Technische Universität Hamburg-Harburg

An Investigation of Machine-Learning Approaches for a Technical Analysis of Financial Markets

Diploma Thesis

October 2010

submitted by:
Karsten Martiny

supervised by:
Prof. Dr. Ralf Möller
Prof. Dr. Kathrin Fischer

Technische Universität Hamburg-Harburg
Institute for Software Systems
Schwarzenbergstraße 95
21073 Hamburg

Statement of authorship

I hereby certify that this thesis has been composed by myself and describes my own work, unless otherwise acknowledged in the text. All references and verbatim extracts have been quoted, and all sources of information have been specifically acknowledged. It has not been accepted in any previous application for a degree.

Hamburg,
October 19, 2010
Karsten Martiny

Contents

1 Introduction

Technical analysis of financial markets is a discipline from the field of finance concerned with forecasting future price movements of securities through the study of historic market data. The basic principle of technical analysis is the observation that history tends to repeat itself. Over time, several subdisciplines have evolved which facilitate different approaches but share the same goal of identifying distinct situations in market history that result in similar successive developments, and exploit these insights to derive predictions of future price movements. Even though the area of technical analysis has been widely researched and a wealth of information is available on this topic, a successful application of these analysis methods towards a profitable trading strategy still requires profound experience in financial markets in order to correctly evaluate the peculiarities of any potential signal with respect to its informative value, reliability, and correlation with other signals.

In short, the task of gaining experience in the application of technical analysis can be seen as a process of learning characteristic patterns from historic market data. The task of automatically extracting patterns from data is in turn called data mining and has been subject to intensive research, as well. Therefore it seems auspicious to combine both areas of research with the goal of obtaining an adaptive technical analysis system. This system should be equipped with basic knowledge of technical analysis methods and then be tasked with learning of how to forecast future price movements through an analysis of historic price movements with data mining methods. The financial domain is especially well-suited to facilitate a machine-learning approach because all key properties are denoted with numerical values from the beginning. This allows for an accurate transformation into a machine-readable form and, most important, the numerical values provide precise performance criteria to quantify the system's utility.

Based on these considerations, this work has two major objectives: first, possible approaches of facilitating probabilistic reasoning processes for a variety of technical analysis methods are explored. These probabilistic reasoning modules are realized with Bayesian networks and aim at obtaining forecasts of future price movements. Second, it is investigated how the resulting forecasts can be exploited with reinforcement learning in order to reach a profitable trading strategy.

The subsequent work is structured as follows:

Chapter 2 gives an introduction to the field of technical analysis. A variety of common technical analysis methods is presented and their respective underlying assumptions, means of generating trading signals, and possible interpretations for the inference of forecasts are explained.

In chapter 3, the principles of Bayesian networks are explained. This chapter starts with a brief recapitulation of required properties from probability theory and introduces the semantics of Bayesian networks subsequently. Afterwards, several inference procedures are introduced that enable the use of Bayesian networks for probabilistic reasoning processes. The chapter concludes with the notion of dynamic Bayesian networks which augment the previously explained models with an incorporation of time as a further dimension in order to cope with temporal dependencies.

After the prerequisites have been introduced in the previous chapters, chapter 4 investi-

gates the feasibility of technical analysis methods for a use in machine learning processes. For all of the analysis methods that proved to be feasible, a machine-readable notion of the domain is introduced, approaches for an appropriate implementation of a probabilistic reasoning module are discussed, and the resulting forecasts are evaluated.

The basic concepts of reinforcement learning are introduced in chapter 5. It is shown how this technique can be used to learn a profitable trading strategy from the forecasts discussed in chapter 4 and the resulting gains are presented.

This work concludes with a summary of the results in chapter 6 and gives an outlook for possible subsesequent research topics in this area.

2 Technical Analysis of Financial Markets

In order to participate in any kind of financial markets a trader should perform some kind of market analysis prior to any investment decisions. The main goal of all analysis techniques is the evaluation of a security's current state as well as a forecast of future price movements.

The discipline of financial market analysis can be divided into two categories: the *Fundamental Analysis* studies key data of a security, e.g., financial statements, management decisions, market perspectives, and economic environments. The second approach is the *Technical Analysis* which does not take any fundamental data into account, but rather relies purely on a market's history (namely movements of price and volume, i.e., number of securities traded within a certain period of time) in order to predict future movements. While both approaches can justify a valid analysis result, there is a magnificent difference in the required input data, background knowledge, and evaluation techniques for each approach.

In order to accomplish a thorough fundamental analysis of any security, it is essential to have a sound knowledge of the corresponding market and the ability to assess the impact of a huge variety of information. Since just the mere task of computationally capturing news' meanings (let alone an evaluation of its possible impact) is highly error prone with current technology, major obstacles have to be overcome when facilitating a machine-based fundamental analysis.

The basis of all technical analysis approaches is the premise that *"market action discounts everything"* (see [Mur99], for example). This premise states that all relevant information regarding a certain security is already reflected in its current price. For instance, if a security's fundamental analysis suggests a prosperous development, the security's price should be rising and therefore the same conclusion can be reached with complete negligence of any fundamental data by simply examining the price and volume movements. The correctness of this core principal has been demonstrated in [Den08].

Informally speaking, technical analysis is all about forecasting future developments by identifying certain characteristics in a security's historical price and volume movements. Since this field has been subject to intensive research for more than a century, technical analysis has a set of well-established rules and correlations available that allow for a profound market analysis. Opposed to the fundamental analysis, most of the knowledge in the technical approach can be conveniently expressed in terms of equations and algorithms and therefore it can be easily adapted to a machine-readable form. Another benefit of the technical analysis is the universal validity of its knowledge and therefore no expert knowledge in certain markets is required. Consequently, this work aims to exploit the well-established knowledge of the technical analysis and does not take fundamental data into account at all.

This chapter gives an introduction to the key elements of the technical analysis as used within this work. For further information see for example [Mur99] or [Voi06].

2.1 Dow Theory

Cornerstone of all modern methods within the field of technical analysis is the work of Charles Dow from the late nineteenth century which is known as the *Dow Theory* today. Central element of this theory is the consideration of local extreme points that enable a classification of price movements into *trends*:

- A market is in an *uptrend* (also known as bullish trend[1]) if the price movement exhibits a series of successive higher highs and higher lows (see figure 2.1 for a visualization of this concept).
- Analogously, a market is in a *downtrend* (or bearish trend) if the price movement exhibits a series of successive lower highs and lower lows.
- If a market meets neither of the above criteria it is said to be *trendless* or in a sideways trend.

A trend is further divided into different phases: the phases while the prices move towards the trend's direction are called *trend movement* while phases in the opposite direction are called *trend correction*. A trend is started as soon as the movements exhibit three consecutive local extremes that meet one of the mentioned trend criteria. Accordingly, a trend is said to be broken once a local extreme occurs which violates the above criteria.

Note that a single security's price movement may exhibit several trends on different time scales at the same time: For instance, if a security is in a long-term uptrend, each phase will resemble a complete short term trend; each trend movement is a short uptrend by itself while each trend correction is a small downtrend.

Additionally, Dow stated that the trading volume of a security should resemble the corresponding price trend, i.e., during trend movements, the trading volume should be significantly higher compared to the volume during trend corrections. Turning points from one trend phase to the next mark important points of the movement. Therefore, they should be accompanied by a relatively large trading volume.

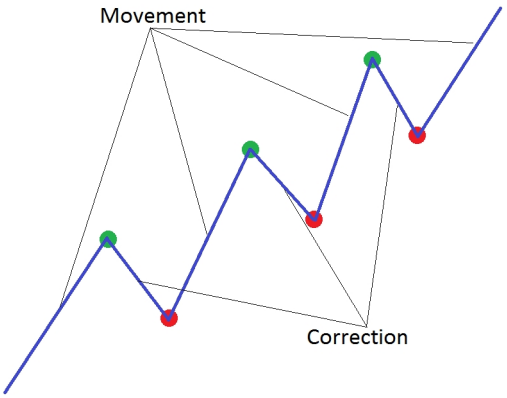


Figure 2.1: Schematics of an uptrend according to the Dow Theory: the local highs are marked with green dots, the local lows with red ones. This clearly points out the series of rising highs and rising lows, as required for the existence of an uptrend.

[1]The terms bull and bear are commonly used picturesque metaphors that resemble the fighting styles of animals: The bear tries to wrestle its opponent down to the ground while a bull tries to throw an opponent up in the air with its horns. Therefore, downward movements are said to be bearish while upward movements are said to be bullish.

This simple concept of trends is able to generate a variety of signals: For instance, if a security is currently in an uptrend and the current price just rose above the previous local high, the trend is confirmed and a buy signal is generated. Similarly, if the current price drops below the previous low, the trend is broken and a sell signal is generated.

For more information on the Dow Theory and possible trading strategies directly based on it, see [Voi06], for example.

While the definition of this concept is fairly straightforward, its application towards a successful trading strategy is all but simple, for human analysts as well as machines. Major obstacle in the transformation of the Dow Theory into a practical trading strategy is the distinction between arbitrary price movements that should be considered as noise and true local extremes. While one can easily determine local extremes in hindsight, identification of a security's current trend state relies heavily on instinct and guesswork. However, based on the Dow Theory, auxiliary methods have evolved that allow for a less ambiguous analysis, namely the analysis of characteristic price patterns.

2.2 Price Patterns

The movement of a security's price tends to exhibit easily identifiable patterns that reflect the current trend state. Since these patterns are easy to identify (and therefore easy to incorporate into a trading strategy), they are among the most popular methods in technical analysis. Price patterns can be coarsely divided into two categories: line patterns that provide continuation signals and point patterns that provide reversal signals.

2.2.1 Line Patterns

Trend Lines

All line patterns make use of the concept of *Trend Lines*. A trend line is an imaginary straight line that connects local extremes of the same type. A line that connects local lows is referred to as *support line*, while a line of local highs is called a *resistance line*. An example of trend lines is depicted in figure 2.2.

The rules for the identification of trend lines are simple:

- A trend line arises when more than two local extremes can be connected via a straight line. There may be other extremes of the same type in between that are not in the line's proximity, but the price movement must not cross the line significantly.
- A trend line is confirmed each time the price movements develop a new local extreme in the proximity of the line. A line's significance increases with its number of confirmations.
- A trend line is valid until it is crossed by the price movements (i.e., the security's price drops below a support line or rises above a resistance line). Once a line has been crossed it is said to be broken and will be ignored in the future.
  In practice, it is reasonable to introduce a second criterion to invalidate trend lines: the security's price may move away significantly from a trend line without ever crossing it. Without ever invalidating such a line, the set of existing trend lines may become extremely large, although most lines do not appear to have a significant relation to the current movements. Hence, a line should be artificially invalided if it has not received any confirmation within a certain time period.
  For instance, see the resistance lines in figure 2.2: line 3 is a long-lasting resistance

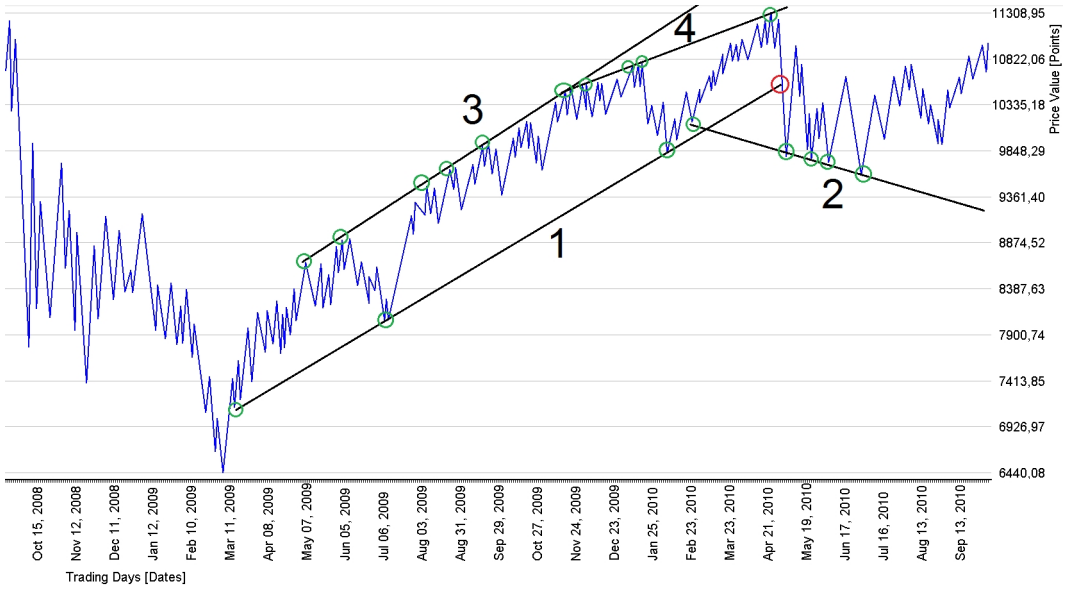


Figure 2.2: Recent price movements of the Dow Jones Industrial Average Index with exemplary trend lines. Lines 1 and 2 are examples of support lines, 3 and 4 resistance lines, respectively. Each confirmation of a line is marked with a green circle and the red circle depicts a line break. Note that many more lines can be found, but only few are depicted for clarity.

line that has been confirmed several times and remains unbroken in the depicted time frame. However, a new trend line (4) with a smaller slope evolved in November 2009. Apparently, this new line imposes stronger limits on potential upward moves and thereby the old line's significance fades.

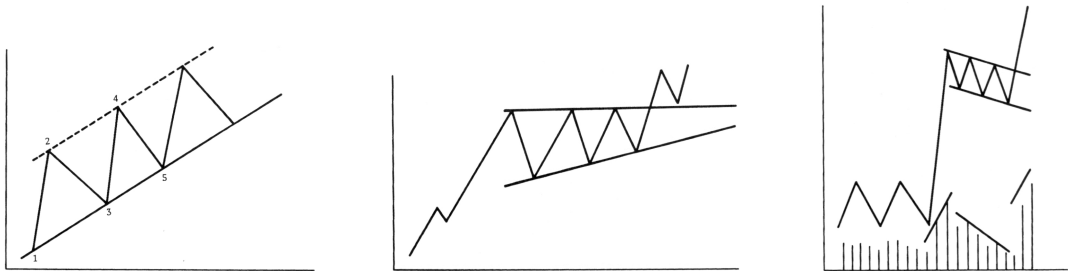
- The slope of a line gives information about the current trend state: in general, positive slopes reflect up movements and negative slopes reflect down movements. The slope's absolute value indicates the strength of the trend: the higher the value, the stronger the trend. Values close to zero can be considered as trendless states (independently of the slope's sign).

Note that the concept of trend lines directly incorporates the Dow Theory but imposes a severe restriction on the local extremes' positions. While according to the Dow Theory a sequence of extremes is only required to be monotonically increasing or decreasing in order to form a trend, the trend line concept additionally requires the extreme sequence to be subject to a linear function. However, once a trend line is identified, its interpretation is straightforward and not subject to ambiguities.

As it can be seen in figure 2.2, trend lines provide two kinds of valuable information: as long as a line remains valid and is confirmed by price movements frequently, it provides hints that future prices will continue to move in the line's direction. By breaking such a line, another valuable signal is generated, indicating that future movements are likely to change their direction. For instance, the break-through of the ascending resistance line 1 coincides with the development of the new descending resistance line 2 - obviously the upward move has lost a lot of its strength.

Combination of Trend Lines into Continuation Patterns

By relating one resistance and one support line per time interval, several characteristic patterns can be found. Since each pattern consists of two trend lines, a certain pattern



(a) Trend Channel (b) Triangle (c) Flag

Figure 2.3: Schematic line patterns (from [Mur99])

can be described by the following parameters:

- m_s, m_r: slopes of the support and resistance line, respectively
- d: duration of the pattern
- c: (optional) context

Key features of any pattern are the slopes of its trend lines and, most important, the difference between the slopes. This leads to two basic shapes of line patterns: if both slopes are approximately equal, the lines form a channel, whereas different slopes lead to a triangle. These basic shapes can be divided into several more patterns with different interpretations. To illustrate the usage of price patterns, figure 2.3 depicts some of the basic patterns:
(a) depicts a pattern of two long-lasting parallel lines. This is referred to as a trend channel and future prices are expected to continue moving within the boundaries of these lines.
(b) depicts the general shape of a triangle. If this pattern occurs, future prices are expected to move within a decreasing range until they arrive near the triangle's tip and yield a significant outburst subsequently.
(c) is called a flag and has the same shape as a trend channel. The difference is that this pattern only has a fairly short duration and is pointing opposite to the major direction of price movements. This is typically considered to be a temporary correction move that is followed by continuing movements into the previous direction.

For a detailed overview of different line patterns and their peculiarities see [Mur99].

2.2.2 Point Patterns

The second type of typical price patterns is defined by a set of typical points and indicates trend reversals. Again, these patterns are an extension of the Dow Theory. Mainly, there are two different point patterns:

The first is called *Head and Shoulders Pattern*. Key element of this pattern is a series of three consecutive local highs with the first and third high being on approximately the same price level (forming the shoulders) and the second high being on a significantly higher level (forming the head). The low points between head and shoulders are called neck. Figure 2.4 depicts an example of such a pattern. In accordance with the Dow Theory, this pattern indicates the end of an uptrend since the second shoulder was not able to rise above the head and thereby broke the current uptrend. After an occurrence of this pattern within an uptrend, a reversal is indicated and prices are expected to fall subsequently. Technical analysts consider this as one of the most reliable patterns. There is a counterpart to this pattern called *Inverse Head and Shoulders Pattern*. Similarly, this counterpart is defined

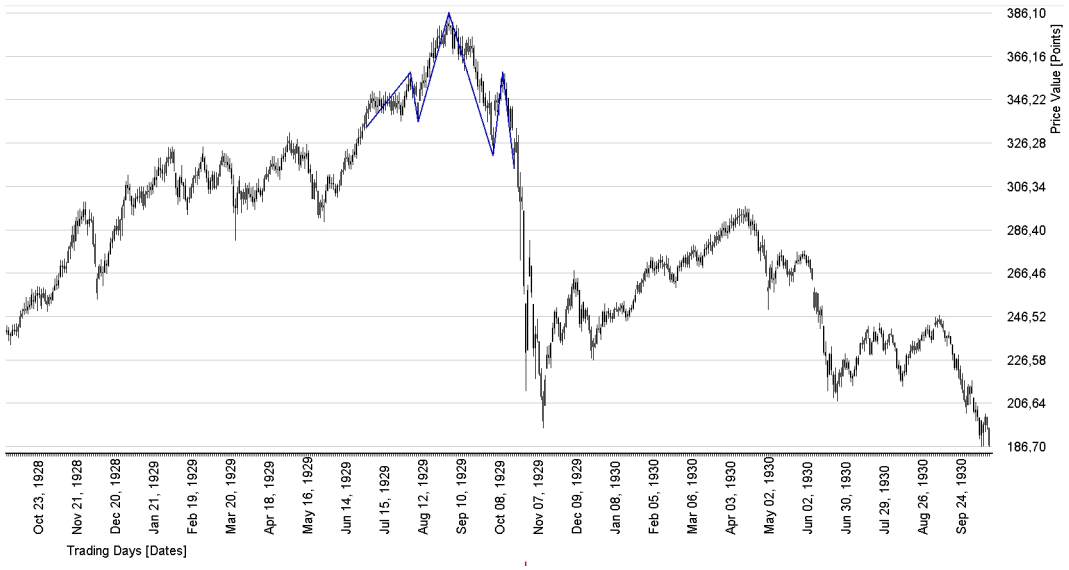


Figure 2.4: An example of the Head and Shoulders Pattern in the Dow Jones Industrial Average Index. This is probably the most infamous instance of the pattern, as it completed merely days before the Black Thursday (October 29, 1929) and therefore was preparatory to the Great Depression.

by a series of consecutive local lows with the center one being significantly lower than the other ones. Consequently, if this pattern occurs within a downtrend, upward movements are expected.

The other type of common point pattern is called *Double Top* and, as the name suggests, consists of two consecutive local highs on the same price level. If the same price value proved to be a barrier twice, it is assumed that this value forms a significant resistance that cannot be exceeded currently. Hence, if this pattern occurs within an uptrend, the trend is expected to stop with this pattern. This complies with the Dow Theory as the prices were not able to mark a new local high. This reflects only a missing confirmation of the trend while the previously described pattern clearly showed a trend break. Consequently, this pattern is not regarded to be as reliable as the Head and Shoulders Pattern. There also is an equivalent counterpart to this pattern: if two consecutive local lows on the same price level appear within a downtrend, the pattern is called *Double Bottom* and indicates the potential end of this trend. Examples of a Double Top and a Double Bottom are depicted in figure 2.5.

For more information on these patterns and possible variations see [Mur99].

2.3 Technical Indicators

Other popular instruments for the technical analysis of stock prices are *Technical Indicators*. Common to all these indicators is that they are defined by some formula that includes past values and are supposed to generate a certain signal. Since the resulting values are completely deterministic and therefore not subject to varying interpretations[2], they are quite popular with technical analysts as well as mechanical trading systems.

There exists a vast variety of different indicators with varying meaning and complexity.

[2]Only the indicator's value is deterministic of course, the value's interpretation is still subject to the trader's interpretation. However, most indicators suggest certain thresholds for signal generation.

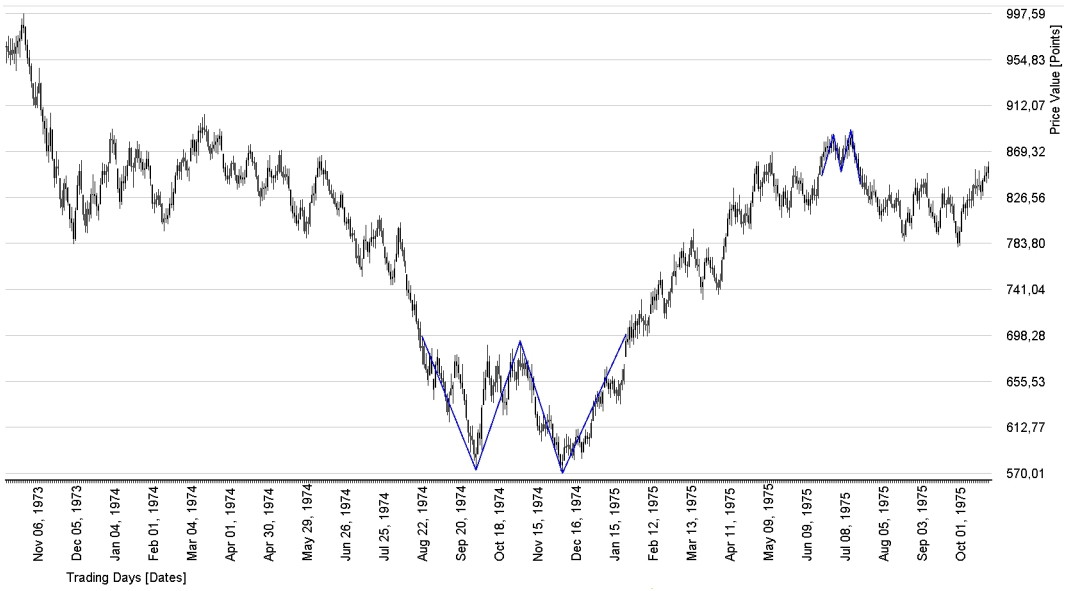


Figure 2.5: Examples of a Double Bottom (1) and a Double Top (2) in the Dow Jones Industrial Average Index. Note that the pattern's size hints to the magnitude of future price movements.

The following section provides a brief survey of the most commonly used indicators utilized in this work.

It should be noted that within this work the basic time unit is assumed to be one trading day. Depending on the desired trading frequency and the available data, one can also use other time units such as hours, weeks, or months.

2.3.1 Moving Averages

Moving averages are a very simple yet highly regarded type of a technical indicator. Basically, a moving average for a certain time t is calculated by averaging a certain number d of past values.

