products in the display view.

Load All Products With a Selected SKU

1. Enter the SKU for a specific product in the SKU Text Box.
2. Press the Load Product Button to display the latest prices for the product at each vendor where it exists in the database.

Classify Products

1. Load products into the data view by following the instructions in the previous sections.

2. Fill the Class column in the data view for the products that should be classified.
3. Press the Update Classification Button to save the classes to the database.

Automatic Mode If the Automatic Mode Checkbox is checked when pressing the Update Classification Button, the program will automatically fill the AutomaticClass column in the data view with class suggestions. These suggestions are calculated from the available information in the Class column. This means that a larger amount of products with a class will make the suggested classes more helpful.

Export Data to Excel

To export data to Excel, simply press the Export to Excel Button. It will open a new Excel spreadsheet with the information displayed in the data view. If there is a large amount of data in the data view, the action can take a couple of seconds to complete.

A.3.2 Plot Data

Plot Price for Products With a Selected SKU

1. Enter the SKU for a specific product in the SKU Text Box.
2. Press the Add to Graph Button to plot prices for the product at each vendor where it exists in the database.

Plot Price for Products Within a Selected Class

1. Select a class in the Class Combo Box.
2. Press the Show Graph Button to plot prices for the products in the selected class.

Show Price Forecast for Products Within a Selected Class

1. Select a class in the Class Combo Box.
2. Select a forecast length larger than zero in the Forecast Length Combo Box.
3. Press the Show Graph Button to plot prices and forecasts for the products in the selected class.

Clear the Graph

To clear the graph, press the Clear Graph Button.

Bibliography

[1] Accord.NET. *C4.5*. 2014. URL: https://github.com/accord-net/framework/blob/development/Sources/Accord.MachineLearning/DecisionTrees/Learning/C45Learning.cs.

[2] Accord.NET. *Framework*. 2014. URL: http://accord-framework.net/.

[3] H. Akaike. "A new look at the statistical model identification". In: *IEEE Transactions on Automatic Control* 19.6 (Dec. 6, 1974), pp. 716–723.

[4] E. Alpaydin. *Introduction to Machine Learning*. 2nd ed. The MIT Press, 2010.

[5] *Attivo Classification Engine*. 2014. URL: http://www.attivio.com/how-we-do-it/key-differentiators/classification-engine.

[6] M. J. Azur et al. "Multiple imputation by chained equations: what is it and how does it work?" In: *International journal of methods in psychiatric research* 20.1 (2011), pp. 40–49.

[7] L. Brailovskiy and Dr. M. Herman. "Prediction of Financial Time Series Using Hidden Markov Models". In: Stanford University, 2014.

[8] P. J. Brockwell and R. A. Davis. *Introduction to time series and forecasting*. Springer New York, 1996.

[9] P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods*. Springer New York, 1986.

[10] L. Cao and F. E. H. Tay. "Financial Forecasting Using Support Vector Machines". In: *Neural Comput and Applic* 10 (2001), pp. 184–192.

[11] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". In: *Commun. ACM* 13.6 (June 1970), pp. 377–387.

[12] J. Grandell. *Formulas and survey Time series analysis*. 2014. URL: http://www.math.kth.se/matstat/gru/sf2943/tsform.pdf.

[13] C. W. J. Granger. "Testing for causality : A personal viewpoint". In: *Journal of Economic Dynamics and Control* 2.1 (May 1980), pp. 329–352.

[14] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009.

[15] *Ipredict*. 2014. URL: http://www.ipredict.it/.

[16] S. B. Kotsiantis. "Supervised Machine Learning: A Review of Classification Techniques". In: *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technolo-*

gies. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2007, pp. 3–24.

[17] L. Ljung. *System identification - Theory for the User*. Prentice-Hall, 1999.

[18] H. Lütkepohl. *New Introduction to Multiple Time Series Analysis*. Springer Publishing Company, 2007.

[19] H. Madsen. *Time Series Analysis*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis, 2007.

[20] MathWorks. *Supervised Learning (Machine Learning) Workflow and Algorithms*. 2014. URL: http://www.mathworks.co.uk/help/stats/supervised-learning-machine-learning-workflow-and-algorithms.html.

[21] Microsoft. *Blend*. 2014. URL: http://msdn.microsoft.com/en-us/library/jj129478.aspx.

[22] Microsoft. *Serialization*. 2014. URL: http://msdn.microsoft.com/en-us/library/ms233843.aspx.

[23] Microsoft. *SQL Server*. 2014. URL: http://msdn.microsoft.com/en-us/library/bb545450.aspx.

[24] Microsoft. *TFS Server*. 2014. URL: http://msdn.microsoft.com/en-us/library/ms181238(v=vs.90).aspx.

[25] Microsoft. *Visual C#*. 2014. URL: http://msdn.microsoft.com/en-us/vstudio/hh341490.aspx.

[26] Microsoft. *Visual Studio*. 2014. URL: http://msdn.microsoft.com/en-us/vstudio/cc136611.aspx.

[27] Microsoft. *Windows Presentation Foundation*. 2014. URL: http://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx.

[28] J. Nielsen. *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1993.

[29] CERN press office. *CERN experiments observe particle consistent with long-sought Higgs boson*. 2012. URL: http://press.web.cern.ch/press-releases/2012/07/cern-experiments-observe-particle-consistent-long-sought-higgs-boson.

[30] *Oxyplot*. 2014. URL: http://www.oxyplot.org/.

[31] B. Pfaff. "VAR, SVAR and SVEC Models: Implementation Within R Package vars". In: *Journal of Statistical Software* 27.4 (2008).

[32] J. R. Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1993.

[33] J. R. Quinlan. "Induction of Decision Trees". In: *MACH. LEARN* 1 (1986), pp. 81–106.

[34] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2014. URL: http://www.R-project.org/.

[35] *R.NET*. 2014. URL: https://rdotnet.codeplex.com/.

[36] *Salford Systems Cart*. 2014. URL: http://www.salford-systems.com/products/cart.

[37] T. D. Schneider. *Information Theory Primer*. 2013. URL: http://schneider.ncifcrf.gov/paper/primer/primer.pdf.

[38] B. E. Sørensen. *Granger Causality*. 2005. URL: http://www.uh.edu/~bsorense/gra_caus.pdf.

[39] L. Stanca. *Multivariate time series models*. URL: http://dipeco.economia.unimib.it/Persone/Stanca/defap/DEFAPtopicsbeamerVAR_2011_nop.pdf.

[40] *Strategico*. 2014. URL: https://code.google.com/p/strategico/.

[41] *Zaitun Time Series*. 2014. URL: http://www.zaitunsoftware.com/.

[42] G. Zhang, B. E. Patuwo, and M. Y. Hu. "Forecasting with artificial neural networks: The state of the art". In: *International Journal of Forecasting* 14 (1998), pp. 35–62.