Variants of Moving Averages

Within the technical analysis, three different approaches are being used for calculation of an average:

- **Simple Moving Average (SMA)**
  This is the most simple way of constructing a moving average, its parameter d determines the "length" of the average. To determine the average's value at time t, d past price values P are simply summed up and divided by d:

$$SMA_d(t) = \frac{\sum_{i=1}^{d} P(t-i)}{d}$$

  While this approach is appealing because of its simple computation, one might argue that it lacks precision because each value used for its calculation contributes the same weight to the result and therefore older values influence the result by the same amount that newer values do.

- **Weighted Moving Average (WMA)**
  The construction of a weighted moving average is similar to the simple average, except that differentiated weights are used for its computation: starting with the newest value, each value is assigned a weight in a descending order. Therefore, a value's influence on the result fades with its age:

$$WMA_d(t) = \frac{\sum_{i=1}^{d}(d+1-i)\cdot P(t-i)}{d\cdot\sum_{i=1}^{d} i}$$

  Compared to the SMA, the WMA responds quicker to changes in the data, due to its differentiated weights. Thus it is tied closer to the actual movements.

- **Exponential Moving Average (EMA)**
  The exponential moving average (or, more precisely, the exponentially weighted moving average) assigns descending weights to older values as well. However, opposed to the WMA, all previous values are taken into the calculation with exponentially decreasing weights. Analogously to the previously introduced averages, the EMA takes a length-parameter d as an input. This parameter determines a constant smoothing factor α that represents the degree of weighting decrease and is used for the EMA's calculation:

$$\alpha = \frac{2}{d+1}$$

$$EMA_d(t) = EMA_d(t-1) + \alpha(P(t) - EMA_d(t-1))$$

  $EMA_t(1)$ is not defined and $EMA_t(2)$ may be initialized in different ways, most commonly it is set to $P(1)$. All subsequent values can be calculated by the above formula.

  Some analysts prefer this type of average, because it "takes the complete history into account". However, due to the exponential decrease, the influence of old values fades quickly to almost zero. Hence, this "advantage" is only of theoretical interest.

Parameter Determination

When using moving averages of any kind, one has to set a parameter for the average's length d. The lower d's value, the higher is the average's responsiveness. A fast response has the advantage of quickly reacting upon market changes, but at the same time this will yield to a rather noisy line. On the other hand, a large value of d leads to a smoother line and therefore fewer false signals, but it comes with the price of a considerable lag between market movements and an average's signal. Thus, there is always a trade-off between lag and signal quality which has to be taken into account for a determination of a feasible d. These considerations also hold for all of the following indicators that are duration-dependent.

Applications of Moving Averages

There are several concepts that exploit moving averages in order to infer certain information about a security's state. By examining the slope of an average one can easily determine the current direction of a long-term trend without the distortion of local noise. By comparing the relation between correct prices and an average one can determine the short-term trend state: if prices are quoted above the average, the share is in a short uptrend. Analogously, prices below the average identify a short-term downtrend. See figure 2.6a for a visualization of these concepts. More applications of moving averages are explained in the next section as those are treated as separate indicators.

2.3.2 Cross Average

By comparing two (or more) moving averages of different duration one can obtain a smoother method for signal generation. If the short-duration average (the "quicker" average) resides above the "slower" average, the share is considered to be in an uptrend, otherwise it is considered to be in a downtrend. Consequently, each time the average lines cross, the trend states are assumed to reverse and therefore a trading signal is generated.

It is common to plot the difference between two indicators separately (as shown in figure 2.6b). This makes the identification of trend states even easier: bullish and bearish trends are reflected by positive and negative values, respectively. Each crossing of the zero-line leads to the generation of a trading signal.

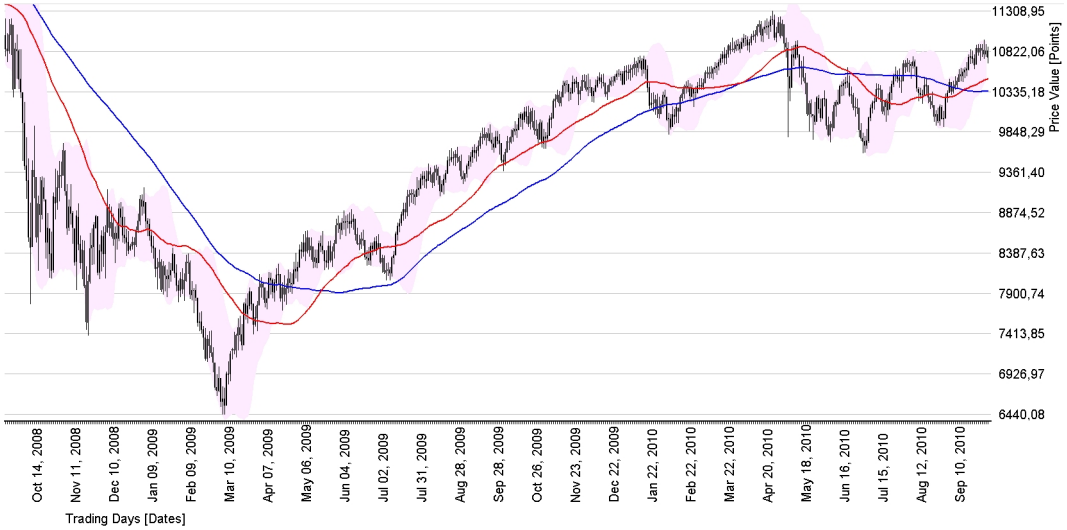
2.3.3 Moving Average Convergence/Divergence (MACD)

The *Moving Average Convergence/Divergence* is a widely used indicator based on moving averages with a more complex construction. It consists of two separate lines and is constructed as follows (described for example in [Mur99]).

1. Two separate exponentially smoothed averages are created. While in theory the averages' length can be adjusted as desired, practice established the standard of using a 12-day and a 26-day average.
2. The difference of these averages is calculated and plotted as the first line (called MACD line).
3. The resulting line is smoothed with another exponential average (usually of length 9) and plotted as the second line (called signal line).

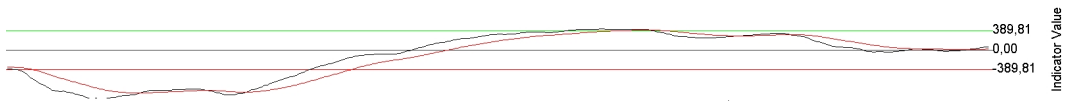
As an example for the MACD, see figure 2.6c.

There are several signals that the MACD can emit: the major trading signals are created by crossings of the two lines. If the signal line crosses the MACD line upwards, a buy signal is generated. Analogously, if the signal line crosses the MACD line downwards, a sell signal is generated. Additionally, the position of the MACD line is able to provide information by itself: large positive values reflect an overbought market (i.e., the market is expected to fall in the near future as a correctional response to previous strong upward movements) and large negative values reflect an oversold market. By combining these two facts, the signal's quality can be further increased. If a line crossing appears in a highly oversold or overbought area, the indicator emits two realizations of a forecast and thereby creates a more reliable signal.

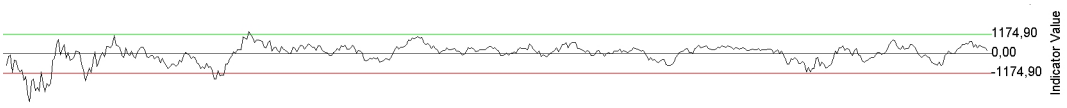


(a) SMA_{50} (red line), SMA_{100} (blue line), and Bollinger Band (shaded in rose, explained subsequently)

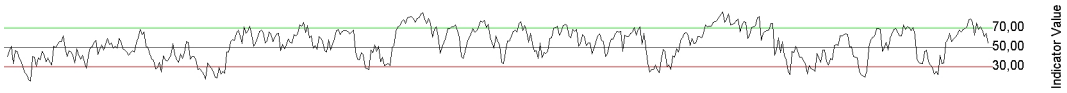
(b) 50/100-Day Cross Average



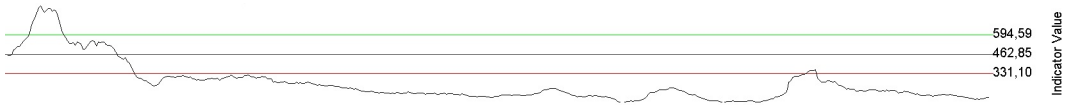
(c) Moving Average Convergence/Divergence with MACD line (black) and signal line(red)



(d) Momentum



(e) Relative Strength Index



(f) Average True Range

Figure 2.6: Examples of various technical indicators

2.3.4 Bollinger Band

The *Bollinger Band* is an analysis tool that measures the relative level of current prices. It consists of two pieces:

- A simple moving average of a certain length (SMA_d) forms the middle line
- A multiplicity (K) of the past d values' standard deviation (σ) determines the distance to the band's borders, i.e., the band is defined by the interval $[SMA\ -\ K\sigma,\ SMA\ +\ K\ \sigma]$. Commonly used values are $d = 20$ and $K = 2$.

An example of a Bollinger Band with the default values is shown in figure 2.6a.

The majority of price movements is assumed to happen within the Bollinger Band's range. Therefore two types of information can be extracted from this indicator.

The band width provides information about the price's volatility: a narrow band indicates small fluctuation amplitudes and vice versa. This fact, in combination with the band's direction, can be used to determine a trend's strength. A narrow band indicates a lucid movement into a certain direction while a spaced out band represents large oscillations in both directions. By comparing the current band width to the band width's historic average, one might try to predict the volatility's state in the near future. Since price oscillations mostly tend to happen within a certain range, situations in which the band width exhibits a magnificent deviation from its historic average are expected to quickly revert to this average.[3]

By analyzing the position of the current price value relative to the bollinger band, one can obtain an idea of the relative short-term price level. If the prices are close to the lower border, the security is assumed to be oversold and consequently prices are expected to bounce off the border, leading to a short-term upward movement. Analogously, prices located close to the upper border are expected to signal downward movements. Additionally, by taking the current bandwidth into consideration, one can also infer hints about the magnitude of expected movements.

2.3.5 Momentum

The *Momentum* is a simple indicator which measures speed and direction of the price motion (in the style of the physical quantity momentum, hence the name). It is calculated by subtracting the closing price d (usually 10) days ago from the current closing price:

$$M(t) = P(t) - P(t - d)$$

In other words, momentum measures the slope of price movements. If prices move steadily in one direction, the momentum indicator exhibits a horizontal line. This can be useful as an early warning sign of changing trends: in an uptrend, prices might still continue to climb, while the momentum starts to decline. This can indicate that the force of a (currently still valid) uptrend is fading. See figure 2.6d for an example.

Again, this indicator can be used to identify uptrends and downtrends through positive and negative indicator values, respectively. Just like with the previously explained indicators, large distances from the zero line indicate overbought and oversold situations and

[3] In financial mathematics, the concept of using the current state of a security's price in order to obtain information about future volatility is called *Implied Volatility*. While this concept is only used indirectly within this work, it plays a major role in other fields of finance, particularly in the pricing of options with the *Black-Scholes Formula*, as described in [BS73].

zero crossings indicate trend reversals.

One disadvantage of the momentum's simple concept is the fact that previous price movements can lead to sudden erratic fluctuations in the indicator line. If a strong rise or decline happened d days ago, the current indicator value will exhibit a significant outburst, even though current price changes may be rather small.

2.3.6 Relative Strength Index (RSI)

The Relative Strength Index[4] was originally introduced in [Wil78] and is also used to measure strength or weakness of price movements. In essence, the RSI is a refinement of the momentum's concept that aims to overcome the aforementioned disadvantage of a simple momentum indicator. The RSI is calculated as follows:

$$RSI = 100 - \frac{100}{1 + RS}$$

$$RS = \frac{\text{Average of closing prices within } d \text{ days with rising values}}{\text{Average of closing prices within } d \text{ days with falling values}}$$

De facto standard for the RSI's calculation is a duration of 14 days, as depicted in figure 2.6e.

Next to solving the aforementioned problem of erratic fluctuations, the RSI brings another advantage: opposed to the momentum, it has a constant vertical bandwidth of 100 points. Consequently, fixed values may be used as trigger marks for signal generation. RSI values above 70 points indicate overbought situations while values below 30 points indicate oversold situations (marked in the chart with a green and red line, respectively).

If the RSI resides in the overbought or oversold range, divergences between RSI's curve and actual price motions may emit early warnings of trend changes. According to [Wil78], these divergences are in fact the most valuable information emitted by the RSI.

2.3.7 Average True Range (ATR)

Another indicator introduced by [Wil78] is the *Average True Range* which measures a price trend's degree of volatility.

The term *range* refers to the maximum motion within a given time interval. Thus, a trading day's range is simply the difference between the day's high and low price. While the range gives a first impression about a stock's volatility, its concept is too simple to truly grasp the volatility state since it only indicates intra-day movements but is not concerned with inter-day motions. For instance, stock quotes on a certain day may open significantly above the previous day's close (thereby leading to a so-called *gap*) without exhibiting significant movements throughout the day. The range in this situation would be rather small while the actual price volatility is considerably higher.

To overcome this problem, the *True Range* takes the previous day's key points into account and is defined as the maximum of the following three values:

- Difference between today's high and today's low
- Difference between yesterday's close and today's high

[4] The name is somewhat misleading because in technical analysis, the term *relative strength* usually refers to the ratio of a certain stock price to the market average. However, this is not what is measured by the RSI.

- Difference between yesterday's close and today's low

Hence, the true range's formula is

$$TR(t) = \max(\text{High}(t), \text{Close}(t-1)) - \min(\text{Low}(t), \text{Close}(t-1))$$

Consequently, the true range provides valuable hints at the volatility state.

The *Average True Range* is a d-day exponential moving average of true range values. In [Wil78] it is recommended to employ a 14-day average, as shown in figure 2.6f.

The average true range characterizes the magnitude (and thereby the "force") of a given trend state. Note that it is independent of the motion's direction, which makes it impossible to infer any predictions concerning a movement's direction solely based on this indicator. Therefore, it is rather a supplement to other indicators which possibly allows for an increased quality in the inferred predictions.

2.3.8 Aroon Indicator

The *Aroon*[5] *Indicator* is a two-line indicator system originally proposed in [Cha95]. It allows for an identification of trends and their respective strength. As depicted in figure 2.7b, it is created upon two indicators called *Aroon Up* and *Aroon Down* that measure strength of up and downtrends, respectively.

The Aroon Up and Aroon Down indicators for a certain duration d (usually 14 days) are calculated as follows:

$$AU = 100 \cdot \frac{d - (\text{Number of days since highest high in previous } d \text{ days})}{d}$$

$$AD = 100 \cdot \frac{d - (\text{Number of days since lowest low in previous } d \text{ days})}{d}$$

The indicator lines signal a new extreme within the given period with a value of 100 (the indicator's maximum value). Conversely, a value of 0 indicates that price movements were not able to reach a new local extreme within the given time period.

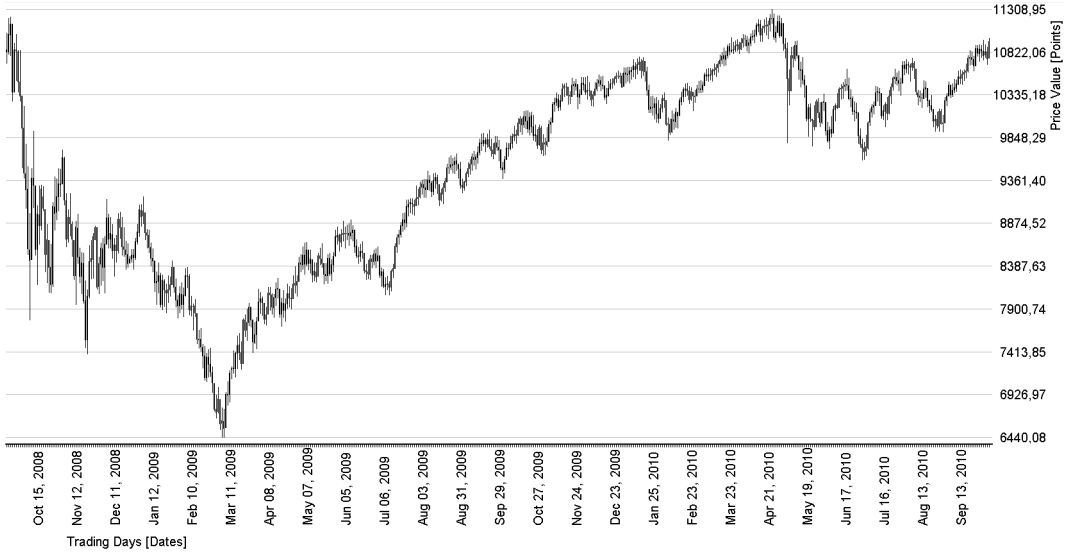
This leads to a rather straightforward interpretation of the indicator lines. If the Aroon Up resides at its upper limit, a strong upwards move is indicated. If the Aroon Down is at its upper limit, a strong downward move is indicated. Trends are indicated if a signal line remains persistently within the range above 70. Ideally, the trend's state is confirmed via the opposing indicator residing in the range below 30 (these thresholds are depicted in figure 2.7b via the green and red horizontal line, respectively).

A crossover of the indicator lines should receive special attention since this will most likely indicate the end of a previous trend.

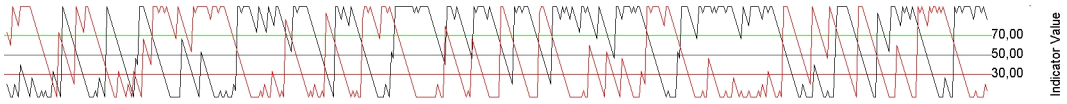
2.3.9 Commodity Channel Index (CCI)

The *Commodity Channel Index* is an indicator introduced by Donald Lambert in an article of the October 1980 issue of Commodities magazine. A reprint of this article can be found in [Lam83]. Although this indicator was originally intended to be used in commodity

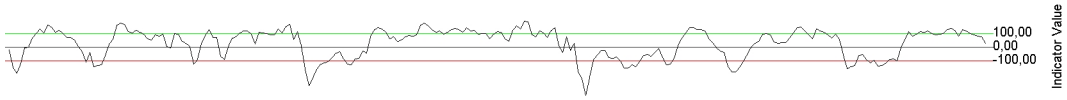
[5] "Aroon" is a Sanskrit word meaning "dawn's early light" or the change from night to day, as explained in [Cha95].



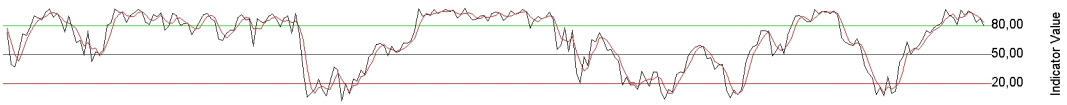
(a) Price movements (for reference purposes)



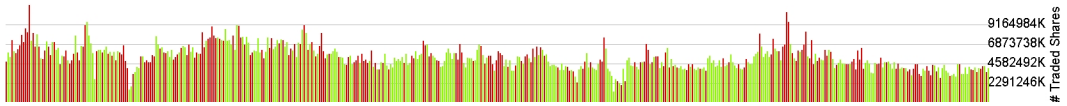
(b) Aroon Indicator, consisting of Aroon Up (black line) and Aroon Down (red line)



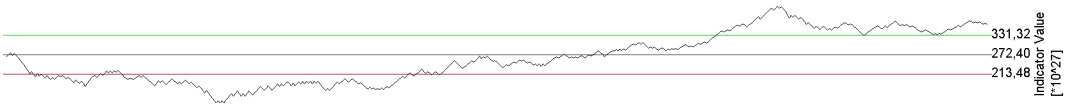
(c) Commodity Channel Index



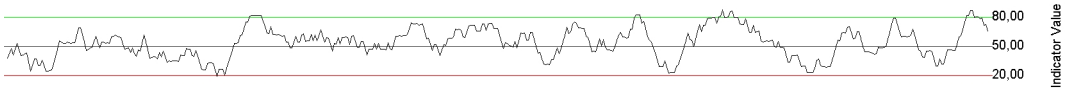
(d) Stochastic %K (black line) %D (red line)



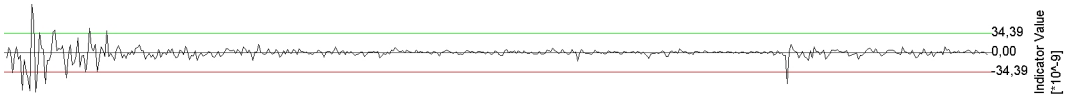
(e) Trading Volume



(f) On-Balance Volume



(g) Money Flow Index



(h) Ease of Movement

Figure 2.7: Further examples of various technical indicators

markets, over time it became popular with technical analysts in various markets.

Opposed to previously introduced indicators, the CCI's calculation is based on the day's *Typical Price*:

$$TP(t) = \frac{\text{High}(t) + \text{Low}(t) + \text{Close}(t)}{3}$$

The CCI is calculated by dividing the difference of the typical price and its simple moving average by the typical price's mean absolute deviation:

$$CCI = \frac{1}{0.015} \frac{TP - SMA}{\sigma(TP)}$$

According to [Lam83], the additional scaling factor of $\frac{1}{0.015}$ is used to ensure that the majority of values falls between -100 and 100.

Lambert originally aimed for the identification of strong trends. He argued that values above 100 and below -100 signal strong uptrends and downtrends respectively. He also suggested to use those values as trigger marks for signal generation: a buy signal is generating via the CCI's upward breach of 100 and a sell signal via a downward breach of -100.

While the indicator itself is rather popular among traders, its common interpretation has changed: instead of using the trigger marks for an anticipation of beginning trends, popular interpretation is exactly the opposite. Similar to some of the previously described indicators (e.g., the RSI), large values are believed to signal overbought and oversold situations. Hence, a crossing of one of the trigger marks leads to the expectation of an imminent correction.

An example of the CCI is depicted in figure 2.7c, the high and low signal triggers are marked by a green and a red line, respectively.

2.3.10 Stochastics Oscillator (%K%D)

The *Stochastics Oscillator* is a two-line indicator system that attempts to predict turning points by comparing a security's current closing price to its recent price range. It is based on the observation that closing prices in an uptrend tend to reside near the range's upper limit while closing prices in downtrends happen to appear near the lower border. Thus, the indicator's first line (called %K) denotes the current price level relative to its recent trading range:

$$\%\ K(t) = 100 \cdot \frac{P(t) - L_d(t)}{H_d(t) - L_d(t)}$$

As usual, P denotes the closing price. L_d and H_d refer to the highest high and lowest low within the last d (usually 14) days.

The second line (called %D) is a 3-period exponential moving average of the %K-line:

$$\%D = EMA_3(\%K)$$

This results in two lines oscillating between 0 and 100 (see figure 2.7d for an example). The indicator's extreme areas that signal overbought and oversold situations are above 80

and below 20 (marked by the green and red line in the figure).

Conditions for the stochastic's signal generation are somewhat more complex than with other indicators: in general, this indicator emits trading signals if divergences between the indicator and the corresponding price movements occur. A negative divergence occurs if the %D-line resides above 80 and exhibits two peaks with a descending tendency, while the price continues to move upwards. This indicates that, although prices still proceed towards new highs, the motion loses its momentum and therefore an imminent turning point with a subsequent downward move is expected. If these prerequisites are met, the actual signal is generated by a crossover of %K and %D. Conversely, a buy signal is generated if the %D-line resides below 20 and exhibits two peaks with an ascending tendency, while price movements still continue to move downward.

To summarize, the stochastic indicator generates a signal if all of the following conditions are met:

- the %D-line is in an extreme area (> 80 or < 20)
- the %D-line forms two peaks with a certain tendency
- price movements diverge from the %D-line tendency
- the %D-line is crossed by the %K-line

2.3.11 Trading Volume

As already briefly mentioned in section 2.1, while analyzing price movements it is common to take the trading volume into account, also. According to the theory of technical analysis, the trading volume (i.e., number of shares traded within a certain time period) can be considered as a measurement for any signal's strength. Since the volume itself can rarely provide feasible information about future price movements by itself, it is referred to as a *secondary indicator* which may be used to enhance the quality of other indicator's signals.

Usually, the trading volume is displayed as a bar chart below the price chart (as depicted in figure 2.7e). For a better distinction of the price movement's direction it is common to color the volume bars of rising days in green and falling days in red.

The reasoning behind analyzing the volume is that price movements arise from the actions of market participants. Consequently, the trading volume can be considered as a majority vote among market participants: for instance, if an upward movement is accompanied by a high trading volume, it can be seen as a hint to the fact that a large number of traders believes in rising prices. Consequently they would be expected to continue buying a particular share and thereby driving its price further upwards.

Also, many market participants use a low-frequency trading strategy, i.e., they do not convey market orders very often. They usually only step into action if prices arrive at a level that they regard as relevant enough to alter their positions. Hence, such levels are accompanied by a high trading volume which stresses their importance.

To further illustrate the volume's impact, it is useful to examine an example for low-volume trading days. A typical situation for the occurrence of very low-volume trading days is the holiday season at the end of December (this can be clearly seen in figure 2.7e, as the volume exhibits significant lows during this time). It is obvious that only very few people work (and therefore participate in financial markets) during this time. Consequently, it only takes relatively few market participants to push prices into a certain direction. However, once the other market participants return in the beginning of the following year, they are able to quickly correct any movements that do not reflect the majority's opinion due to the significantly higher volume.

2.3.12 On-Balance Volume (OBV)

A simple indicator based upon the concept of trading volume is the *On-Balance Volume*. It simply cumulates the trading volume with regard to the price movement's direction (initialized with $OBV(0) = 0$):

$$OBV(t) = OBV(t-1) + \begin{cases} \text{volume}(t), & \text{if } P(t) > P(t-1) \\ 0, & \text{if } P(t) = P(t-1) \\ \text{-volume}(t), & \text{if } P(t) < P(t-1) \end{cases}$$

As depicted in figure 2.7f, this indicator usually moves into the same direction as price movements do. While the indicator's actual values are not of much interest, its direction may provide valuable information. Since it usually reflects the price movements, any occurring divergences between OBV and prices may hint to imminent turning points. As explained in [Mur99], it is common believe that "volume prevails prices", i.e., a change of trend should be signaled early through the OBV before any signs become visible in the price movements.

A careful comparison of the OBV in figure 2.7f with the corresponding price movements in figure 2.7a already forecloses subsequent results of this work: the OBV's development *exactly* coincides with the price movements, i.e., local extremes in the OBV curve appear at the same positions as they do in the price movements. Without any prevailing changes in the OBV, it can hardly extend the information derived directly from an analysis of the prices.

2.3.13 Money Flow Index (MFI)

The *Money Flow Index* is an indicator that relates price and volume movements. Its construction starts with the calculation of the so-called *Money Flow* based upon a security's typical price TP (see section 2.3.9) and volume:

$$MF = TP \cdot \text{volume}$$

Subsequently, the *Money Ratio* is calculated for the previous d (typically 14) days. It is the ratio of cumulated positive and negative money flows during that time period:

$$MR = \frac{\text{cumulated positive money flow for } d \text{ days}}{\text{cumulated negative money flow for } d \text{ days}}$$

Finally, based upon the money ratio, the MFI is calculated as follows (see figure 2.7g for an example of the MFI):

$$MFI = 100 - \frac{100}{1 + MR}$$

As with many of the previously described indicators, the MFI is used to signal overbought and oversold situations through trigger marks at 80 and 20, respectively. Also, any divergences in the trend directions of MFI and price movements are considered to be early signals of a forming trend reversal.

2.3.14 Ease of Movement (EOM)

The *Ease of Movement* is another indicator which relates price and volume and can be seen as a sensible seismograph of supply and demand. It is calculated as follows (as before, H and L refer to a day's high and low values):

$$\begin{aligned}
&\text{Midpoint:} && MP(t) && = \frac{(H(t) + L(t))}{2} \\
&\text{Midpoint Move:} && MPM(t) && = MP(t) - MP(t-1) \\
&\text{Box Ratio:} && BR(t) && = \frac{\text{volume}(t)}{(H(t) - L(t))} \\
&\text{Ease of Movement:} && EOM(t) && = \frac{MPM(t)}{BR(t)}
\end{aligned}$$

As depicted in figure 2.7h, the indicator oscillates around the zero line. An EOM value near zero represents a rather balanced relation between supply and demand. It is commonly assumed that any trend reversals are accompanied by significant shifts in the relation of supply and demand. Therefore, this indicator can be used as a filter: if other indicators point to a pending trend reversal, the EOM should exhibit fluctuations of a high magnitude. If instead the EOM's graph remains fairly steady, the signal may be regarded as being likely erroneous.

2.3.15 Summary

In general, the indicators presented previously can be classified into four categories:

- trend direction indicators
- trend strength indicators
- volatility indicators
- secondary indicators

Table 2.1 lists a summary of each indicator's key features.[6] A comparison of the indicators shows that there are plenty of them with high similarities as they use the same input and aim to yield the same results through similar formulas. Thereby, one should expect to obtain an analysis of similar quality by utilizing only a small subset of these indicators. Hence, at first glance it might appear to introduce an unnecessary increase in complexity to use a great variety of indicators. However, the usage of several similar indicators serves two purposes: first, if several indicators point to the same fact, they confirm each other and thereby lead to a more reliable result. Second, and more important, there is always the possibility that different indicators diverge in their signals. These divergences can in fact provide highly valuable information (for instance, they can be used to filter out false signals or to refine the anticipated magnitude of any forecast). Of course, due to the many similarities, the employment of further indicators can only improve the result to a very limited extend.

[6] As explained above, many indicators may serve different purposes at the same time. Hence the listed indicator types are not exclusive but rather reflect the main usage.

Indicator	Type	Range
Moving Average	trend direction	positive unlimited
Cross Average	trend direction	unlimited
MACD	trend direction and strength	unlimited
Bollinger Band	trend direction and volatility	positive unlimited
Momentum	trend direction and strength	unlimited
RSI	trend strength	[0,100]
ATR	volatility	positive unlimited
Aroon	trend direction and strength	[0,100]
CCI	trend strength	unlimited
Stochastics	trend strength	[0,100]
Volume	secondary	positive unlimited
OBV	trend direction and strength	unlimited
MFI	trend strength	[0,100]
EOM	secondary	unlimited

Table 2.1: Key features of the technical indicators

2.4 Candlestick Analysis

The third method of analyzing securities used within this work is the technique of *Candlestick Charts* and the recognition and interpretation of distinguished patterns within these charts.

2.4.1 About Candlestick Charts

Candlesticks charts are a visually appealing method[7] of presenting price movements that has been developed in Japan centuries ago. In fact, the use of candlesticks amongst ancient Japanese rice traders dates back to the early 18th century and thereby, this method is established significantly longer than modern financial markets themselves. However, this technique remained unknown to the western world until the early 1990s, when it was first made accessible in [Nis91]. It has gained popularity amongst western analysts ever since.

To create a candle chart, the key points of every day are used to construct a candle. A day's key points consist of its opening, highest, lowest, and closing price. (These data sets are sometimes also referred to as *OHLC-data* and will form the actual machine input subsequently).

As depicted in figure 2.8, the construction of a candle works as follows: a rectangle is drawn between opening and closing price (called the candle's body), a line is drawn from the body's upper edge to the highest value (called upper shadow or upper wick) and, accordingly, another line is drawn from the body's lower edge to the lowest value (called lower shadow or lower wick). Additionally, the candle's body will be colored according to the situation: "rising days" (i.e., the closing price is above the opening price) are marked with a green color, while "falling days" are marked red. Alternatively, it is also common (namely in print media) to color the candle bodies in white and black instead of green and red.

Note that a candle does not necessarily have to exhibit all of these features: since the

[7] Of course, the concept of a visually appealing chart does not provide any use for a computerized processing. However, since the analysis of the corresponding patterns is closely related to this form of depiction, the candlestick chart is still explained briefly here.

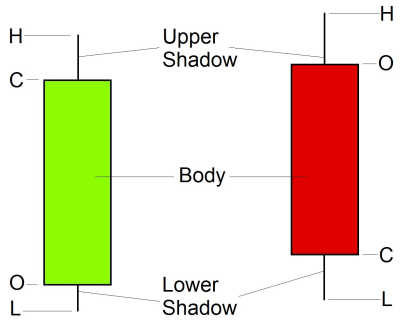


Figure 2.8: Depiction of time intervals as candles: O = opening price, H = highest price, L = lowest price, C = closing price; if the closing price is above the opening price, the candle's body is colored green, otherwise it is colored red.

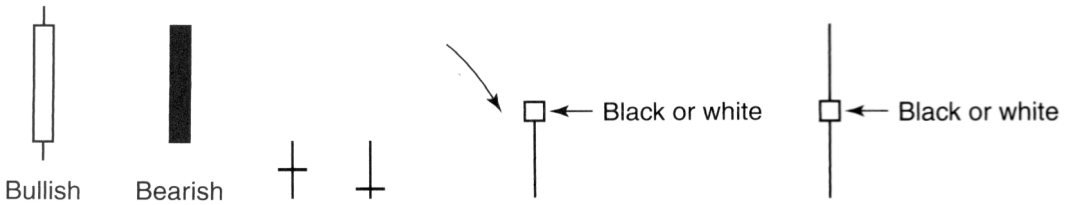
opening or closing values may coincide with a day's high or low values, there can be candles with only one or even no shadow at all.

2.4.2 Characteristic Candlestick Shapes and their Meaning

The candlestick's main advantage is that it allows for an instant perception of the market participants' attitude. Figure 2.9 depicts a few characteristic candlestick shapes which, according to [Nis03], can be interpreted as follows:

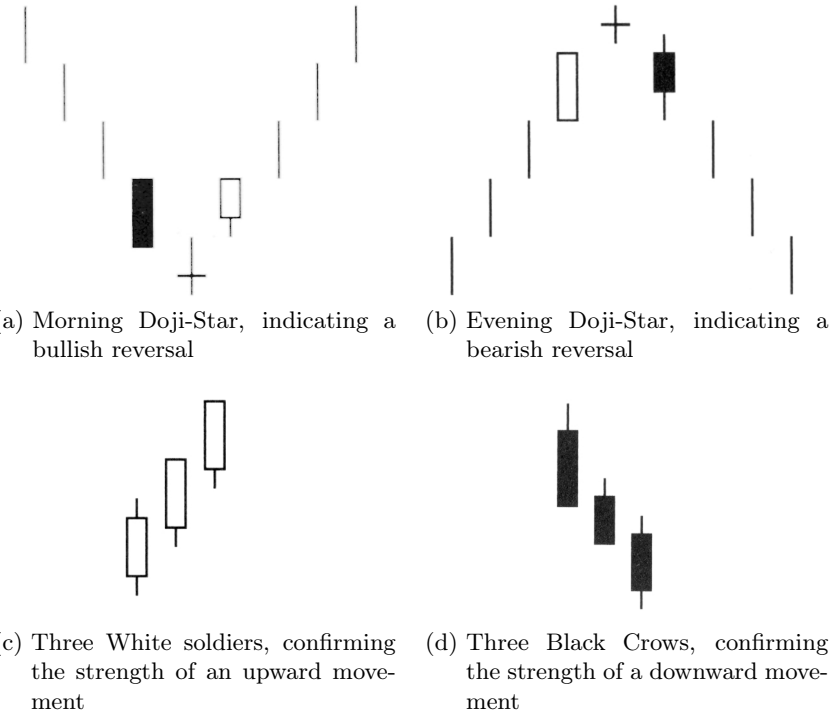
- (a) Candles with large bodies and relatively small shadows are called *Belt-holds*. Such a candle shows that opening and closing prices are near the day's extreme prices and thus a strong and lucid movement into one direction (depending on the body's color) has occurred. This usually shows that market participants are clearly in favor of one definite direction.
- (b) A session in which the opening and closing prices are roughly on the same level (indicated by a minuscule body) is called *Doji*. As there is no significant difference between the session's opening and closing, market participants are obviously undetermined about the direction. Hence, this candle is associated with uncertainty.
- (c) If a candle with only a large lower shadow and a rather small body appears within an ongoing downward movement, this candle is called a *Hammer*. As opening and closing prices do not differ in large, the body's color is of no importance for this candle, but it is rather distinguished by its characteristic shape. As this candle is prevailed by a downward movement, it appears that this movement continued in the session's beginning but came to a halt sometime during the day. Since prices were able to recover their losses until the session's end, this candle may signal that the previous downward movement has ended and prices have arrived at a turning point.
- (d) A candle with a diminutive body and unusually large shadows in both directions is called a *High Wave Candle*. Again, this candle resembles indetermination as it does not exhibit a clear direction. However, the large shadows resemble a rather high volatility and therefore significant forces into both directions. This may constitute an early warning sign that the prevailing movement's direction is about to revert.

These are only a few examples to provide some insight into the possibilities of candlestick interpretations. More detailed information about characteristic candlesticks and their interpretation can be found for example in [Nis03].



(a) Belt-holds (b) Doji (c) Hammer (d) High Wave Candle

Figure 2.9: Examples of characteristic candlesticks (from [Nis03])



(a) Morning Doji-Star, indicating a bullish reversal

(b) Evening Doji-Star, indicating a bearish reversal

(c) Three White soldiers, confirming the strength of an upward movement

(d) Three Black Crows, confirming the strength of a downward movement

Figure 2.10: Examples of candlestick patterns (from [Nis03])

2.4.3 Candlestick Patterns

While candlesticks are able to provide valuable insight into a market's situation, single candlesticks are usually regarded as too fragile to allow for a prognosis of required reliability. Hence, instead of relying only on a single candle's shape, forecasts are based upon constellations of successive candlesticks. These so-called *Candlestick Patterns* usually consist of a series of three candlesticks with certain properties. Next to the candle's shapes, their positions relative to each other, as well as the prevailing direction of movement, are taken into consideration.

To give an impression of possible candlestick patterns, figure 2.10 illustrates a few examples.[8]

According to the previous explanations, the first candle in (a) indicates a solid downward movement, the second candle indicates insecurity (and thereby already demonstrates the weakness of the prevailing movement), and finally, the third candle resembles a rather

[8] As shown in the figure, it has become a tradition to associate a rather fantastic name with each particular pattern. However, the patterns' names are of no importance whatsoever within this work.

strong upward movement, thereby confirming that the second candle's insecurity indeed lead to a reversal of the previous movement. This represents the character of most patterns: the first two candles indicate a possibility of future movements and by confirming this possibility, the third candle rounds off the pattern. Consequently, this candle sequence forms an upward-pointing reversal pattern. As with practically all candle patterns, mirroring this pattern leads to its equivalent counterpart (depicted in (b),) which indicates a downward-pointing reversal.

As an example for a continuation pattern see (c) (and (d) for its corresponding counterpart). All the candles within this pattern resemble strong movements into one direction and thereby this pattern indicates the continuation of an existing movement.

For a thorough guide to candlestick patterns see [Nis91] or [Mor06].

In general, candlestick patterns can be classified into two categories: reversal patterns indicate an imminent turning of the current movement's direction while continuation patterns confirm the current movement. Opposed to the previously introduced analysis methods, candlestick patterns only require a succinct time period to form a characteristic pattern and thereby emit a signal. Consequently, it is only natural to utilize these patterns for short-term forecasts. In fact, distinguished patterns have the means to forecast the following day's direction with a rather high certainty, but by extending the forecast further into the future, its quality will quickly dimmish. [Mor06] analyzed statistical correlations between a forecast's length and its quality and reached the conclusion that a feasible hit ratio can be achieved for a maximum of three days, while after a maximum of seven days a pattern's prognosis is hardly able to outperform a random guess.

Due to their short-term nature, candlestick patterns may be employed to facilitate strategies with a rather high trading frequency. Also, since the previously described analyzing methods tend to forecast movements on a larger scale, they can be augmented with candlestick patterns to pinpoint the exact positions of turning points.

3 Preliminaries on Bayesian Networks

This chapter introduces the concept of Bayesian networks as an approach to implement probabilistic reasoning procedures. After providing a recapitulation of required properties from probability theory, the semantics of Bayesian networks and possible inference procedures are introduced.

3.1 A Brief Summary on Probability Theory

Probability theory is the branch of mathematics concerned with the representation of uncertain knowledge. Within the field of artificial intelligence, probabilities can be used to express degrees of belief, i.e., the likelihood that a certain event is true. This section is an abstract of the introduction to probability theory from [RN02].

Basic element of probability theory is the *random variable*, commonly denoted by a capital letter. Each random variable has a domain of possible values. Therefore random variables describe the state space of propositions. Single elements of the state space are usually denoted by lowercase letters. The probability of a certain event (i.e., a particular value assignment x to a random variable X) is denoted by $P(X = x)$ or, if there is no possibility for ambiguities, it can be simply denoted by $P(x)$.

Random variables can be divided into two types depending on their respective domain:

- **Discrete random variables** take on values from a domain of countable elements. The values in the domain must be mutually exclusive and exhaustive.
- **Continuous random variables** take on values from the set of real numbers. The domain can be either the complete set or a specified subset.

3.1.1 Probability Axioms

The semantics for probability statements are defined by three basic probability axioms :

1. The probability for any event x is a real number between 0 and 1:
   $0 \leq P(x) \leq 1$
2. Assumption of unit measure: the probability that some elementary element from the entire sample space X will occur is 1. Consequently, there are no elements outside the sample space:
   $P(X) = 1$ and $P(\emptyset) = 0$
   In other words, events that are necessarily true have a probability of one and events that are necessarily false have a probability of zero:
   $P(\text{true}) = 1$ and $P(\text{false}) = 0$
3. The probability of a disjunction of two events x and y is given by:
   $P(x \vee y) = P(x) + P(y) - P(x \wedge y)$

The Russian mathematician Andrei Kolmogorov has shown that the rest of probability theory can be built upon these axioms and therefore they are also called *Kolmogorov's axioms* (see [Rud08] for further information).

3.1.2 Prior Probabilities

The *prior probability* $P(x)$ (also called *unconditional probability*) is the degree of belief associated with a particular proposition x in the absence of any other information. It is important to note that prior probabilities only hold if no other information is available. As soon as auxiliary information becomes available one must use *conditional probabilities*, as explained in the next section.

The specification of the prior probabilities of a random variable depends on its type. For discrete variables, this specification (called *prior probability distribution*) can be provided by enumerating all possible states and the corresponding probabilities. For instance, the probabilities for certain weather forecasts might be (example from [RN02]):

$$\begin{aligned} P(\text{Weather} = \text{sunny}) &= 0.7 \\ P(\text{Weather} = \text{rain}) &= 0.2 \\ P(\text{Weather} = \text{cloudy}) &= 0.08 \\ P(\text{Weather} = \text{snow}) &= 0.02 \end{aligned}$$

Information on a combination of multiple variables can be expressed by enumerating all possible combinations of events and their respective probabilities. This is called a *full joint probability distribution*. To illustrate this, table 3.1 depicts the example of a full joint distribution for a dentist consultation. The domain consists of three random variables: *toothache* indicates whether the patient has a toothache, *cavity* indicates whether the patient has a cavity, and *catch* indicates whether the dentist's instrument gets caught in the patient's tooth. Each entry in the table denotes the probability for a specific combination of events.

The probability of particular subset of variables (called *marginal probability*) can be obtained by summing over all entries in which these variables are true. For instance, the first row in the example depicts all cases in which *cavity* is true. Hence the marginal probability for *cavity* is:

$$P(\text{cavity}) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2$$

If a random variable has a continuous domain, it may take on infinitely many values and therefore enumeration is not possible. Instead, the probability that a random variable X takes on a value x is usually defined as a parametrized function of x. For instance, a very simple continuous function can be used if x can take on any value in the interval $[a, b]$ with the same probability. This can be expressed by the uniform distribution:

$$f(x) = \begin{cases} \frac{1}{a-b}, & \text{for } a \leq x \leq b \\ 0, & \text{else} \end{cases}$$

Such a distribution function is called *probability density function*. Probably the most widely used distribution function for continuous variables in Bayesian networks is the gaussian or

	toothache		¬toothache	
	catch	¬catch	catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

Table 3.1: Example of a full joint distribution for the dentist word (from [RN02])

.

normal distribution which is in fact used for all continuous variables in this work:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The shape of this distribution function is determined by its parameters mean μ and variance σ^2. The fact that a random variable X is distributed normally with certain parameters μ and σ^2 is for simplicity commonly denoted by

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

3.1.3 Conditional Probabilities

The prior probability of any proposition x is not applicable any more if some further information y concerning this proposition is obtained. Instead, *conditional probabilities* (also called *posterior probabilities*) have to be used. This is expressed by $P(x|y)$ and has the meaning "the probability of x, given that y is known".

Conditional probabilities can also be expressed in terms of unconditional probabilities as shown in the following equivalent equations:

$$P(x|y) = \frac{P(x \wedge y)}{P(y)} \tag{3.1}$$

$$P(x \wedge y) = P(x|y)P(y) \tag{3.2}$$

For instance, the unconditional probability in table 3.1 for *cavity* is $P(\text{cavity}) = 0.2$, as shown in the previous section. However, if the patient complains about a toothache, additional information is available and the prior probability is no longer applicable. Instead, to obtain the new probability for *cavity*, one has to calculate the conditional probability of a cavity if the patient has a toothache. This can be done by calculating the marginal probabilities for *cavity and toothache* and *toothache* first:

$$\begin{aligned} & P(\text{cavity} \wedge \text{toothache}) && = 0.108 + 0.012 = 0.12 \\ & P(\text{toothache}) && = 0.108 + 0.012 + 0.016 + 0.064 = 0.2 \\ \Rightarrow \quad & P(\text{cavity}|\text{toothache}) && = \frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} = \frac{0.12}{0.2} = 0.6 \end{aligned}$$

Apparently, the additional information leads to a significant change in the belief state. This is a very important property that will be exploited in the design of probabilistic reasoning systems.

3.1.4 Independence

When analyzing the probabilities of a set of random variables, it is important to know whether these variables influence each other. In the previously explained example of the dentist consultation all of the variables depend on each other, which means that obtaining information about one variable alters the belief state of the remaining variables. This example can be augmented with the aforementioned random variable *weather*. While this leads to a larger state space, it is obvious that for example the probability of a cavity is not influenced by the current weather. Therefore *weather* and *cavity* are *independent* variables. In fact, none of the entries from table 3.1 would change by obtaining knowledge about the weather.

This is especially important in the context of conditional probabilities. If two events x and y are independent, the conditional probability simplifies to:

$$P(x|y) = P(x)$$

Two random variables X, Y are *conditionally independent* if their relationship does not satisfy the independence condition in general, but if they become independent once another random variable z is known, i.e., X and Y are conditionally independent given Z:

$$P(x|y \wedge z) = P(x|z)$$

To illustrate the concept of conditional independence, consider the probability for a failure of the headlights of a car. Assume that there may be two causes for a headlight failure: if the car's generator is broken both of the lights will fail. If a bulb burned through, only the corresponding headlight will fail. Since both lights depend on a working generator, their failure probabilities are obviously not independent. However, if the generator is known to be working, each light can only fail due to a burned through bulb and therefore the failure probabilities become independent. Consequently, the failure probabilities of the headlights are independent given the state of the generator.

3.1.5 Bayes' Rule

The *Bayes' rule* is a very important relation between two reverse conditional probabilities stemming from equation 3.2 (named after the English minister Thomas Bayes, who formulated this rule in the early eighteenth century).

As explained in [Pea88], the rule is formulated as

$$P(x|y) = \frac{P(y|x) \cdot P(x)}{P(y)} \tag{3.3}$$

and states that the degree of belief of some event x after obtaining some additional information y can be calculated by multiplying the previous belief $P(x)$ with the so-called *likelihood* $P(y|x)$. The likelihood gives the probability that event y will happen if x is actually true. $P(y)$ is a normalizing constant which can be calculated by requiring that $P(x|y)$ and $P(\neg x|y)$ sum to unity. Since the individual terms from this rule are required for subsequent explanations, their respective names are summarized again:

- $P(x)$: prior probability (or simply prior)
- $P(x|y)$: posterior probability (or simply posterior)
- $P(y|x)$: likelihood
- $P(y)$: normalizing constant

Key predication of this rule is the fact that every conditional probability can be expressed in terms of its reverse. While this may not seem very beneficial at first glance, in practice there are many situations with good probability estimates for the right hand side, but poor estimates on the posterior. A simple example from [RN02] can illustrate the rule's usefulness:

Consider a simple medical diagnosis scenario concerning meningitis: a doctor may know that meningitis causes a stiff neck 50% of the time. Also, the doctor may have information about some unconditional facts: the probability that any patient has meningitis (denoted

by m) is $\frac{1}{50000}$, and the probability that any patient complains about a stiff neck (denoted by s) is $\frac{1}{20}$. Now the doctor is concerned with the task of estimating the probability of a patient suffering from meningitis, if he has a stiff neck. While this problem could be solved through an appropriate survey, the solution can also be obtained by using Bayes' rule:

$$\begin{aligned} & P(s|m) && = 0.5 \\ & P(m) && = \frac{1}{50000} \\ & P(s) && = \frac{1}{20} \\ \Rightarrow \quad & P(m|s) && = \frac{P(s|m)P(m)}{P(s)} = \frac{0.5 \cdot \frac{1}{50000}}{\frac{1}{20}} = 0.0002 \end{aligned}$$

Now, to reveal the real power of Bayes' rule, imagine that there is a sudden epidemic of meningitis and therefor the corresponding prior $P(m)$ goes up. Any information about the posterior obtained by statistical observations would become useless in this scenario. The Bayesian solution however can be easily adapted by adjusting $P(m)$. According to equation 3.3, the posterior should increase proportionately with the prior and, most important, the causal information $P(s|m)$ remains unaffected by the epidemic. Hence the use of this kind of model-based knowledge yields the required robustness to construct any reliable probabilistic reasoning system.

3.2 Bayesian Networks

3.2.1 Semantics of Bayesian Networks

As described in the previous section, knowledge about domains with uncertainty can be provided through full joint probability distributions. However, next to the aforementioned difficulty of gaining sound knowledge about each possibly entry in the full joint distribution, handling of these distributions explicitly may become intractable with an increasing size of the model.

These obstacles can be resolved by using *Bayesian belief networks* or *Bayesian networks* for short. Formally, a Bayesian network is a *directed acyclic graph (DAG)* with the following specifications, as described in [RN02]:

- The network's node are made up by the random variables of the domain. Hence, each variable is represented by a corresponding node in the network.
- Nodes can be pairwise connect by directed links. If there is a link from some node X to another node Y, X is said to be a parent of Y (and vice versa, Y is a child of X). Links in the network denote dependencies among random variables.
- Each node X_i has a conditional probability distribution $P(X_i|Parents(X_i))$ that quantifies the effect of the parents on the node.
- The Network does not contain any directed cycles (hence the name directed acyclic graph).

To illustrate this concept, figure 3.1 depicts the Bayesian network corresponding to the previously discussed dentist domain. This structure facilitates an easy identification of the domain's key features: the isolated node *weather* shows that it does not influence the remainder of this domain. The events *toothache* and *catch* both depend on the state of *cavity*. Also, due to the missing link between *toothache* and *catch*, it is obvious that both

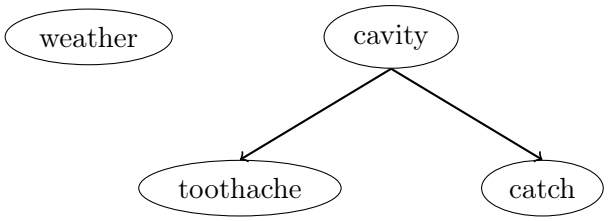


Figure 3.1: Bayesian network for the dentist domain (from [RN02])

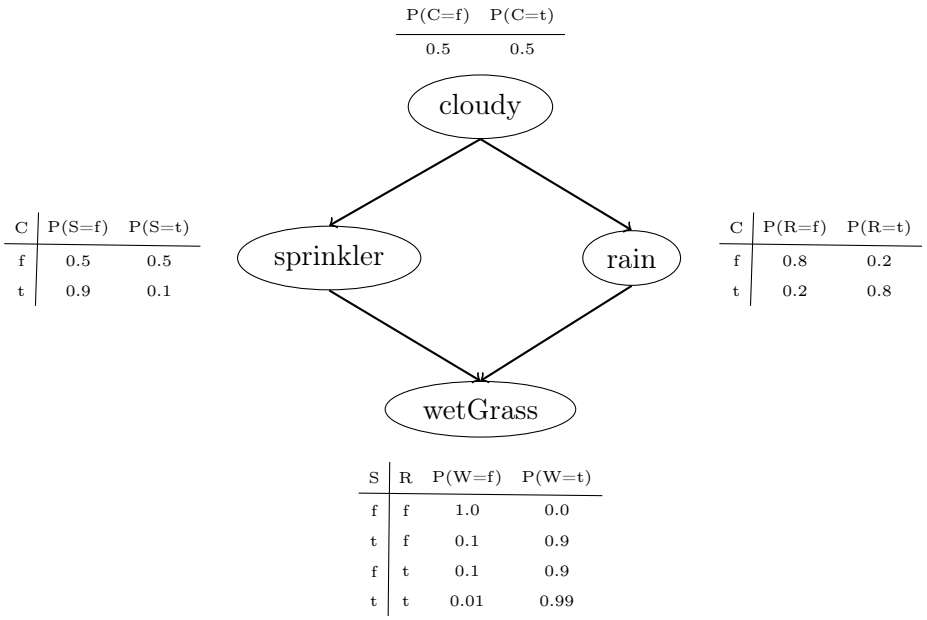


Figure 3.2: Bayesian network for the sprinkler domain with corresponding probability distributions (from [RN02])

variables are conditionally independent given that the state of *cavity* is known.[1]

A Bayesian network can be seen as an alternative presentation of a domain's full joint probability distribution. Instead of providing the full distribution explicitly, it is sufficient to provide conditional probabilities for each node. Thereby, the probability for each particular combination of events $x_1, ..., x_n$ can be obtained by

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i | parents(X_i)) \tag{3.4}$$

To further explore the potential of Bayesian networks, consider another simple example from [RN02] (a more detailed analysis of this example can be found in [Mur07]). As depicted in figure 3.2, this domain consists of four binary random variables: *cloudy* indicates whether it is a cloudy day, *rain* indicates whether it has rained, *sprinkler* indicates whether the sprinkler has been turned on, and *wetGrass* indicates whether the grass is wet. The dependencies in the graph are straightforward: a cloudy day brings a high probability of rain and thereby renders it less probable that the sprinkler is turned on. Both sprinkler and rain will likely cause the grass to be wet. First, it is assumed that the only variable which can be observed in this scenario is the grass, all other variables are hidden (i.e., they cannot be observed) and therefore their most probable states have to be inferred depending

[1] Consequently, the task of identifying conditional independences in Bayesian networks is identical to the graph theoretical problem of *d-separation*. See [Pea88] for details.

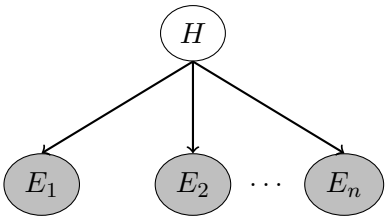


Figure 3.3: Naive Bayes model

on the grass's state. Now one might be interested for example in the probability that the sprinkler was turned on given that the grass is wet. Hence the evidence is $W = t$ while the states of all other nodes are unknown. Application of the rules from the previous section yields to $P(S = t|W = t) \approx 0.43$. Now suppose that the new information that it has rained before $(R = t)$ is acquired and this new fact is added to the evidence. Calculating the probability for the sprinkler again now leads to $P(S = t|W = t \wedge R = t) \approx 0.19$ which is significantly lower than before. Since rain is known to cause wet grass, this result is not surprising as it provides a feasible explanation for the evidence. In other words, without any additional knowledge, the causal relationships between wet grass and rain and sprinkler lead to a rather high degree of belief in both causes. However, since the knowledge about rain provides a satisfactory explanation for the grass being wet there is little reason to assume that the sprinkler was turned on also and therefore the corresponding degree of belief decreases. This fact is an important capability of Bayesian networks and is usually referred to as *explaining away* the evidence (the rain "explained away" the fact that the grass is wet).

If a causal model exhibits a clear distinction between observed states (called *evidence*) and hidden states which are to be inferred (called *hypothesis*), it is common to denote the states by E and H. To allow for an easy distinction between observed and hidden nodes the following depictions of Bayesian networks follow the standard convention of shading observed nodes.

The simplest and most commonly used Bayesian network model (and the only one employed within this work) is called the *naive Bayes model*. In this model, the class variable H is the root and the feature variables $E_1, ..., E_n$ are the leaves, as depicted in figure 3.3. This model is called "naive" because it assumes that all evidence variables are conditionally independent of each other, given the class. Since most domains in fact do not satisfy this assumption the use of a naive Bayesian network often leads to an erroneous model and therefore the naive Bayesian approach is sometimes also called "Idiot Bayes". However, even though such a network may be known to reflect an incorrect model of the corresponding domain, it may still be able to perform surprisingly well in practice. This fact has been subject to intensive research (e.g., [DP97], [Ris01], and [HY01]) and will be examined in more detail in the following chapter.

Main advantage of this naive model is its easy calculation of joint probabilities. According to Bayes' rule (equation 3.3) the conditional probability for this model can be expressed as

$$P(H|E_1, ..., E_n) = \frac{P(H)\ P(E_1, ..., E_n|H)}{P(E_1, ..., E_n)}.$$

As explained before, the interesting part of this expression is its numerator since the denominator does not depend on H and is therefore constant with given evidence $E_1, ..., E_n$. According to equation 3.1 the numerator is equivalent to the joint probability

$$P(H \wedge E_1 \wedge ... \wedge E_n)$$

which through repeated application of the product rule (equation 3.2) can be expressed as

$$\begin{aligned} P(H \wedge E_1 \wedge ... \wedge E_n) &= P(H)\ P(E_1 \wedge ... \wedge E_n|H) \\ &= P(H)\ P(E_1|H)\ P(E_2 \wedge ... \wedge E_n|H \wedge E_1) \\ &\vdots \\ &= P(H)\ P(E_1|H)\ P(E_2|H \wedge E_1)...P(E_n|H \wedge E_1 \wedge ... \wedge E_{n-1}) \end{aligned}$$

Due to the assumption of conditional independence the individual terms of this expression can be simplified to (with $i \neq j$)

$$P(E_i|H \wedge E_j) = P(E_i|H)$$

and therefore the joint probability is simplified to

$$\begin{aligned} P(H \wedge E_1 \wedge ... \wedge E_n) &= P(H)\ P(E_1|H)...P(E_n|H) \\ &= P(H) \prod_{i=1}^{n} P(E_i|H) \end{aligned}$$

and thus the conditional distribution over H can be expressed as

$$P(H|E_1 \wedge ... \wedge E_n) = \frac{1}{z} P(H) \prod_{i=1}^{n} P(E_i|H) \tag{3.5}$$

with a scaling factor z only depending on the evidence. Therefore, such naive Bayesian models are much more manageable compared to any alternative models since they can be separated into a class prior $P(H)$ and independent probability distributions $P(E_i|H)$.

3.2.2 Inference in Bayesian Networks

In order to facilitate reasoning for domains with uncertainty, there are two main inference tasks that have to be performed in Bayesian networks, namely inferring the state of unobserved variables and estimating the parameters of unknown probability distributions.[2] Although there exists a vast variety of different inference algorithms, this section is constrained to the algorithms employed within this work.

Parameter Estimation from Complete Data via ML Learning

In most probabilistic reasoning scenarios the causal dependencies of the target domain are identified by domain experts and a corresponding Bayesian network is designed. Hence, the initial situation for a learning task consists of a Bayesian network of known structure and known *types* of conditional probability distributions, but *parameters* of the distribution functions are yet to be determined. This parameter determination can be performed with the use of sets of exemplary data (usually called training data) and therefore the task of parameter estimation is commonly also called *statistical learning* or *training*.

The choice of an appropriate learning technique depends mostly on the available training data. If every instance of the training data comprises values for all random variables (i.e., the domain is *fully observable*), the learning task is only concerned with estimating

[2] If dependencies of random variables within the domain are unknown, a third major inference task is *structure learning*. Since all models introduced in this work are built upon a pre-defined and fixed structure, this task is not explained here.

unknown parameters of the probability distribution functions such that they represent the most likely reflection of the domains correlations. This learning process is called *parameter estimation from complete data* and is usually performed through *Maximum-Likelihood Learning* or *ML Learning* for short.

Informally speaking, ML learning estimates numerical parameters such that the resulting probability distributions yield the most plausible model of the observed training data.

To simplify notation in the following explanations, the complete training instances are denoted by $X = \{x_1, ..., x_n\}$, i.e., each x_i denotes a certain pair of observation e_i and corresponding hypothesis h_i from the training set. The likelihood distribution of the training set X is denoted by $P(X|\theta)$ with respect to the conditional distribution's unknown parameter(s) θ to be determined. Since all realizations are assumed to be independent and identically distributed (i.i.d), the distribution can be factorized as follows:

$$P(X|\theta) = P(x_1, ..., x_n|\theta) = \prod_i P(x_i|\theta)$$

Since the realizations x_i are observed and therefore fixed, the distribution function can be seen as a function of θ (the so-called likelihood function):

$$\mathcal{L}(\theta) = P(X|\theta) = \prod_i P(x_i|\theta) \tag{3.6}$$

Maximizing this expression with respect to θ results in a maximum likelihood estimate (*MLE*, denoted by $\hat{\theta}$) and thereby provides the best estimate of distribution functions for the given training data. Hence with a parameter space Θ, ML learning is performed by solving

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \mathcal{L}(\theta) \tag{3.7}$$

As a result, a full specification of distribution functions is obtained, which provides the best reflection of model correspondences exhibited in the training data.

Note that equation 3.7 may be rather inconvenient to solve due to the product in equation 3.7. To ease computations, it is common to exploit the fact that the logarithm provides a monotone transformation of a distribution function. Hence, instead of maximizing the likelihood function directly, one can also use its logarithmic transformation ℓ called *log likelihood*:

$$\ell(\theta) = \ln \mathcal{L}(\theta) = \sum_{i=1}^{n} \ln P(x_i|\theta) \tag{3.8}$$

Since the logarithm is a monotone transformation, maximization of $\ell(\theta)$ will yield the same result as maximizing $\mathcal{L}(\theta)$, but, due to the replacement of the product in equation 3.6 with the sum in equation 3.8, the maximum of the log likelihood is usually easier to calculate and therefore results in a speedup of the learning process. as both variants yield the same results, they do not alter the principle of this learning procedure but merely influence the processing speed. Consequently, even though the logarithmic functions are used for the implementation of the models presented in this work, the following explanations occasionally omit this step for the sake of simplicity.

Parameter Learning with Missing Values via the EM Algorithm

The fairly easy task of ML learning is only applicable if the values of all random variables can be observed from the training data. If some of the values are not observable (i.e., the domain contains hidden states), another learning approach has to be employed, namely the

Expectation Maximization Algorithm or *EM Algorithm* for short. The lack of observable values for some variables requires an additional parameter compared to ML learning: next to the already defined observations X and distribution parameters θ, the domain now contains one or more latent variables denoted by Y. Thus, next to estimating θ, the learning procedure is now also tasked with inferring values for Y.

Main goal of this procedure is again the determination of a maximum likelihood estimate. However, due to the latent variable Y all of its possible values must be taken into account when calculating the likelihood of observed data:[3]

$$\mathcal{L}(\theta) = P(X|\theta) = \sum_{Y} P(X \wedge Y|\theta)$$

Since a maximization of this expression is often intractable, the EM algorithm offers an alternative approach of finding a maximum likelihood estimate.

Informally speaking, the idea behind this algorithm is the replacement of latent variables with plausible values (called *imputation*) and a successive calculation of an ML estimate based upon these assumed values. This MLE can be used afterwards to guess new (possibly more plausible) values for the latent variables. Consequently, this algorithm proceeds with an alternating execution of guessing plausible values for Y and maximizing the corresponding likelihood.

More formally, the algorithm proceeds by performing the following two steps iteratively. Since θ varies for each iteration, the parameter of iteration i is denoted by $\theta^{(i)}$. Even though the results of this algorithm are heavily dependent on an appropriate initialization of the parameter $\theta^{(0)}$, for simplicity it is assumed that no prior knowledge concerning this parameter is available and therefore it is set to some value randomly. Strategies for an expedient initialization are discussed for instance in [BCG03].

- **Expectation Step (E-step)**
  First, a plausible value for Y is determined based on the previous iteration's parameter estimate $\theta^{(i-1)}$, i.e., Y is set to the y that maximizes $P(y|X \wedge \theta^{(i-1)})$. This is used to obtain a new estimate of θ through the so-called Q-function. This function calculates the expected value E of the log likelihood function with respect to the conditional distribution $P(Y|X)$ and the current parameter estimate $\theta^{(i-1)}$:
  $Q(\theta) = Q(\theta|\theta^{(i-1)}) = E_{P(Y|X \wedge \theta^{(i-1)})}\big(\ell(\theta)\big)$
- **Maximization Step (M-step)**
  After the expected value has been calculated, the next estimate $\theta^{(i)}$ is determined such that
  $\theta^{(i)} = \arg\max_{\theta} Q(\theta|\theta^{(i-1)})$.
  This step can be carried out through ML learning as described before.

These steps are performed alternately until the resulting $\theta^{(i)}$ do not exhibit significant changes in successive iterations any more. Though it has been shown that this procedure will always converge (discussed for example in [MK96]), it is not guaranteed to reach a maximum likelihood estimate but instead the algorithm may also stall in a local maximum.

Next to the obvious advantage of being able to cope with hidden variables, this algorithm is quite popular because it is usually easy to implement and is known to reach at least some local maximum (i.e., it will always return a solution even though not necessarily the optimal one). As mentioned before, the algorithm is heavily dependent on a feasible initialization

[3] Y is assumed to be a discrete variable in this case. If it is a continuous variable instead, the sum would have to be replaced by an integral.

of its parameter $\theta^{(0)}$, since this selection may determine whether the result is limited by some local maximum. Thus the problem of undesired local maxima may be resolved (or at least diluted) by executing the algorithm multiple times with varying initial parameters. However, as convergence of this algorithm is rather slow, such a multiple execution may be inappropriate if computing time is an important factor. Alternative improvements may be achieved through sophisticated strategies of determining feasible initial parameters, as described for instance in [BCG03].

Prediction of Hidden Variables via the Junction Tree Algorithm

Once the network has been trained successfully, it can be used for prediction, i.e., inferring the state of a hidden variable H for a certain observation e, or, more formally speaking, it is the task of finding the h_i that maximizes $P(H|e)$.

The most simple approach of answering queries about the states of hidden variables is simply summing over terms from the full joint distribution, as explained in section 3.2.1. However, with increasing size of the network such an explicit enumeration of different terms from the conditional probability tables will quickly become intractable. More efficient approaches have been developed to allow for an admissible processing time. Algorithms for the inference of hidden variables have been subject to intensive research and therefore a large variety of different inference algorithms is available. In general, these algorithms can be classified into two types of inference: *exact inference* yields results that are in exact accordance with the probability model while *approximate inference* usually facilities an increase in computational speed but only produces approximate solutions.

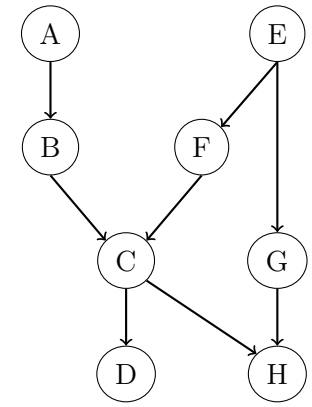
Thus, alternative algorithms mostly differ in computational speed and precision. The inference algorithm employed in this work is the *junction tree algorithm* (called JTA subsequently), which belongs to the class of exact inference algorithms. There may be other algorithms that are suited equally well (and it terms of speed probably significantly better) for the required inference tasks, but since this algorithm yields exact results and computational speed is not a vital issue within the context of this work, this algorithm proved to be a sufficient choice. For more information on different inference algorithms for Bayesian Networks see for example [RN02], [Pea88], and [CGH96].

This section only summarizes the JTA's functionality, for an in-depth analysis of its features and justification see [CGH96]. Aim of this algorithm is the transformation of a Bayesian network into an undirected tree in order to provide the means of efficiently computing exact marginals. The algorithm comprises four steps as listed below, each of which is explained subsequently.

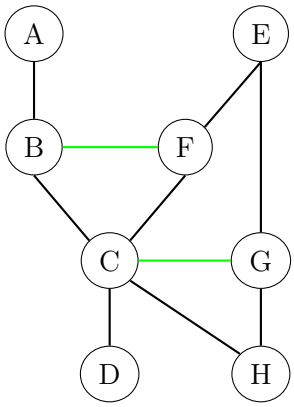
1. Moralize the network
2. Make the graph chordal through triangulating
3. Form the junction tree
4. Inference in the junction tree

Moralizing

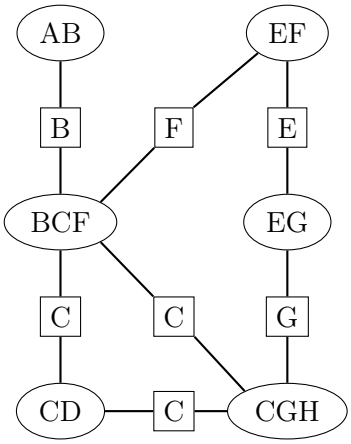
The moral graph corresponding to a Bayesian network is obtained by first creating links between every pair of nodes with a common child and then dropping the directionality of all links. This process is called *moralizing* (if two parents share a common child, they should be "married" in order to adhere to certain concepts of morality, hence the name of this step). To illustrate this, figure 3.4a depicts some arbitrary Bayesian network and figure 3.4b depicts the corresponding moralized graph.



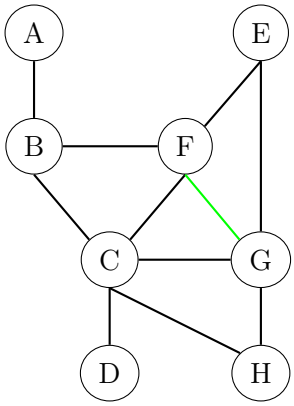
(a) Original Bayesian network



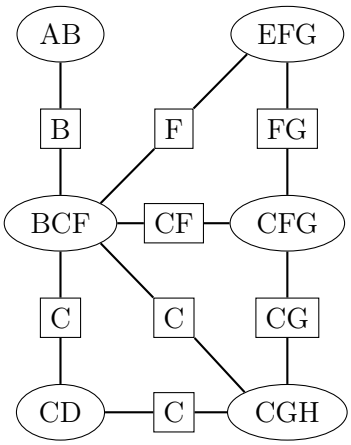
(b) Moralized undirected graph, additional links are marked green



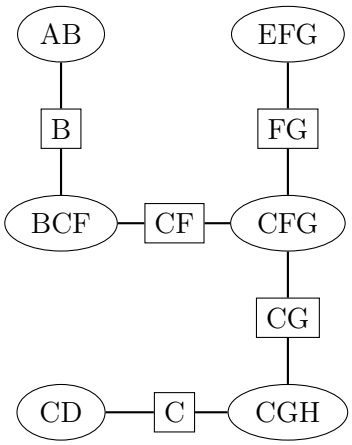
(c) Junction graph from (b)



(d) Chordal graph, the additional link is marked green



(e) Junction graph from (d)



(f) Resulting junction tree

Figure 3.4: Transformation of a Bayesian network into a junction tree

This transformation of a DAG into an undirected graph results in a loss of some conditional independencies. To ensure an equivalent representation of all original dependencies, the resulting graph is clustered into *cliques* (i.e., subgraphs in which all pairs of nodes are connected) and *separators* (i.e., subgraphs which form an intersection between individual cliques). See figure 3.4c for an example (cliques are denoted by ellipses, separators by rectangles). Such a graph is called *junction graph*. Consequently, separators in a junction graph adopt the function of class variables in Bayesian network as they denote conditional independence of the separated cliques.
Each of the resulting links is assigned a so-called potential ϕ which denotes a certain weight of this connection. Since the aim of the algorithm is the inference of marginals, these potentials should be defined such that they can be used to retrieve joint probabilities. The correct determination of potentials is the actual inference task in this algorithm and will be explained below, but it should be already noted that it is the goal to determine all potentials such that a joint probability for a set of nodes X can be calculated by dividing the product of all involved cliques C_i and separators S_j:

$$P(X) = \frac{\prod_i \phi(C_i)}{\prod_j \phi(S_j)} \tag{3.9}$$

Triangulating
Equation 3.9 is only applicable if the cluster graph fulfills the condition that for all pairs of nodes X and Y, all nodes on the path between X and Y must contain the intersection of X and Y. This is also called the *running intersection property*. To illustrate this property, consider for instance the nodes *BCF* and *CGH* in figure 3.4c. The intersection between these nodes is formed through the separator C and thus, in order to fulfill this condition, all connections between these nodes must contain C. However, the graph also provides a detour via *EF* which does not include C and hence does not comply with the running intersection property.
It has be shown (see [CGH96]) that compliance with the running intersection property can be ensured by using *chordal* graphs. A graph is said to be chordal if and only if each of its cycles with four or more nodes contains a *chord*, which is a link joining two non-adjacent nodes in the cycle. To illustrate this, consider figure 3.4b again: the cycle $\{E, F, C, G\}$ does not contain a chord and therefore this graph is not chordal. This can be solved by simply adding chords to this cycle until it is chordal, as depicted in figure 3.4d. The addition of further links introduces new triangular relations and therefore this process is called *triangulation*.
From the resulting triangulated graph one can construct another junction graph as depicted in figure 3.4e.

Constructing the Junction Tree
By assigning a weight corresponding to the separator size for each connection of cliques, one can construct a *minimum spanning tree (MST)* from this junction graph. An MST of a graph is a subgraph which forms a tree and connects all nodes of the graph. It is said to be minimal if no other spanning tree with less weight can be found. Result of such an MST search on a junction graph results in a tree with separators of maximum size as depicted in figure 3.4f (if their weights reflect the size of the separators, this would be strictly speaking a maximum tree rather than a minimum one, but the procedures for finding this tree are the same for MSTs). Detailed descriptions on algorithms for finding minimum spanning trees can be found for example in [PR02] and [FW94]. Note that neither the junction graph nor the resulting junction tree is guaranteed to be unique, but it is guaranteed that

a valid junction tree can be constructed from any triangulated graph.

Inference in the Junction Tree

As mentioned before, this algorithms aims at providing potentials that reflect marginal probabilities. Thus, the actual inference task is a modification of potentials such that they fulfill the following requirements:

- the joint probability is consistent (global consistency)
- adjacent cliques are consistent (local consistency)
- clique potentials are equivalent to clique marginals
- separator potentials are equivalent to separator marginals

To initialize the junction tree, the potentials of all clusters and separators are set to $\phi(C_i) = 1$ and $\phi(S_i) = 1$. Afterwards some node X is picked, a Clique C_i containing X and $Parent(X)$ is selected and the clique potential is multiplied with the corresponding conditional probability: $\phi^*(C_i) = \phi(C_i) \cdot P(X|Parent(X))$. This step is repeated for all nodes X.

After the values have been initialized, a two-way message passing scheme is used to modify the potentials such that the aforementioned requirements are fulfilled. To illustrate this, consider the schematic junction tree with two cliques X and Y and a separator S depicted in figure 3.5. In order to ensure local consistency it is required that for two adjacent cliques the marginals on their separator must be equal:

$$\sum_{X \setminus S} \phi(X) = \phi(S) = \sum_{Y \setminus S} \phi(Y)$$

Consequently, a so-called "absorption message" is sent from X to S containing a new potential ϕ^* for S (denoted as message 1 in the figure):

1. $\phi^*(S) = \sum_{X \setminus S} \phi(X)$

As already stated in equation 3.9, a joint probability can be expressed as a ratio of potentials of cliques and separators. Therefore the second message passed from S to Y updates the potential accordingly:

2. $\phi^*(Y) = \dfrac{\phi(Y)\phi^*(S)}{\phi(s)}$

Subsequently, the equivalent messages are sent back the other way:

3. $\phi^{**}(S) = \sum_{Y \setminus S} \phi^*(Y)$

4. $\phi^*(X) = \dfrac{\phi(X)\phi^{**}(S)}{\phi^*(s)}$

After this message passing (also called *belief propagation*) has been carried out, one obtains a consistent junction tree which fulfills all of the previously stated requirements.

Finally, predictions of hidden states can be inferred by setting the potentials of evidence

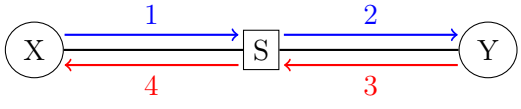


Figure 3.5: Message passing in a junction tree

nodes to the observed values. This introduces temporary inconsistencies in the junction tree which are resolved by a new turn of the described message passing. The forward messages (denoted by the blue arrows in figure 3.5) are then usually called *evidence collection* and the backward messages (denoted by the red arrows) are called *evidence distribution*. After all distribution messages have been sent, the tree is again in a consistent state and, since the resulting potentials reflect marginals, the predictions of hidden states can be obtained by simply reading the potentials of hidden nodes. This message passing scheme is also called *belief propagation*.

3.3 Dynamic Bayesian Networks

The means of probabilistic reasoning presented so far are only concerned with *static* worlds, in which each random variable has a single fixed value. However, there are many domains in which one deals with time series of data, i.e., one has a set of data instances so that the random variables take on different values for each instance and successive instances are not independent of each other but have *temporal* relations. *Dynamic Bayesian networks (DBN)*[4] provide an extension to Bayesian networks in order to cope with such temporal relations. The following provides a summary of the properties of Dynamic Bayesian Networks. More information can be found for example in [RN02] and [HW99].

3.3.1 Domain Requirements for Dynamic Networks

In order to allow for the use of a Dynamic Bayesian network, the modeled domain has to fulfill certain requirements, namely it has to be stationary and Markovian. These properties are each explained subsequently.

Stationary Processes

To allow for the representation of a domain with a DBN, the dependencies of the domain must be governed by stationary processes. A process is stationary if its properties do not change over time, i.e., the joint probability distribution remains unaffected by temporal shifts. As a result, if parameters of such a process have been learned once, they can be applied to future scenarios as well. In other words, a stationary process is only subject to relative differences in time but is independent of the actual time itself.

To illustrate this, consider the weather forecasting domain again. One may have certain evidence (e.g., clouds in the sky, temperature, or humidity) on a particular day and is concerned with forecasting weather on the next day. The correlations between evidence and forecast should be unaffected by the exact date of the forecast, i.e., if for example a cloudy sky is observed today and therefore it is forecasted to rain tomorrow, the same result should be obtained when a cloudy sky is observed some time next week and the following day's weather is forecasted. If the relationship between today's evidence and tomorrow's forecast is applicable completely independent of the actual day, the process of weather forecasting is said to be stationary. More formally, a process is stationary if

$$P(X(t)|X(t-1)) = P(X(t+\tau)|X(t+\tau-1)) \quad \forall\, t, \tau \in \mathbb{R}^+. \tag{3.10}$$

[4]The name Dynamic network can be somewhat misleading because one might expect a network with dynamically changing structure. However, the notion of a dynamic network denotes that the domain changes dynamically over time while the network's structure remains the same. Temporal Bayesian network might be a better name for such a network but it has become standard to call such networks dynamic.

Though it is required to employ stationary processes when designing DBNs, this requirement can be relaxed in practice. For instance, it is apparent that the stationary assumption in the weather domain simplifies the domain too much to capture all relationships correctly. Obviously, the forecasts are not only subject to a certain day's evidence but they also exhibit seasonal effects (e.g., the probability for a rainy day is presumably higher in the fall than in the summer). These seasonal effects will lead to a slowly changing environment. As a consequence, if certain relations have been learned currently, they should also apply when obtaining a forecast next week but probably they will not apply exactly when obtaining a forecast in half a year. Such a process is called *locally stationary* because for rather short periods of time it can be assumed to be stationary but for long time periods the stationarity is violated. As long as a process is known to be only local stationarity, it can still be employed in a DBN if the network is retrained ever so often in order to adapt to a changing environment.

Markov Processes

The second condition which has to be met is the *Markov property*. This property states that the current state of a temporal process depends only on a finite number of previous states. If a process meets this condition, it is said to be *Markovian* and is usually called *Markov process* or *Markov chain*. In other words, a Markov process is memoryless (or more precisely memory-limited) as it is allowed to forget about its history. Common definition of a Markov process is "a process for which conditional on the present state of the system, its future and past are independent" ([enc09]).

This property is of utmost importance for the facilitation of temporal reasoning because for a non-Markovian process all previous states need to be considered for inference. This would lead to an infinite growth of the state space with increasing time and thereby hinder any meaningful form of reasoning.

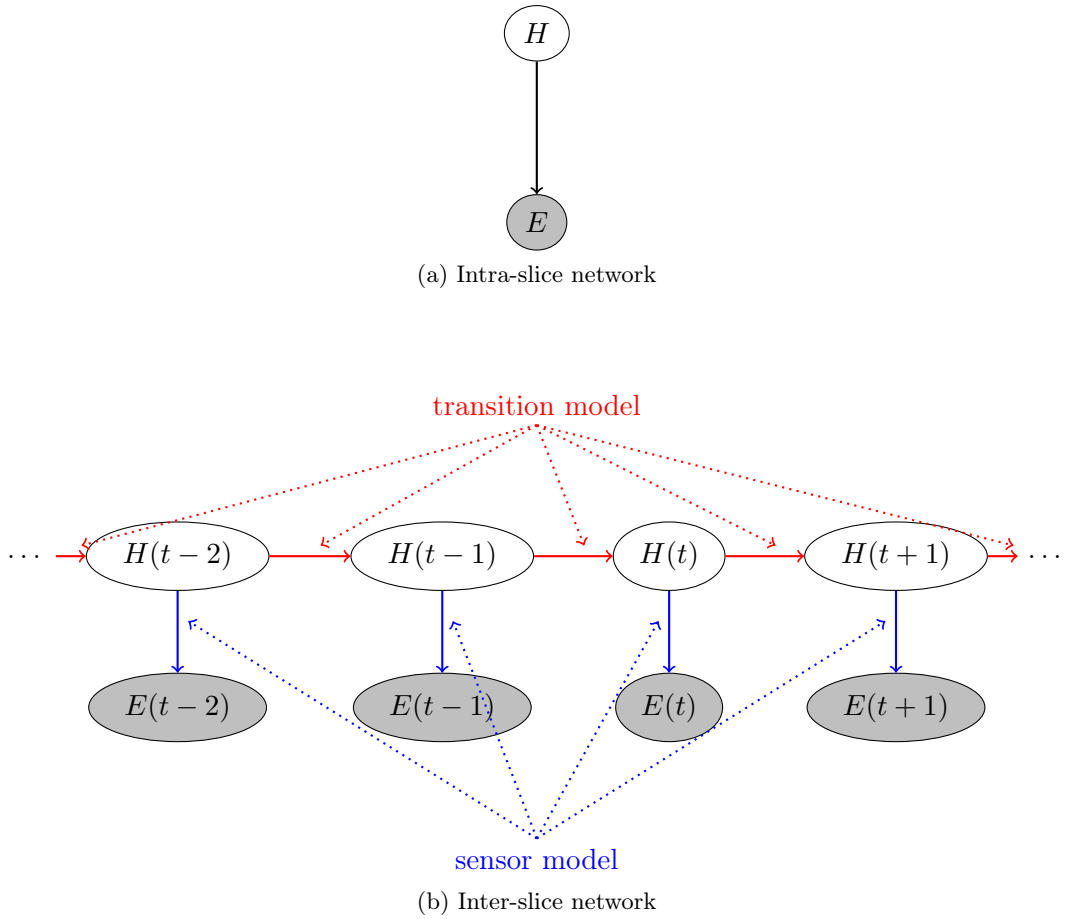
Markov processes are further classified with respect to the length of their memory (i.e., the number of previous states which influence the current state). This information is denoted by the *order* of a Markov process. For instance, in a first-order Markov process the current state depends only on the previous state while all prior states are irrelevant. More formally, a process is first-order Markovian if

$$P\big(X(t) \mid (X(1), ..., X(t-1)\big) = P\big(X(t) \mid X(t-1)\big). \tag{3.11}$$

Note that first-order Markov processes are by far the most common variant. If someone simply refers to a Markov process without explicitly stating its order, it is usually assumed to be a first-order process. In fact, all temporal processes used in this work are modeled as first-order Markov processes.

3.3.2 Temporal Models

If the relationships of a domain adhere to the aforementioned properties, one can construct a dynamic Bayesian network as a series of static networks. A single time step (also called time slice) is modeled through a static network (referred to as intra-slice network). The same static network is then repeated for each time step and additional links are introduced to denote temporal dependencies. For instance, consider the simple naive Bayesian classifier in figure 3.6a. If this network is used to model a single time slice, a corresponding DBN can be constructed by repeating this time slice over time with additional temporal dependencies, as depicted in figure 3.6b. Since this dynamic Bayesian model extends the



(a) Intra-slice network

(b) Inter-slice network

Figure 3.6: Dynamic Bayesian network

time slices with links between different slices, it is sometimes also called inter-slice network. From the temporal connections it can easily be seen that this network refers to a first-order Markov process.

Note that a Dynamic Bayesian network does not necessarily has to exhibit a structure as depicted in figure 3.6b. There may be also models in which non-consecutive time slices are connected (leading to Markov processes of higher order), or one can employ networks without hidden states. However, if the dynamic network is modeled by a Markov process and exhibits a clear distinction between observed evidence nodes and hidden hypothesis nodes, it is called a *Hidden Markov Model (HMM)*. Consequently, HMMs are a specific subcategory of DBNs.

The separation into intra-slice and inter-slice models allows for the specification of two separate probability models: the network for a particular time slice corresponds to a static network as described in the previous section and describes how a hypothesis is affected by particular evidence values. Therefore the intra-slice network is also called *sensor model*. The temporal connections describe how states evolve over time and therefore these inter-slice connections form the *transition model*. Full specification of a DBN consequently consists of specifications for both models, as depicted in figure 3.6b.

3.3.3 Inference Tasks in Dynamic Bayesian Networks

Since the use of dynamic models introduces time as an additional dimension of the domain, these models provide various means of inference. For simplicity, the tasks in the following are explained with respect to a Hidden Markov Model but they can be applied analogously to other dynamic networks as well.

- **Filtering**
  The task of filtering is concerned with computing the current belief state, i.e., the posterior of the current state given all evidence up to date. More formally, it computes
  $$P\big(H(t) \mid E(1), ..., E(t)\big)$$
- **Prediction**
  A prediction is similar to filtering except that the series of hidden states is extended into the future and a posterior is computed for some future state:
  $$P\big(H(t+\tau) \mid E(1), ..., E(t)\big)$$
- **Smoothing**
  If new evidence arrives, this has the possibility of changing the belief of a previous state in hindsight. Hence smoothing computes the posterior
  $$P\big(H(t-\tau) \mid E(1), ..., E(t)\big)$$
  $$(\text{for some } 1 < \tau < t)$$
- **Most Likely Explanation**
  If a sequence of observations is available, it is expedient to find the sequence of hidden states that most likely generated this series of observations:
  $$\arg \max_{h(1),...,h(t) \in H(1),...,H(t)} P\big(h(1), ..., h(t) \mid E(1), ..., E(t)\big)$$

These tasks illustrate the higher power of dynamic networks compared to static ones: because subsequent observations provide means of altering previous belief states in retrospective (namely through smoothing and filtering), they also allow for more accurate filtering and prediction tasks. This increase in accuracy is due to the dynamic state transitions: if a particular state can be predicted with an increased certainty, the prediction for a successive state will also become more reliable.

For actually performing these inference tasks, one can transform a DBN into a static network through "unrolling", i.e., replicating the time slices until the network is large enough to cope with the specified series of observations. As a result, one obtains a static network and therefore can apply all algorithms suitable for Bayesian networks, namely the ones presented in the previous section. Each of the inference tasks listed above then can be simply carried out by introducing the evidence and computing the values of the respective hidden variables.

Note that this technique of unrolling has the undesired side effect of an exponentially increasing complexity with a growing number of time slices and therefore is rarely the best choice for reasoning in temporal models. Many alternative algorithms have been developed which are able to perform reasoning significantly faster. However, as explained before, processing time is not a vital issue within this work and unrolling the implemented networks was able to deliver results within an acceptable time and therefore the algorithms presented in the last section are used for DBNs as well.

4 Design of Forecasting Modules with Probabilistic Reasoning

This chapter describes the design of three different probabilistic reasoning modules that facilitate forecasts of future stock price movements based on the analysis techniques introduced in chapter 2. All analysis results presented in this chapter refer to historic data of the *Dow Jones Industrial Average Index (DJI)* obtained from *Yahoo Finance*[1]. The data consists of OHLC (see section 2.4) and volume data for 20588 trading days covering the time from October 1, 1928 to September 24, 2010. The Dow Jones Industrial Average Index has been selected for test purposes because it is the world's most highly regarded economic barometer and comes with by far the largest set of historical data. A large set of historical test data is important for the purpose of this work, because it allows for an evaluation of any procedures' long-term performance. Historical data sets for other securities covered only significantly shorter time periods or were found to be incomplete.

All modules have been implemented with Kevin Murphy's open-source software *Bayes Net Toolbox for Matlab*[2].

4.1 Trend Lines: Forecasting Module 1

The first module is based upon the theory of trend lines and line patterns as described in section 2.2.1. Consequently, the first step is a preprocessing of the raw price data in order to identify trend lines.

4.1.1 Identification of Trend Lines

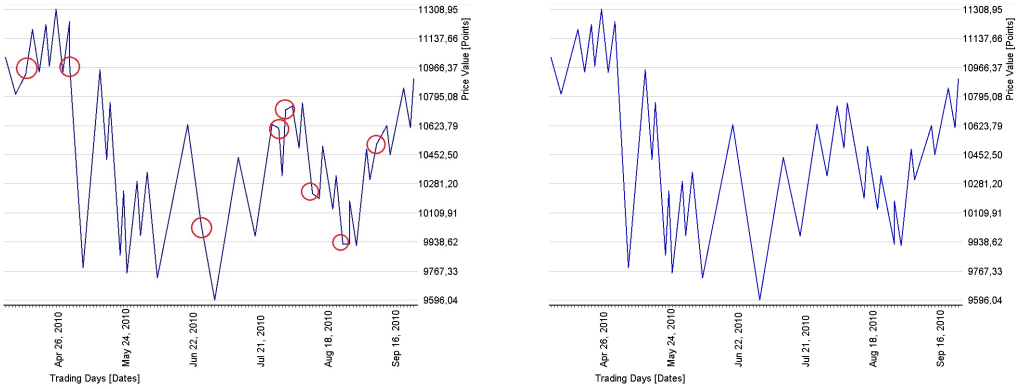
The identification of trend lines is performed in several successive steps, each of which is explained subsequently:

1. Identification of local extremes
2. Collection of local extreme points that may form a trend line
3. Construction of trend line candidates based upon these extreme points
4. Merge of similar trend lines

Identification of Local Extremes

This step analyzes each day's high and low information from the raw OHLC data and returns a sequence of local extremes. It is evident that an expedient result should comprise a sequence of alternating local highs and local lows. However, as there are two values (high and low, denoted by H_t and L_t subsequently) assigned to each day, the input is not a mathematical function and therefore standard procedures for finding local extremes

[1] http://finance.yahoo.com/
[2] http://bnt.googlecode.com/



(a) Local extreme candidates, undesired effects are marked with red circles

(b) Cleaned local extremes

Figure 4.1: Recent local extremes in the Dow Jones Industrial Average Index

have to be modified. More precisely, due to the double value assignment for each day it is possible that the time series exhibits consecutive local extreme points of the same type without an converse extreme in between (e.g., two local maxima may occur without an intermediate local minimum).

As a first step, all candidates for local extremes are filtered from the time series. The criteria for local extreme candidates are as follows:

t is a local high candidate if $H_t > H_{t-1} \quad \wedge \quad H_t > H_{t+1}$
t is a local low candidate if $L_t < L_{t-1} \quad \wedge \quad L_t < L_{t+1}$

While technically all of these points are local extremes, they do not necessarily yield the expected results yet, namely because the resulting series may contain multiple successive extremes of the same type (see figure 4.1a for an example). Obviously, such subsequences of extremes of the same type contain more than just the significant points. Therefore, further criteria for cleaning the extreme series are introduced:

- If a single day exhibits two local extremes (i.e., both H_t and L_t exist for some day t), their order in the time series is ambiguous. Since the following rules depend on the extreme's chronological order, such days are ordered as follows:
  If a particular day with two extremes has been a rising day (i.e., closing value higher than opening value), it is assumed that the local low occurred before the local high. Conversely, for falling days it is assumed that the local high occurred first.
  While in theory there may be rare occasions in which this order is not correct, these are plausible assumptions in accordance with candlestick interpretations (as explained in section 2.4) and yield sufficient results in practice.
- If two successive local extremes of the same type and exactly the same price values occur (and thereby form a plateau), they are merged into a new extreme positioned in the center between both points (as suggested in [Bao08], which is concerned with a similar problem).
- If two successive local extremes of the same type but with different price values occur, the "less extreme extreme" (i.e., the lower high or the higher low) is removed.

Application of these rules results in a clean series of local extremes, as depicted in figure 4.1b.

Collection of Local Extreme Points that may form a Trend Line

After the series of local extremes has been obtained, this series can be used as input for the search of potential trend lines. In a mathematically correct sense, a trend line should be created for each set of points whose prices $P(t)$ adhere to a linear equation of the form $P(t) = m \cdot t + n$. As stock price movements behave like a noisy signal, exact compliance with a linear equation is such a strict constraint that this criterion would barely find any of the significant trend lines. Hence, the criterion is extended by a noise parameter $v(t) \in [-VP(t); VP(t)]$ that allows prices to fluctuate around the trend line by a maximum of some tolerance value V (relative to the current price). Consequently, to form a trend line, a set of extreme points has to adhere to an equation of the form

$$P(t) = m \cdot t + n + v(t). \tag{4.1}$$

In other words, all of the line's points must reside between the lines $P(t) = m \cdot t + n - VP(t)$ and $P(t) = m \cdot t + n + VP(t)$.

Additionally, it is meaningful to limit the distance between the extreme points (denoted by $E_1, ..., E_n$ in the following) that form the line (or, in other words, a trend line has to be confirmed ever so often in order to remain valid). This is done by requiring that the time interval between any two successive extremes of a trend line is smaller than a certain threshold. This threshold should be defined with respect to the line's duration, because spacing of the confirmation points should scale with the overall length:

$$t(E_{i+1}) - t(E_i) \leq t_{max}\big(t(E_n) - t(E_1)\big) \quad \forall i = 1, ..., n-1 \tag{4.2}$$

These criteria are sufficient to test whether a set of given points forms a trend line, but since the potential line's slope cannot be known beforehand, the search criteria is still more broadened. To allow for an easier comprehension of the following search concept, figure 4.2 visualizes all required elements. The search starts for any two local extremes of the same type and two boundary lines are constructed in a way that they form a widening beam. This is done by creating one line through the points $P(t_1) + VP(t_1)$ and $P(t_2) - VP(t_2)$ and another line through the points $P(t_1) - VP(t_1)$ and $P(t_2) + VP(t_2)$. All extremes within the boundaries of those lines may potentially form a trend line. Within this area, all extreme points are collected until one of the boundary lines is broken. Following the break of a boundary line, the points collected so far are used for trying to create a trend line through linear regression (see below). Afterwards, the broken boundary line is replaced

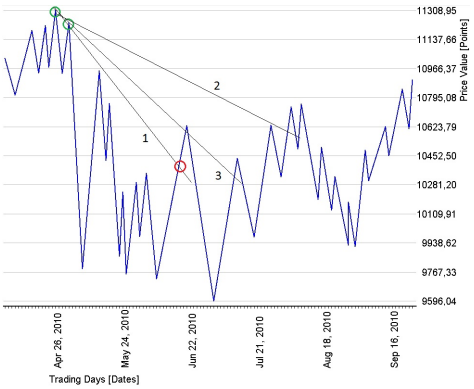


Figure 4.2: Search area for a trend line. The two initial local extremes are marked by green circles. This leads to the initial search area marked by lines 1 and 2. After boundary line 1 is broken (marked by the red circle) a new line 3 is created for a continuation of the search.

by a new valid line (which narrows the search area, as depicted in figure 4.2) and the search continues with collecting extremes again. This search then continues until one of the following termination criteria is met:

1. both boundary lines meet and thereby eliminate the search area
2. both lines drop below zero
3. the search reaches the end of the data set

Construction of Trend Line Candidates

Once a set of potential extremes has been obtained (as mentioned in section 2.2.1 such a set must contain at least three points), these points are used as input for forming a line through linear regression (i.e., determining a line so that the sum of squared residuals is minimized). This is done recursively until the result either complies with both line criteria (equations 4.1 and 4.2) or until there are less than three points remaining. If these criteria are met, the line is added to the set of resulting trend lines. If the line does not meet the criteria, the next action depends on the situation:

- If the time criterion is not met, all points after exceedance of the time limit are removed.
- If the linear criterion is not met, it must be distinguished in which direction the limit is exceeded. If a single point is too far from the line but does not yield a line break, it is simply removed.
- If the linear criterion is not met and the corresponding point yields a line break, all subsequent points are removed.

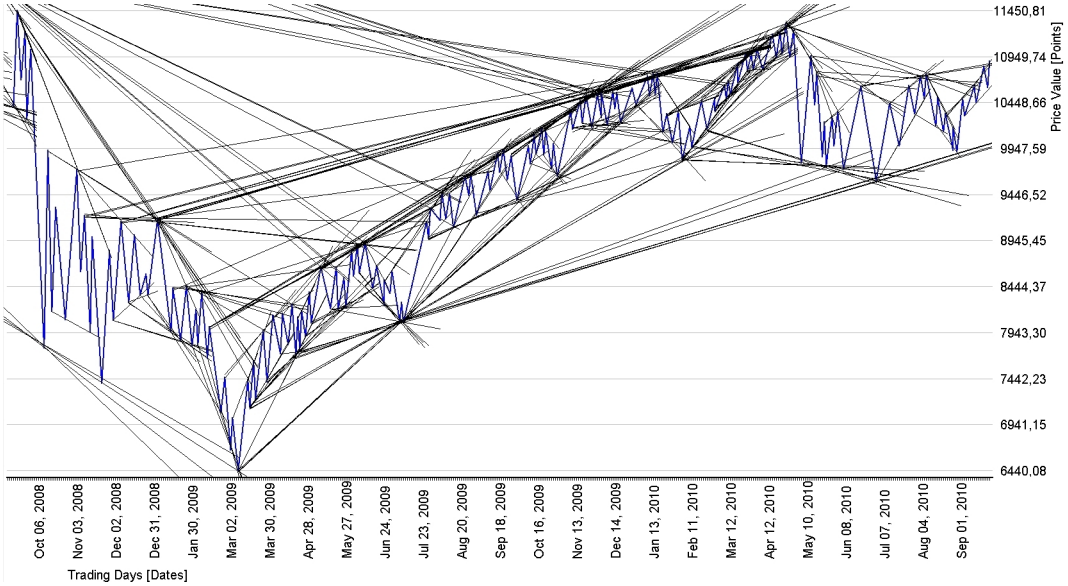
The described framework now contains all methods required for the search of trend lines. This search is performed through nested iterations:

One extreme i (starting with the first extreme) is picked as the start point for the search and the successive extreme $j = i + 1$ is selected as the second extreme to form the search area. Search in this area continues until one of the termination criteria listed above is met. If criterion 1 is met, the next search area is formed by maintaining the first extreme i and moving forward the second extreme (i.e., $\hat{j} = j + 1$). If criterion 2 or 3 is met, results for extreme i are exhaustive and therefore the search proceeds with the next extremes $\hat{i} = i+1$ and $\hat{j} = i + 2$. This search continues until the end of the data set is reached. The results of this procedure are depicted in figure 4.3a.

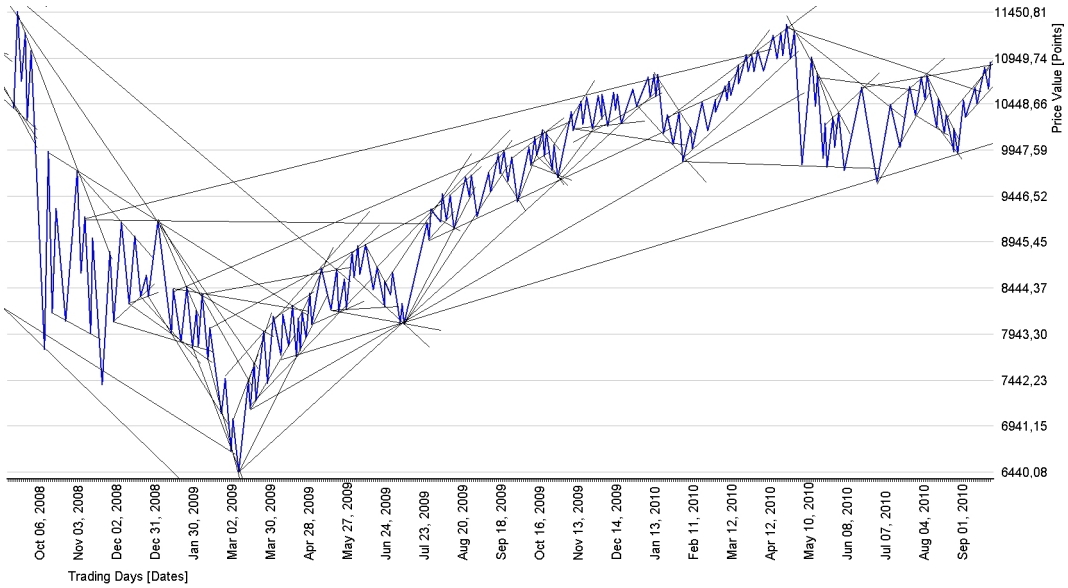
Merge of Similar Trend Lines

As it can be seen from figure 4.3a, the outcome of the previously described search procedure results in an exhaustive set of trend lines, but apparently there are large sets of lines with high similarities. Hence, the result contains many redundancies which would increase the complexity of the inference process unnecessarily. Cause of the many redundancies is the fact that adjoining local extremes tend to reside on very similar price levels. This leads to many trend lines that differ only in adjoining confirmation points and therefore exhibit almost exactly the same slope and length. To filter out such lines, the results from the previous section are searched for lines with high similarities, which are consequently merged into a new line (again, this is created through linear regression).

Natural similarity criterion for the comparison of two lines is the difference of their respective angles of elevation. From a given slope m (which is known after performing the linear regression), the corresponding elevation angle can be calculated by $\alpha = \arctan(m)$.



(a) Exhaustive set of trend lines



(b) Set of merged trend lines

Figure 4.3: Recent trend lines in the Dow Jones Industrial Average Index

parameter	value	meaning
V	0.005	relative tolerance for the distance between a particular extreme and a trend line
t_{max}	0.15	maximum spacing on the time axis between two points of a trend line (i.e., maximum duration without any confirmation) relative to the line's total length
δ	5°	maximum angle to consider two trend lines as similar

Table 4.1: Parameters for the line search

This formula is not applicable however in this scenario, as the slope is dependent on the current price level and therefore cannot be used as an ubiquitous input. To illustrate this, consider an imaginary diagonal through figure 4.3a. A visual inspection would suggest an angle of $\alpha \approx 30°$. As this figure depicts roughly 500 trading days in the range from 6440 to 11450 points, the slope is $m \approx \frac{5010}{500} = 10.02$ and therefore obtaining the angle calculatively would yield to $\alpha = \arctan(m) \approx 84°$. And if the figure would instead of the DJI depict a penny stock ranging from 0.664USD to 1.145USD, the same diagonal would have a calculative slope of 0.001002 and the resulting elevation angle would be $\alpha \approx 0.05°$. This problem is due to the fact that the lines depicted in the figure provide visually useful angles, but as the chart can be scaled arbitrarily in the vertical direction, the corresponding angles are rather arbitrary, too.

To resolve this problem, an additional slope normalizing constant N is introduced. It is calculated upon the average values of both lines l_1 and l_2 that are to be compared:

$$N = \frac{\frac{\max(l_1)-\min(l_1)}{2} + \frac{\max(l_2)-\min(l_2)}{2}}{2}$$

Consequently the elevation angle of a trend line is calculated as

$$\alpha = \arctan\left(\frac{m}{N}\right). \tag{4.3}$$

Based upon this, two lines are considered to be similar if their angle difference is below a certain threshold δ:

$$\alpha_1 - \alpha_2 \leq \delta \tag{4.4}$$

In order for two lines to be merged, there are additional obvious requirements: naturally, both merging candidates must overlap on the time axis and the distance between them must not exceed the tolerance threshold $VP(t)$, as defined before. If all of these conditions are met, all points from both lines form the input for another recursive linear regression process, as described above. As before, it may happen that regression does not yield a valid line and in this case both of the original lines are kept.

After applying this cleaning process to the trend lines depicted in figure 4.3a, the rather slender set of trend lines shown in figure 4.3b is obtained. These lines form the actual input for the probabilistic reasoning process. To summarize this procedure, table 4.1 lists all criteria required for the identification of trend lines together with respective values that yield feasible results in practice.

4.1.2 Domain Parameters

Evidence

As explained in section 2.2.1, the key properties of a trend line are its duration, the number of confirmations, and its elevation angle. Also, it is important to know how close current price movements are to a certain trend line, since this determines the current influence of a line (if current price movements are not within the proximity of a certain line, it is unlikely that this line influences price movements in the near future). Hence, each line l_i obtained from the methods described in the previous section is described with the following parameters for each day t:

- $l_i^d(t)$ duration of the line, i.e., the number of days from its starting point
- $l_i^c(t)$ number of confirmations, i.e., number of supporting points
- $l_i^a(t)$ the line's elevation angle
- $l_i^o(t)$ offset between current price and current line position. This value is scaled by current prices to obtain universally applicable values, i.e., $l_i^o = \frac{\text{line position - price position}}{\text{price position}}$

The trend lines are grouped into support lines s and resistance lines r so that the evidence for each day consists of two sets of lines. This leads to a problem in the design of an appropriate network since the number of lines present at a certain day will vary over time. Hence, the observation space has a variable size in this domain but the Bayesian network requires a fixed size. While there are certain variants of Bayesian networks that allow for a variable input space (e.g., *switching networks* or *segment models* as described in [Mur02]), these approaches come with an increased complexity in the model design as well as in the inference algorithms. Within the domain of trend lines, there is a much simpler solution to solve the problem of the variable input space: While building the input data set, it can be easily determined what the maximum number of co-occurring resistance lines and support lines is throughout the available data. Consequently, a fixed-size input space can be incorporated.

In addition to the information about trend lines, the observations are augmented with the percentaged change in the closing price in order to capture the current movement's direction. This price change is denoted by c and calculated as $c(t) = \frac{\text{closing price}(t) - \text{closing price}(t-1)}{\text{closing price}(t-1)}$.

With m denoting the maximum number of resistance lines and n denoting the maximum number of support lines, the complete evidence vector for any day is:

$$\vec{e}(t) = (r_1(t), ..., r_m(t), s_1(t), ..., s_n(t), c(t)) \tag{4.5}$$

Thereby, the evidence space is fixed to the size $m + n + 1$. For days with less than the maximum number of trend lines the evidence vector is simply filled with zeros denoting the absence of further lines. For instance, if the observations at same day t consist of $i < m$ resistance lines and $j < n$ support lines, the according evidence would be

$$\vec{e}(t) = (r_1(t), ..., r_i(t), 0, ..., 0, s_1(t), ..., s_j(t), 0, ..., 0, c(t)) \,.$$

Prognosis

Before introducing the hidden states that form the prognosis in this module, it is important to examine the nature of price movements. These movements reflect a rather noisy signal and may be subject to seemingly arbitrary outbursts on any day. Therefore aiming at the prediction of exact values for a certain day would be a futile task.

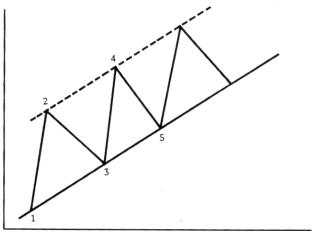


Figure 4.4: Schematic trend channel

As explained in section 2.3, moving averages can be used as an instrument for denoising the signal. The use of a short-term moving average will still track actual price movements closely while at the same time the influence of arbitrary fluctuations is highly reduced.

A second problem is the difficulty of forecasting the exact timing of any movements. If the system is trying to predict values for a certain day in the future, it may easily happen that the prediction itself has a high quality but the timing has not been estimated correctly. For instance, consider the situation in which the system is trying to obtain a five-day forecast and is inferring a strong upward movement. Now it may happen that the prognosis itself was correct and prices indeed start an ascent in the subsequent days, but after, say, three days some new information leads to a drop in the price values. Hence, after five days it could be that a strong upward movement indeed has happed as forecasted, only with its impact having already vanished at the day that was subject to the forecast.

Consequently, instead of inferring a prognosis about the state *after* a certain time period, the movements *within* a certain period should be employed for a prediction.

To further illustrate this, consider the situation of a trend channel (as described in section 2.2.1, depicted again in figure 4.4): the assumption that price movements in the near future will reside inside the channel's boundaries is a reasonable prediction. The fluctuations of price movements can be easily forecasted by projecting the boundary lines into the future. However, the identification of this channel does not provide any information about the movements within this channel and therefore it would be a hard task predicting whether prices are close to the upper or to the lower boundary line (or somewhere in between) at a certain time and it would not significantly enhance the forecast's merit, either.

Following these considerations, the hidden state in this model is formed by prognoses about the percentaged minimum and maximum changes of a short-term moving average within a certain time period p, i.e.,

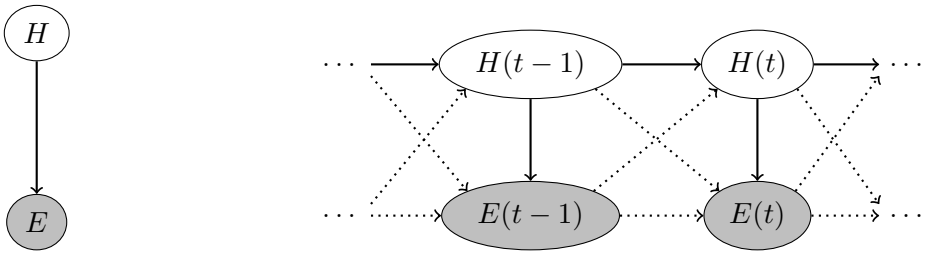
$$\vec{h}_p(t) = \left(\min_{x \in \{t+1,\ldots,t+p\}} \frac{WMA(x)}{WMA(t)}, \max_{x \in \{t+1,\ldots,t+p\}} \frac{WMA(x)}{WMA(t)} \right) \tag{4.6}$$

Since the evidence contains trend lines of varying duration, it can be used to infer predictions on different time scales (obviously, a short trend line is suitable for near-future prognoses, while a long trend line is better used for mid-term or long-term prognoses). Therefore, the hidden state is designed to infer a set of predictions in the form of equation 4.6 and consequently the complete hypothesis is formed by a set of n predictions (and thereby leads to hidden state space of size $2n$):

$$\vec{h}(t) = \left(\vec{h}_{p_1}(t), ..., \vec{h}_{p_n}(t) \right) \tag{4.7}$$

4.1.3 Prediction Network

For the construction of the Bayesian network it is assumed that separate trend lines occur independently of each other. While this assumption might be subject to an thorough



(a) Naive Bayesian intra-slice network for the trend line module

(b) Dynamic Bayesian network for the trend line module

Figure 4.5: Bayesian networks for the trend line module

investigation, practical tests proved it to be sufficient for the design of a feasible prognosis module. Assuming i.i.d. observations allows for the use of a naive Bayesian approach, as explained in the previous chapter. This leads to the slender naive Bayesian intra-slice network depicted in figure 4.5a.

Since the available data forms a successive time series of data sets, it is expedient to augment this network to a dynamic network in order to capture temporal dependencies. The adequate network is depicted in figure 4.5b. Due to the hidden states $H(t)$, this network is an instance of a Hidden Markov Model, as explained in the previous chapter.

For the forecasting domain it would be reasonable to assume that all nodes in the inter-slice network influence each other: if a forecast concerning movements for, say, the next week is obtained today, one should expect that a similar forecast will be obtained tomorrow. Consequently, today's forecast (or more precisely: the actual pending price movement subject to the forecast) should influence tomorrow's observations and today's observations should influence tomorrow's forecast. Also, since trend lines remain present for longer periods of time, the observations of consecutive days should have a strong connection. This would lead to a so-called *fully connected vector autoregressive process* as described in [Mur02] and is depicted through all (solid and dotted) arrows in figure 4.5b.

Experiments with different variants of the dynamic model have shown however, that most connections do not contribute to an appreciable enhancement of the forecasts' quality though they increase the network's complexity and therefore slow down the inference process significantly. Consequently, all dotted connections in figure 4.5b have been omitted since they do not yield better results compared to the network including only the solid connections.

Also, the considerations from section 3.3 suggest an alternative modeling of the domain: instead of encapsulating the forecasts into a hidden state and thereby employing a Hidden Markov Model, one could create a fully observable Bayesian network such that the evidence nodes still depict observations of trend lines while the hypothesis node would be replaced with information about the current change in price $C(t)$. The corresponding network is depicted in figure 4.6. Consequently, instead of inferring hidden states, one could use this network to extend the sequence of price changes into the future and, by determining the most likely sequence, one could obtain forecasts about future price movements, too. However, tests have shown that this alternative network performs significantly worse than the presented Hidden Markov Model and therefore was not employed. The same considerations also hold for the next forecasting module, which is why it has been modeled as a Hidden Markov Model, as well.

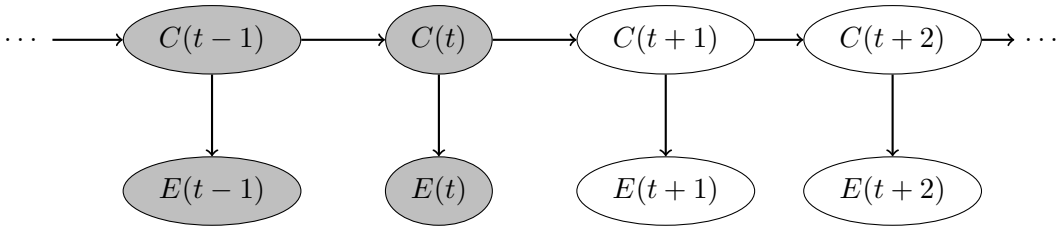


Figure 4.6: Alternative dynamic Bayesian network for the trend line domain

Preceding Considerations on the Choice of Distribution Functions

An analysis of movements in stock prices shows that the majority of short-term price changes exhibit only a small magnitude (thereby suggesting a large probability of small changes) and, vice versa, magnificent short-term price changes only have a small probability. Consequently the use of gaussian distributions seems to be the appropriate choice for the implementation of this network.

While gaussian distributions are indeed used within this work, some comments are necessary on this choice, before describing the distributions in detail.

Though there always has been some controversy, gaussian distributions have been the default choice in any kind of financial model for several decades. It has been criticized early (e.g., in [Man63]) that the features of the gaussian distribution (skewness and kurtosis) are inadequate for modeling the returns of financial assets. This criticism is mainly due to the fact that the kurtosis in a gaussian distribution leads to a bell curve shape that falls off too quickly (see figure 4.7 for an illustration). In other words, short-term price fluctuations of a high magnitude receive a very low probability and a true crash of prices becomes virtually impossible according to a gaussian distribution and hence the risk of losses is notoriously underestimated. As an alternative, "fat-tail" (or, more formally: leptokurtic) distributions as depicted in figure 4.7 have been suggested (for example by [Man63]). The figure illustrates the consequences of this alternative clearly: since high fluctuations are more probable according to the alternative distribution, a model constructed upon a fat-tail distribution should be more risk aware. While this might appear instantly desirable, a higher risk awareness has the "disadvantage" of yielding more careful prognoses and, since potential gain and potential risk are usually proportional, naturally limits the potential rewards of such a model. These considerations kept most financial engineers from employing fat-tail distributions for several decades.

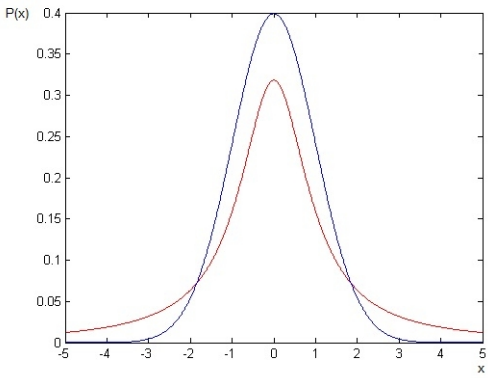


Figure 4.7: Comparison of distribution functions: a standard gaussian distribution (blue) and a leptokurtic distribution (red)

Popular opinion about the use of gaussian distributions in financial models only changed recently as a consequence of the US subprime mortgage crisis which eventually led to the recent world economic crisis. One of the main reasons for this crisis was a lack of risk awareness throughout major market participants. And as most of them used gaussian-based risk models, some even blame the gaussian distribution for causing this economic crisis (as explained for example in [Pat10]).

Despite these considerations, there are still reasons to design a forecasting module based on gaussian distributions. First, the models presented in this work do not aim at providing an accurate risk model but they rather aim at predicting probable movement directions. Additionally, as explained below, the inferred prognoses of this module only depend on the expected value and since both types of distribution yield the same expected value, the prognoses would be unaffected by employing a leptokurtic distribution. Also, as explained in [DVR07], a mixture of gaussian distributions can be used to alter the distribution's shape and thereby overcome the aforementioned obstacles.

To summarize, although the use of gaussian distributions is by now condemned in many financial models, they are still suitable for the setting of this work. Since the used Matlab toolbox facilitates only gaussian distributions (though it could be extended accordingly) and both kinds of distribution will yield the same expectation values, employing leptokurtic distribution functions would only require more implementation efforts without altering the results at all. Thus, all continuous distributions in this work have been modelled as gaussians.

Specification of Probability Distribution Functions

After deciding upon gaussian distribution functions, the probability density function for conditional gaussian (or normal) distribution that a certain hypothesis h_i emits a certain evidence value e_j is defined as[3]

$$P(e_i|h_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{-\frac{(e_i-\mu_{ij})^2}{2\sigma_{ij}^2}} \tag{4.8}$$

with unknown parameters mean μ_{ij} and variance σ^2. This gaussian (or normal) distribution is usually denoted by

$$P(e_i|h_j) \sim \mathcal{N}(\mu_{ij}, \sigma_{ij}^2) \tag{4.9}$$

Since all evidence variables are assumed to be conditionally independent, the joint probability distribution for all evidence variables can be expressed as (see equation 3.5)

$$P(\vec{e}|h_j) = \prod_i \mathcal{N}(\mu_{ij}, \sigma_{ij}^2) \tag{4.10}$$

with unknown parameters μ_{ij} and σ_{ij}^2.

Similarly, the dynamic state transitions are specified as

$$P(h_i(t)|h_i(t-1)) \sim \mathcal{N}(\mu_{it}, \sigma_{it}^2) \tag{4.11}$$

[3] As explained in [JL95], equation 4.8 is not strictly correct: the probability that a real-valued variable exactly equals any value is zero. Instead one can provide the probability that a variable x lies within some interval: $p(x \leq X \leq x+\Delta) = \int_x^{x+\Delta} \mathcal{N}(\mu, \sigma^2) dx$. By the definition of a derivative, $\lim_{\Delta \to 0} \frac{p(x \leq X \leq x+\Delta)}{\Delta} = \mathcal{N}(\mu, \sigma^2)$. Thus for some small constant Δ, $p(X = x) \approx \mathcal{N}(\mu, \sigma^2) \cdot \Delta$. The factor Δ is canceled out by performing normalization so equation 4.8 may be used.

4.1.4 Learning and Results

After the network has been fully specified, the unknown parameters (means and variances) can be learned through the ML learning procedure described in the previous chapter. The learning process results in a fixed set of variances, while the expected values depend on the respective observation. Hence, the expected values give information about the most likely values of the hidden states and thereby form the actual prognosis. Note that this explains the aforementioned irrelevance of the shape of the distribution's tail: both distributions depicted in figure 4.7 exhibit the same expected value (namely 0 in this figure) and since the general shape of the distribution is fixed after learning (due to the fixed variance values) and thereby cannot provide additional information about the forecast's certainty, some leptokurtic distribution would yield exactly the same results as a normal distribution in this context.

For the inference process of obtaining a forecast for d days, each data instance comprises data for $d+1$ successive days, i.e., the network is unrolled so that it consists of the $d+1$ time slices $t-d \ldots t$. Hence, at day t one can always determine the correct forecast from day $t-d$ in retrospective. This has the advantage that the "hidden" state of the instance's first day is known and therefore can be provided for inference purposes. This strongly enhances the inference of the time series' most likely sequence of states as it can be performed with respect to one fixed value.

Standard learning procedure is to divide the available data into a training and a test set. For creation of the training set, all available information from the corresponding data is specified and therefore in this scenario it consists of the daily information about trend lines $e(t)$ as well as the subsequent developments of prices $h(t)$. While these subsequent developments are unknown for current observations, they can be easily obtained by using historical data for the network training. This training set is fed into the network and as a result one receives estimates for all unknown parameters. After all parameters have been learned, the remainder of the available data is used as a test set. In this case, only the evidence is provided while the information about future developments is intentionally left out. Therefore the network has to infer states of the hidden nodes (or make predictions) based upon the previously learned relationships. After the whole test set has been processed, one then can compare the predictions to the actual developments that have been held back in order to evaluate the quality of the network's prognoses.

Due to the rather large set of available training data the learning procedure had to be altered a little in this work: instead of using a classical division into two separate sets, training and evaluation are performed similar to a sliding window technique. Initially the network has been trained with the first 3000 datasets to obtain estimates of the unknown parameters. Subsequently, these parameters have been tested with the following 1000 datasets (the "test window"). Upon completing the test procedure, these 1000 datasets have been added to the training data (this time containing the actual future developments) and parameters are re-estimated for this new, larger dataset. These re-estimated parameters are used for testing the next 1000 data sets. This process continues until the end of the dataset has been reached. Consequently, instead of having a fixed test set, the test set resembles a window that slides through the available data.

The forecasting results of this module can be best described through a visualization, as depicted in figure 4.8. While it is of no surprise that the forecasts are not able to match the actual developments exactly, it can bee seen that the predicted movements resemble a feasible approximation of true price changes. Some interesting facts about the results can be derived from this visualization: obviously there are market phases (e.g., the datasets $\approx 500, ..., 600$) in which this module is able to track the actual movements rather closely,

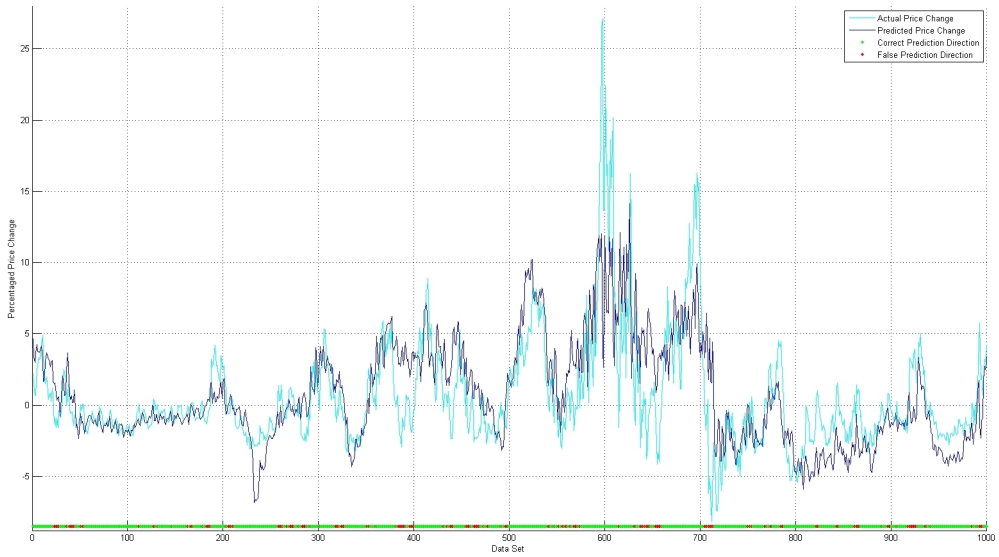


Figure 4.8: Evaluation of the line forecasting module: the graph shows a comparison of a 5-day forecast and the corresponding actual price changes in the Dow Jones Industrial Average Index for the 1000 most recent instances of the test data.

while there are other phases (e.g., the following $\approx 600, ..., 700$ datasets) where there are rather large discrepancies between forecast and actual development, mainly due to heavy fluctuations in the price movements. In general, it can be said that this module tends to underestimate the magnitude of any outbursts.

Though it is neat to obtain information on the exact magnitude of a price change, this information should not be overrated. The most important information to decide upon an action is the expected direction of future price movements. As long as the predicted direction turns out to be correct, any actions based upon this prediction are guaranteed to yield a gain, while an additional incorrect estimation of the movement's magnitude can only boost or reduce this gain. As mentioned before, this module usually tends to underestimate the magnitude of any outburst which means that trading actions based upon these forecasts tend to yield higher gains than predicted, if the magnitudes of forecast and actual movement differ. This should be an acceptable risk.

Consequently, the predicted directions should be treated as the essence of this module's forecast while the information about magnitude should be treated as a supplement that may potentially refine any trading strategies. The correctness of each prognosis' direction is marked separately in the bottom of figure 4.8 with colored dots. This shows that the vast majority of forecasts is able to capture the direction correctly and thereby provides highly useful results.

4.2 Point Patterns

As described in section 2.2.2, point patterns consist of a sequence of consecutive local extremes with a certain relationship to each other. The definition of consecutive extremes is somewhat arbitrary since the patterns' size may vary on the time scale (as it could be seen in figure 2.5). While a short-term pattern may be formed through truly consecutive extremes and thereby could be identified within the extreme point series described in the

previous section, there may be also patterns on a larger time scale. Obviously the key points for such patterns are formed by more significant local extremes and consequently between these significant points there may be several other extremes which have to be treated as noise. To identify such large-scale patterns, a filtering method for the local extreme series is required. Two separate approaches have been investigated within this work, both of which are explained in the following.

4.2.1 Scalable Extreme Point Models

Higher-Order Extreme Point Model

The first method of identifying extremes on a larger scale is an extension to the previously described method of finding local extremes. To illustrate this procedure, figure 4.9 depicts all intermediate results occurring during this procedure.

Initial input of this procedure is the series of local extremes resulting from the aforementioned method (i.e., a set of lowest-order extremes, depicted in orange). Based on this initial set, two new lines are created by each connecting all local lows and all local highs (depicted as the red lines). These lines represent boundaries of the price movements and already visualize price movements on a larger scale.

Next, both of this lines are treated as separated time series of datasets and are both searched for local extremes through the usual procedure. As a result, two new lines (depicted in black) are created which resemble a kind of filtered channel for local extremes. Since each of these lines is constructed upon local extremes, the resulting lines' extreme points represent extremes on a higher order. It can be seen that both lines generally tend to move into the same direction (if small fluctuations are considered as noise) and therefore turning points of this channel can be considered as the resulting extremes of higher order.

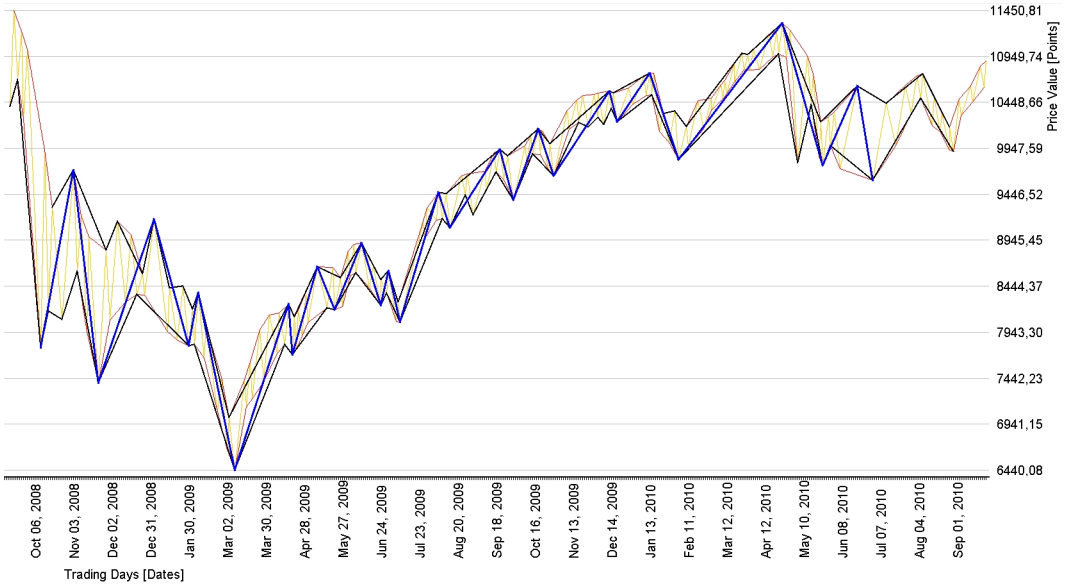


Figure 4.9: Identification of higher-order extremes: original series of extremes (orange), connection of local lows and local highs(red), extreme channel (black), and higher extreme series (blue). Note that some lines have been left out intentionally on the figure's edges in order to allow for an easier identification of the separate lines.

These turning points are identified through the following criteria: For the identification of higher-order lows, the low-points of the lower line are considered. For each low-point, the interval between the neighboring high-points is analyzed. If the higher line also exhibits a low-point within this interval, it is evident that both lines and therefore the whole channel has changed its direction. Therefore, this low is added as a new higher-order low. Accordingly, higher-order highs are obtained by considering high-points of the higher line. If in the interval of its adjacent low-points the lower line also exhibits a high-point, a new higher-order high has been found. The resulting series of extremes is depicted as the blue line in figure 4.9.

This procedure can be performed recursively in order to continue filtering the extremes. Advantage of this process is that, due to the use of a channel's turning points, each recursion returns only truly significant extremes of the next-higher order. However, this also introduces a drawback: since this filtering method does not enable a stepless scaling of the results, it may happen that certain patterns remain concealed because their key points are located somewhere in between two recursions.

Critical Point Model

An alternative approach for the identification of significant extremes is the *Critical Point Model (CPM)* proposed by [Bao08]. The same model has also been suggested by [PWZP00] under the name *Landmark Model*.

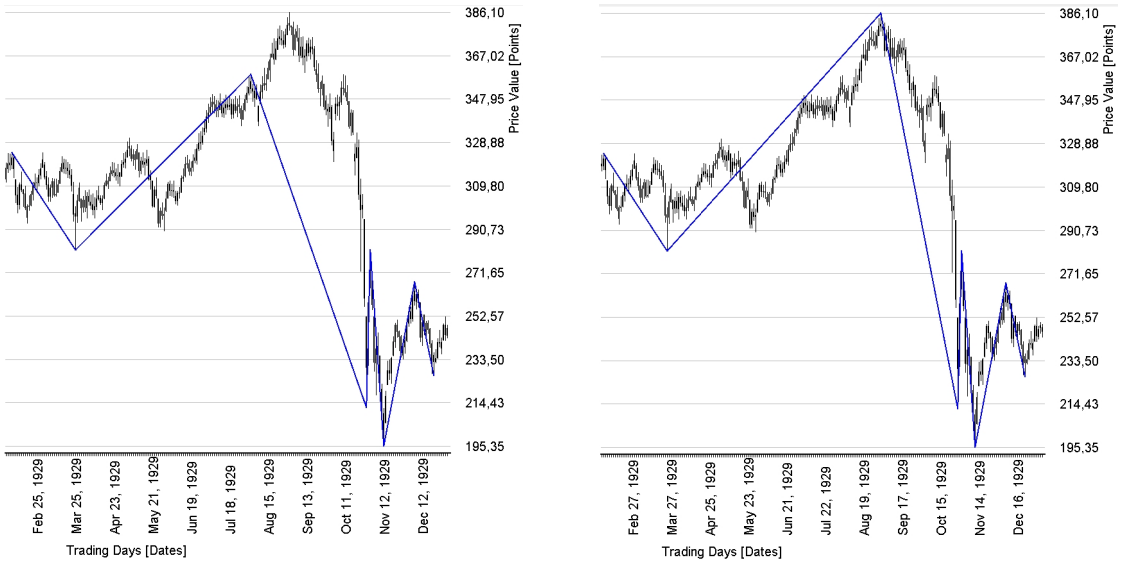
Cornerstone of this model is the specification of a duration threshold T_D and an oscillation threshold T_O. Initial input for this procedure is also an alternating series of local extremes. Informally speaking, the filtering process is performed by analyzing the distances of successive extreme points and, if oscillation or duration between these points is below the specified thresholds, removing points from the time series until all remaining distances exceed the specified thresholds.

More formally, the time series is analyzed by iterating through units of each local maximum point i_2 and the adjacent local minimum points i_1 and i_3. Hence, each unit consists of a rise (from i_1 to i_2) and a following decline (from i_2 to i_3). These units can be divided into four different cases with regard to the thresholds T_D and T_O:

1. both the rise and decline exceed at least one of the thresholds
2. the rise exceeds a threshold but the decline is below T_D and T_O
3. the rise is below T_D and T_O but the decline exceeds a threshold
4. neither the rise nor the decline exceed any threshold

With $P(i)$ denoting the price value of a point i, these cases are are processed as follows (as described in [Bao08]):

1. As both rise and decline exceed the thresholds, i_1 and i_2 are preserved as critical points and the next unit is formed by the points i_3, i_4, and i_5. This is the only case that actually leads to the preservation of points.
2. This case can be divided into two sub cases depending on the following point i_4: if $P(i_4) \geq P(i_2)$, the next unit is formed by i_1, i_4, and i_5, while i_2 and i_3 are discarded as irrelevant. Else, if $P(i_4) < P(i_2)$, the next unit is formed by i_1, i_2, and i_4.
3. The points i_1 and i_2 are discarded as irrelevant and the next unit is formed by i_3, i_4, and i_5.
4. This case will be divided into two sub cases again: if $P(i_1) \leq P(i_3)$ the next unit is formed by i, i_4, and $i5$, else it is formed by i_3, i_4, and i_5.



(a) CPM strictly according to [Bao08]

(b) CPM with additional postprocessing

Figure 4.10: Critical Point Model

An implementation of these methods has shown that they do not yield the expected results in all cases but lead to the identification of insignificant extreme points in rare cases. This is due to the fact that a critical point is preserved as soon as the thresholds are exceeded and subsequently search continues from this point on. It may happen that subsequent movements exhibit another extreme that is more significant than the one previously found, but as the distance to the preceding critical point is not exceeded, this extreme is discarded. This problem is illustrated in figure 4.10a: after finding the local maximum (at August 5), the remaining ascend does not exceed the thresholds and is therefore discarded while the figure clearly shows that the ascend's summit (September 3) constitutes a more significant local maximum. This problem can be easily fixed by introducing an additional post-processing step: after the critical point model has been obtained, the resulting extreme series is compared to the underlying price movements. The comparison proceeds in intervals of three adjacent extremes $i_1 ... i_3$ (i.e., either a local maximum i_2 with its neighboring local minimal i_1 and i_3 or vice versa). If the underlying price movements exceed the center extreme within the given interval, the corresponding critical point i_2 is simply shifted to the actual extreme of price movements within the given interval. For instance, the previously described case in figure 4.10a leads to the analysis of the interval bounded by the neighboring local minima (March 26 and October 29). Analysis of the price movements shows that the previously found critical point is exceeded and therefore its position is shifted to the actual maximum within this interval, as depicted in figure 4.10b.

Pros and cons of this procedure are exactly opposite to those of the previously described method: the use of arbitrary thresholds allows for a stepless scaling of the model. However, there may be rare instances in which a certain choice of thresholds leads to subsets of points in which some are considered as significant according to the first model while others are considered as noise. An increase of the threshold may result in a concurrent removal of both the significant and insignificant points. As a result, it may happen (though only rarely) that the CPM misses certain patterns that would have been identified through the use of the higher-order extreme model.

Consequently, both models should be used simultaneously to query the time series for

the existence of point patterns. In order to form the input to the pattern search, both methods are applied to the data repeatedly (the CPM model with increasing thresholds) until they no longer return any extreme points.

4.2.2 Identification of Point Patterns

Once the extreme point models have been constructed, they can easily be searched for point patterns according to the following criteria:

- *Head Shoulders*
  A Head Shoulders pattern consists of seven consecutive points $i_1 ... i_7$:
  - i_1: preceding minimum
  - i_2: left shoulder (local maximum)
  - i_3: left neck (local minimum)
  - i_4: head (local maximum)
  - i_5: right neck (local minimum)
  - i_6: right shoulder (local maximum)
  - i_7: trailing minimum

  To form a valid pattern, the values $P(i)$ of these points must fulfill the following conditions:
  - $P(i_4) = \max\limits_{k \in \{1...7\}} P(i_k)$, i.e., the head must be the highest point of the pattern
  - $\min\big(P(i_2), P(i_6)\big) > \max\big(P(i_2), P(i_5)\big)$, i.e., both shoulders must be higher than the necks
  - $\max\big(P(i_1), P(i_7)\big) < \min\big(P(i_3), P(i_5)\big)$, i.e., both values preceding and trailing the pattern must be lower than the necks

  The Inverse Head Shoulders pattern can be identified accordingly by inversing all conditions.
- *Double Top*
  A Double Top pattern consists of five consecutive points $i_1 ... i_5$ with i_2 and i_4 being local maximum points. The condition to form a valid pattern is simply $P(i_2) \approx P(i_4)$, i.e., both tops must reside on the same price level (with regard to a certain tolerance threshold).
  Inversing these properties leads to the identification of a Double Bottom.

As explained in section 2.2.2, these patterns hint at trend reversals. Therefore their identification should be limited to situations in which they are preceded by the corresponding trend movement. For instance, it is not feasible to analyze a Head Shoulders pattern (which indicates a reversal from an uptrend to a downtrend) if price movements do not exhibit an uptrend in the first place. Consequently, as explained in section 2.3, a moving average is used to determine the preceding trend state and patterns are only identified if a pattern matches the corresponding trend.

It should be noted that the presented criteria for each pattern are rather relaxed. Technical analysts usually impose further constraints on a valid pattern, namely they require certain relations for the patterns' individual properties. For instance, for a valid Head Shoulders pattern it is usually required that both shoulders and both necks reside on similar price levels, the distance between head and shoulders must exceed a certain threshold, and the whole pattern should exhibit a rather symmetric shape. An integration of these

Security	Time Period (years)	# Head Shoulders	# Inverse Head Shoulders	# Double Tops	# Double Bottoms
Dow Jones Industrial Average Index	≈ 82	35 (≈ 1/2.3 yrs)	16 (≈ 1/5.1 yrs)	38 (≈ 1/2.1 yrs)	12 (≈ 1/6.8 yrs)
Standard & Poor's 500 Index	≈ 60	18 (≈ 1/3.3 yrs)	11 (≈ 1/5.5 yrs)	16 (≈ 1/3.7 yrs)	6 (≈ 1/10 yrs)
Nasdaq Composite Index	≈ 40	2 (≈ 1/20 yrs)	1 (≈ 1/40 yrs)	2 (≈ 1/20 yrs)	12 (≈ 1/3.3 yrs)
General Electric Company	≈ 48	17 (≈ 1/2.8 yrs)	18 (≈ 1/2.7 yrs)	28 (≈ 1/1.7 yrs)	8 (≈ 1/6 yrs)
Amazon.com Inc.	≈ 13	9 (≈ 1/1.4 yrs)	9 (≈ 1/1.4 yrs)	15 (≈ 1/0.8 yrs)	9 (≈ 1/1.4 yrs)
International Business Machines Corp.	≈ 48	16 (≈ 1/3 yrs)	14 (≈ 1/3.4 yrs)	17 (≈ 1/2.7 yrs)	16 (≈ 1/3 yrs)
The Coca-Cola Company	≈ 48	19 (≈ 1/2.5 yrs)	14 (≈ 1/3.4 yrs)	24 (≈ 1/2 yrs)	11 (≈ 1/4.4 yrs)

Table 4.2: Point pattern occurrences in selected securities: the table lists the number of identified patterns for each security and their respective average occurrence frequencies. The analyzed time periods depend on the available historical data from Yahoo Finance.

additional constraints would lead to a significantly reduced set of resulting patterns.

Table 4.2 lists the results of the pattern search for a selection of popular securities. The selection is rather arbitrary but the results provide a good representation of the patterns' quantities for all securities that have been investigated within this work.

An analysis of these results reveals a severe problem that hinders any feasible approach of statistical inference: the resulting set of patterns is simply too sparse to enable the learning of any meaningful relations by examples. It should be noted that a manual analysis of the price charts did not result in the identification of any further patterns and hence it can be precluded that insufficiencies in the extreme point models are the cause for this problem and therefore the resulting sets of patterns can be assumed to be exhaustive.

At first glance, it appears that these deficiencies might be resolved by merging patterns of several securities into a single pattern database until enough pattern instances have been collected to allow for an expedient statistical learning approach. However, an analysis of the individual patterns and their respective future development reveals another key property that is important to all learning processes presented in this work: different securities react differently when following the presence of a certain pattern. For instance, an analysis of subsequent developments after the occurrence of Head Shoulders patterns in the Dow Jones Industrial Average Index reveals that this pattern yields a highly reliable signal since it is followed by a significant downward movement with nearly every occurrence. The same analysis performed on Amazon.com Inc. reaches opposite conclusions. Even though the occurrence frequency is significantly higher, subsequent price developments do not seem to exhibit any clear relation to the occurrence of this pattern and therefore it does not provide any feasible signal for this security.

Hence, although the methods presented in this work are universally applicable to any security, it is of evident importance to realize that each security has its own characteristics

and consequently the reasoning modules have to be trained separately for each security. This realization prohibits the approach of merging patterns from various securities into a single pattern database.

These considerations give rise to the question of the true cause which leads to the detected point pattern deficiencies: the analysis so far has only shown that the available data did not exhibit a sufficient number of well-known significant patterns. It remains unclear however, whether this is due to a sparse *existence* of significant patterns or due to a sparse *knowledge* of such patterns. In principle it might very well be that the data contains a various number of characteristic patterns that are simply unknown (or at least unpublished). In order to search the data for potentially unknown pointpatterns, the resulting local extreme points were organized in hierarchical clusters (this method will be explained in more detail in section 4.4.2) and the resulting clusters were analyzed with respect to common successive developments. These clusters did not reveal any additional patterns and therefore did not provide a meaningful alternative to the manual definition of patterns.

As a consequence, no feasible forecasting module based on point patterns could be constructed with the available data. And even if one would construct a meaningful forecasting module somehow, it would hardly increase the system's overall results since these patterns occur so rarely that they could contribute to the overall analysis results only very seldomly. There is a possibility that the situation is different, if, instead of analyzing daily data, one would choose a smaller time unit (e.g., hours). By analyzing the same time period with a finer time resolution, one should expect the occurrence of significantly more patterns simply due to the larger dataset. This could not be tested due to the lack of historic intra-day data.

It should be noted that several works (e.g., [Bao08], [BY08], [GLL07]) have been published which report on a successful implementation of technical analysis systems incorporating point patterns. However, due to only vague summaries of the testing scenarios, their results could not be reproduced. Also, these papers describe systems incorporating a variety of analysis methods so that it remains unclear how much the point patterns contribute to the overall results (provided that they do contribute to the results at all).

4.3 Technical Indicators: Forecasting Module 2

The next forecasting module is another dynamic Bayesian network which aims at facilitating predictions of future price movements by analyzing the states of the set of technical indicators described in section 2.3.

4.3.1 Domain Parameters

Evidence

Since an implementation of the described indicators directly yields a set of numerical values, these values can easily be used as input for a Bayesian network. However, it has to be considered that many of these values directly relate to the securities' state (i.e., they exhibit absolute values which are informative only when comparing them to the underlying values of the security's current price or volume). Since the prognosis should be universally applicable, such values need to be normalized in order to hinder the forecasting module to infer relations to absolute price levels. For instance, if the observed indicator values suggest a pending ascent, this insight should be retrieved no matter if the underlying security currently resides on high or low levels compared to its historic movements. To

attain this independence of absolute price levels, some of the indicators are normalized before being fed into the Bayesian network. The actual input values for each indicator are listed in table 4.3. Note that not all indicators are connected to absolute levels, but there are also several indicators which have a restricted range in the first place. These indicators can be incorporated directly without any preprocessing.

The analyzed evidence space of the indicator network comprises 19 numerical values, as denoted in the table. As it will be explained below, some of the indicators proved to be of no avail for the inference of a forecast and therefore have been removed from the eventual network. These indicators are marked accordingly in the table.

Prognosis

The same considerations described in section 4.1.2 hold for the modeling of an appropriate forecasting state in the indicator domain. Hence, the forecast is subject to price movements within a certain interval of future developments again.

However, as investigations of various modelings of the forecasting state have revealed, opposed to the trend line module the indicator domain does not facilitate the means of actually quantifying the magnitude of future movements. All attempts of predicting the amount of change resulted in forecasts which exhibited hardly any recognizable relation to the factual developments. Consequently, the indicator module is not commissioned with quantifications of changes but is rather content with predicting the movement's direction. As it was explained in section 4.1.4, the essence of any feasible forecast is reliable information of future directions and therefore this module is still able to provide highly valuable information, even though it is restricted to discrete prognoses of directions.

After examining several variations of modeling the forecast, the method which proved to perform best with respect to reliability and usefulness is the model of determining whether future developments exhibit significant movements into a particular direction within a certain time interval. To identify a significant movement, a price threshold T_P is defined and a percentaged change in price is considered significant if it exceeds this threshold. Again, due to the same reasons explained in section 4.1.2, a moving average is used as a basis to determine forecasts. Hence, the discrete hypothesis for a forecasting period p is defined as

$$h_p(t) = \begin{cases} 1 & \text{if} \quad \min\limits_{x\in\{t+1,\ldots,t+p\}} \dfrac{WMA(x)}{WMA(t)} > -T_P \\ & \quad \wedge \max\limits_{x\in\{t+1,\ldots,t+p\}} \dfrac{WMA(x)}{WMA(t)} > T_P \\ -1 & \text{if} \quad \max\limits_{x\in\{t+1,\ldots,t+p\}} \dfrac{WMA(x)}{WMA(t)} < T_P \\ & \quad \wedge \min\limits_{x\in\{t+1,\ldots,t+p\}} \dfrac{WMA(x)}{WMA(t)} < -T_P \\ 0 & \text{else} \end{cases} \tag{4.12}$$

While the previously described trend line module contained evidence for predictions on various time scales simultaneously and therefore could be used to facilitate several

Indicator	Components	Network Input	Size	
Price Change	• difference between current and previous closing prices (P)	$\frac{P(t) - P(t-1)}{P(t)}$	1	
Exponential Moving Average	• difference between indicator value (EMA) and current price (P)	$\frac{P(t) - EMA(t)}{P(t)}$	1	
Cross Average	• difference between indicator value (CAV) and current price (P)	$\frac{P(t) - CAV(t)}{P(t)}$	1	
Moving Average Conver-gence/Divergence	• MACD-Line ($MACD$) • Signal-Line (MAS)	$MACD(t), MAS(t)$	2	
Bollinger Band	• bandwidth (BW) • distance between current price and band center (BC)	$\frac{BW(t)}{P(t)}, \frac{P(t) - BC(t)}{BC(t)}$	2	
Momentum	• indicator value (MOM)	$MOM(t)$	1	
Relative Strength Index	• indicator value (RSI)	$RSI(t)$	1	
Average True Range	• indicator value (ATR)	$\frac{ATR(t)}{P(t)}$	1	
Aroon Indicator	• Aroon Up (AU) • Aroon Down (AD)	$AU(t),\ AD(t)$	2	
Commodity Channel Index	• indicator value (CCI)	$CCI(t)$	1	
Stochastic %K %D	• %K-line (SK) • %D-Line(SD)	$SK(t),\ SD(t)$	2	
Volume	• difference between value (VOL) and corresponding volume moving average (VMA)	$\frac{VOL(t) - VMA(t)}{VOL(t)}$	1	$\star$
On-Balance Volume	• indicator value (OBV) scaled by volume moving average (VMA)	$\frac{OBC(t)}{VOL(t)}$	1	$\star$
Money Flow Index	• indicator value (MFI)	$MFI(t)$	1	$\star$
Ease of Movement	• indicator value(EOM), scaled to ease processing	$EOM(t) \cdot 10^5$	1	
total			19	

Table 4.3: Indicator preparation for the use with a Bayesian network: the table lists the components for each indicator, the corresponding calculations and the input size (i.e., number of values) for each indicator. Indicators which have been omitted in the final network are marked with a $\star$.

prognoses in parallel, the indicator domain exhibits a definite relationship between the obtained evidence and its viability for a particular forecasting period. As it was explained in section 2.3, all indicators are parametrized with respect to a certain duration, which has a direct influence on the forecasting time scale. Hence, instead of incorporating multiple hypothesis values, the model is designed to deliver predictions only for one particular time scale. Then predictions of alternative time scales can be obtained by employing several instances of the same model with varying evidence duration parameters. Experiments have shown that a ratio of $\frac{1}{3}$ between forecasting period and indicator duration yields the best result. For instance, to obtain a five-day forecast, the indicators are calculated for a duration of 15 days.

4.3.2 Prediction Network

The most simple design of incorporating the technical indicators into a Bayesian network is again the naive Bayesian approach as depicted in figure 4.11.

Successful implementations of such a naive Bayesian approach for this domain have been reported in [BY08] (treating daily observations separately) and [Leo06] (using a dynamic Bayesian network with the same structure depicted in figure 4.11). While these proposals aim at the same target, both of them differ in the pursued strategy of representing the input state space. Common to both approaches is that they infer discrete directional predictions and discretize the observations before they are fed into the prediction network.

[BY08] uses static rules for each indicator in order to obtain a trading signal, i.e., each observation component is classified into the binary states *up* or *down*. Hence, the actual input to the Bayesian network is formed by a boolean vector. While it is reported that this approach yields feasible results, it comes with two major disadvantages:
First, the transformation of continuous indicator values into boolean variables comes with a significant loss of information. As it was explained in section 2.3, most indicators provide considerably more information than just a simple discrete state, namely because their exact values usually hint at the strength of a certain signal. Also, since some of the most valuable information originates from (possibly small) divergences of different indicators, one of the most powerful capabilities of technical indicators is waived by compressing the indicators into boolean values, because small divergences cannot be captured but instead a comparison of different indicators can only lead to the results "same" or "different".
Second, the use of static rules to classify indicator values into boolean trading signals requires sound domain expert knowledge (i.e., a wide experience in the application of technical indicators). Signaling rules for each indicator have to be designed and implemented separately and any shortcomings in these rules will lead to inevitable deficiencies in the resulting probabilistic network. Consequently, instead of delegating the learning task to an adaptive reasoning module, this approach relies on human expertise while the probabilistic reasoning system merely has the potential of determining the human expert's reliability.

[Leo06] aims at identifying all significant states that a particular indicator may adopt.

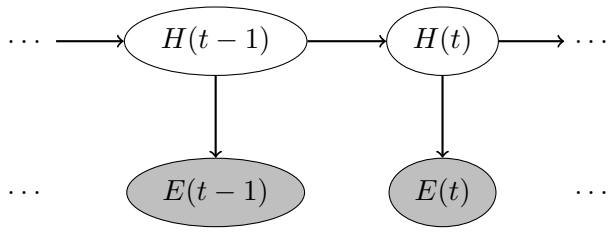


Figure 4.11: Bayesian network for the indicator module

In order to define these states, a set of transformation rules is defined for each indicator. For instance, values of the Relative Strength index are categorized into "below oversold threshold", "below 50 but higher than oversold threshold", "above 50 but below overbought threshold", and "above overbought threshold". All significant states of the RSI are then obtained by combination of current and previous category of the indicator and thereby a total of 16 significant states is identified only for this indicator (though it remains unclear why the indicator states are defined with respect to previous values if a dynamic Bayesian network is used, which is fed with observations of consecutive days anyways). Compared to [BY08], this approach has the advantage of the actual interpretation being left to the Bayesian network instead of a human expert and therefore this approach allows for a consideration of different indicator combinations instead of only isolated indicators. This approach still comes with disadvantages: In order to distinguish significant states of any indicator, it is still required to consult a domain expert with wide experience in the application of technical indicators. Still, a false or incomplete identification of significant states will introduce deficiencies in the resulting forecast, although the effect is less severe than in the previously described approach. The approach of using transformation rules for several indicators leads to a rule set of tremendous size which has to be maintained manually and expanded with the addition of each further indicator. Also, the discrete state space still does not facilitate the means of exploiting small divergences of different indicators.

Consequently, the approach pursued within this work does not employ discrete observation states but instead the continuous values listed in table 4.3 are directly fed into the network. While this may lead to a slower inference process and require more training data, the benefits are evident: since no preceding discretization step is employed, the module is completely independent of any human knowledge (which naturally always bears the risk of being subjective), but instead all potentially significant relations have to be inferred through the machine's probabilistic reasoning procedures. Next to enabling the chance of profiting from observed small indicator divergencies, this approach also has the advantage that no manual implementation of complex preprocessing has to be maintained.

A comparison of the indicator definitions from section 2.3 shows that several of them share high similarities. These similarities give rise to serious doubts about the naive Bayesian assumption of the conditional independence of the evidence variables. Also, it is questionable whether such a simple model is capable of exploiting the full potential of the available information. Some alternative models have been explored within this work, namely a combination of a *Coupled Hidden Markov Model (CMM)* and a *Hierarchical Hidden Markov Model (HHMM)*, as well as a *Segment Model* next to the *Vector Autoregressive Model* already described in section 4.1.3.

A Coupled Hidden Markov model is a dynamic Bayesian network with multiple sequences of evidence variables and multiple sequences of hidden states (as depicted in figure 4.12). Each hidden state exhibits a corresponding observation, but it is additionally assumed that hidden states also influence the successive states of other hidden nodes. In the domain of technical indicators, this model can be implemented such that each indicator yields a separate forecast but it is assumed that all individual forecast states impact other successive forecasts due to the assumption that the indicators are conditionally dependent. To obtain a general forecast from the whole set of observations, this model can be expanded to a Hierarchical Hidden Markov Model. Such a model exhibits a hierarchical structure of hidden states, i.e., there is one (or more) superior hidden state which is linked to several hidden substates actually emitting observations. This can be used to facilitate an all-embracing forecast, as defined in the previous section, built on individual forecasts created from each indicator observation. The resulting model is depicted in figure 4.13. Note that

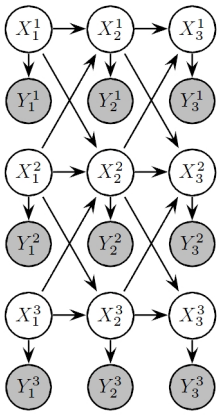


Figure 4.12: Schematic depiction of a Coupled Hidden Markov Model (from [Mur02]), hidden states are denoted by X and observed states by Y.

the general form of this network offers opportunities for many variations in the modeling of dependencies. For instance, one could also introduce connections between successive observations or a coupling structure between evidence nodes, as well.

Another alternative is the use of a Segment Model as depicted in figure 4.14. Main purpose of this model is the sectioning of the hidden state series into subsequences that share certain properties. This is done by introducing hierarchical states of random length which emit a sequence of observations. The number of hierarchical states is governed by a random variable, too. In the indicator domain, this model could be used to separate the hidden states into sequences of upward and downward movements, respectively. This has the advantage that - next to the particular relationships between indicator states and pending developments - the network is also able to infer knowledge about the duration of movements and therefore it has another dimension of information at its disposal, which can supposedly contribute to an enhanced quality of the forecasts. Also, by replacing the evidence nodes with the previously described superior hierarchical hidden states of a Hierarchical Coupled Hidden Markov Model, this Segment Model could be refined further.

For a thorough description of these models and further alternatives as well as suitable inference algorithms see [Mur02].

Although the described alternatives provide a more sophisticated approach of modeling the domain's correlations, practical tests have shown that an application of these models to the technical indicator domain did not yield any benefits. In fact, the results were quite the contrary: while the quality of the results was not significantly altered, the computing time was subject to a tremendous increase due to the considerably higher complexity of the alternative models.

Result of these investigations is that none of the examined alternatives is able to defeat the forecasts obtained from the naive dynamic Bayesian network, while at the same time any other model slows down the inference process significantly. Hence, the naive model (depicted in figure 4.11) has been chosen to implement this forecasting module.

In fact, it is a well-known property of naive Bayesian approaches that - even though for many domains their use deliberately introduces false independence assumptions - they often perform surprisingly well in practice. This phenomena has been subject to extensive research and could be confirmed through several empirical studies. More information on the evaluation of naive Bayes approaches can be found, for instance, in [DP97], [Ris01], and [HY01].

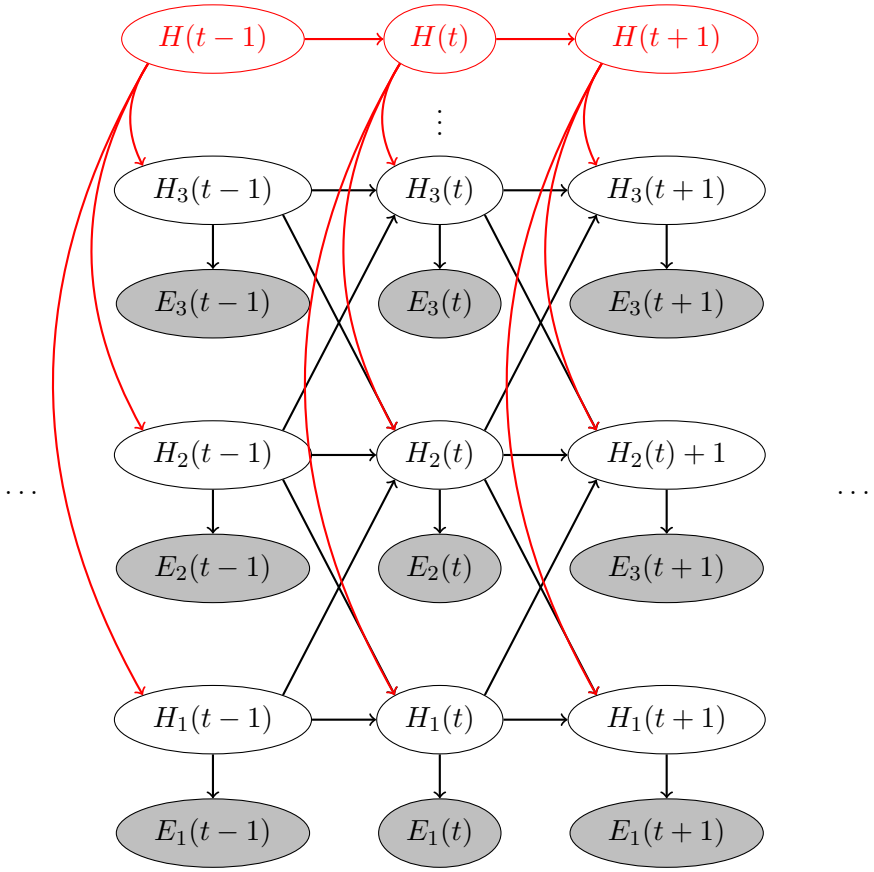


Figure 4.13: Hierarchical Coupled Hidden Markov Model for the indicator domain, the extension with hierarchical states is depicted in red.

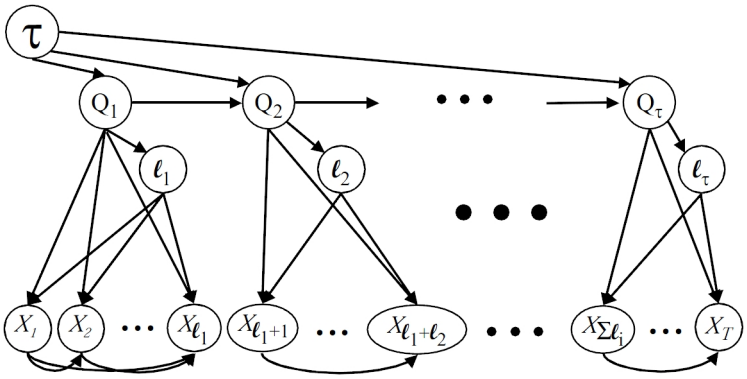


Figure 4.14: Schematic depiction of a Segment Model (from [Bil01]). The X_t nodes represent the observed states, the Q_n nodes represent sequential hidden states, the random variable l_n denotes the sequence length of the corresponding state and τ determines the total number of sequence states Q_n.

The considerations from section 4.1.3 for the choice of conditional probability distributions hold for this domain as well. Consequently, the links between hypothesis and evidence (i.e., the intra-slice conditional distributions $P(H|E)$) are again modeled as gaussian distributions. The inter-slice distribution only links discrete values and therefore can be easily expressed as a 3×3 tabular conditional distribution.

4.3.3 Learning and Results

The learning and testing procedures for this module are the same as described in section 4.1.4, i.e., for a forecast duration of d days, the network is unrolled for $d + 1$ slices and learning and testing is performed alternately with a sliding test window. Final results of this module for the described test data are listed in table 4.4.

The expectations on the nature of a forecasting module based on technical indicators as described in section 2.3 were confirmed with this module: as it was explained before, a small set of indicators should lead to a result of feasible quality while the incorporation of additional indicators can only provide a further increase of rather small magnitude. To confirm this, the network also has been tested with only the value change, a moving average and the Relative Strength Index (these indicators have been selected because they are among the most popular indicators with technical analysts). This subset of evidence states resulted in an average success rate of just under 70 percent. While the success rate denoted in table 4.4 is obviously preferable and therefore justifies the use of further indicators, these results indeed show that the forecasts' improvements are rather small compared to the significant augmentation of the evidence space.

Unfortunately, it is not possible to evaluate every individual indicator's impact on the results, since the large number of combinatorial possibilities for alternative evidence spaces renders an evaluation of all such possibilities intractable. Since some indicators unfold their full potential only in combination with other indicators and conversely an indicator's impact may also vanish if other indicators with nearly redundant information are employed, a complete evaluation of all indicators' contributions to the result would require a testing of all possible combinations.

Though it is intractable to assess all indicators, if one suspects a particular indicator of being useless, it is easily possible to determine this indicator's contributions by simply removing it from the evidence space and comparing the resulting forecasts.

As it was explained in section 2.3, a scrutiny of the On-Balance Volume's developments made its usefulness appear highly questionable. By performing tests for both evidence sets

test instances	success rate
3000-5000	0.83
5001-7000	0.81
7001-9000	0.79
9001-11000	0.82
11001-13000	0.79
13001-15000	0.72
15001-17000	0,67
17001-19000	0.77
19001-20588	0.81
total	0.78

Table 4.4: Evaluation of the technical indicator forecasting module

with and without the OBV's values, it could indeed be confirmed that this indicator did not significantly contribute to the forecast. In fact, introduction of this indicator resulted in only very small fluctuations of the results and since these fluctuations were both positive and negative (i.e., some forecasting instances were improved while an even slightly larger number of other instances was impaired), the average success rate was actually decreased by a small amount. Hence, this indicator represents undesired noise rather than useful information (at least within the presented model) and was therefore removed from the evidence space.

After obtaining this result, other volume-based indicators have been tested the same way and thereby it was revealed that the Ease of Movement is the only one of these indicators that proved to contribute to the results, while the volume and the Money Flow Index only resembled noise, also. These indicators were removed, too, as already marked in table 4.3 (page 63).

The comparison of results for separate test windows as listed in table 4.4 shows that there are different slowly-changing market phases that this module is not able to handle equally well, namely there is a considerable drop in the forecasts' success rate for the data sets from roughly 13000 to 17000. Similar effects were observed with tests of other securities, too. Although the results are satisfactory even in phases of lower success rates and therefore do not raise serious concerns, it would be interesting to determine how these phases differ and what exactly causes the variations in the module's success rate for different phases. Within the limits of this work, these phenomena could not be investigated thoroughly but may rather be subject to future research.

An analysis of the inferred probabilities for each forecast (i.e., the belief state assigned to a particular forecast) revealed another interesting fact: the forecasting module "feels certain" about its forecast virtually every time, roughly 90 percent of the obtained forecasts have an assigned probability of > 0.8 and slightly more than 70 percent of the forecasts even exhibit assigned probabilities of > 0.98 (for comparison: complete indetermination would correspond to a probability of 0.33). While the vast majority of forecasts is obviously correct, as depicted in table 4.4, and therefore may justify such extreme belief states, it appears surprising that the module never shows any signs of indetermination, especially since such high certainties are even assigned if the forecast is wrong. This is unfortunate because a more differentiated assignment of probabilities could possibly assist in improving a trading strategy by keeping away from situations with rather low probabilities (and therefore high uncertainty).

This does not necessarily has to be a deficiency of this model because since stock price movements may always be subject to unforeseen phenomena, even signals considered as highly reliable can be false. In fact, experiments with significantly smaller training sets and small subsets of indicators resulted in significantly lower probabilities for each forecast. This suggests that the large size of training instances and the large set of indicators allows for a sound learning process and therefore high belief states and, consequently, wrong forecasts are likely the result of observations that proved to be reliable in the past but are followed by unforeseen changes in future developments.

It should also be noted that different testing procedures showed that this domain is only locally stationary, i.e., it is not completely time-invariant. This was revealed by using a sliding window for test purposes without updating the training cases. Such a training procedure leads to an increasing gap between the training set and the respective test sets and results have shown that the forecasts' success rate fades while the test window proceeds forward. By retraining the network after completion of each test window, this effect was prevented and therefore it is obvious that the relations of the domain slowly change over time (which is why the process is only locally stationary). While it is not easy

to determine the exact change rate (and this knowledge is useless for the purpose of this work anyways), it could been seen that the effect becomes first perceivable with a gap of roughly 4000 days between training and test data sets. This introduces some moderate constraints on the learning procedures but the sliding window method used within this work remains unaffected by this restricted stationarity and therefore this effect does not require additional attention.

4.4 Candlesticks: Forecasting Module 3

The last forecasting module implemented in this work is based on the analysis of candlestick patterns as described in section 2.4. This domain differs from the previously described analysis methods due to the local character of candlestick prognoses. Since the analysis of a particular pattern depends only on a short sequence of days and since there are usually rather large distances between the occurences of patterns (i.e., there are large sequences which do not emit any information useful for a candlestick analysis), a temporal approach is not feasible in this context. Consequently, this domain is modeled with a static network in which each random variable has a single fixed value for each instance.

4.4.1 Domain Parameters

Evidence

As it was explained before, a particular candlestick is described through its opening, highest, lowest, and closing value (called *OHLC-data*) and thus these values are used to form the observation space. Again, it is necessary to avoid absolute values in order to hinder the network from inferring dependencies of certain price levels. Instead of directly using the OHLC-data, the values are scaled such that they denote a percentaged change with respect to the opening value. Next to resulting in universally applicable descriptions of candlesticks' shapes, this normalization has the convenient side effect that the evidence space is reduced because the opening value can be omitted. Consequently, the observation of a particular candlestick CS_t is described by:

$$CS_t = (H'_t, L'_t, C'_t) \qquad \text{with } H'_t = \frac{H_t}{O_t},\ L'_t = \frac{L_t}{O_t},\ C'_t = \frac{C_t}{O_t} \tag{4.13}$$

Next to the included candlestick shapes within each pattern, it is also necessary to provide information about the candlestick's positions relative to each other. This information is based on the midpoint M_t of a candlestick:

$$M_t = \frac{H_t - L_t}{2} + L_t$$

Using these midpoints, the relative position of a candle j with respect to a preceding candle i is defined as the percentaged change Δ_{ij} of their respective midpoints:

$$\Delta_{ij} = \frac{M_j - M_i}{M_i} \tag{4.14}$$

As explained in section 2.4, a candlestick pattern comprises three consecutive candlesticks and therefore a particular instance of the evidence e_t^T formed by a complete candlestick pattern and a certain trend state T is described by

$$e_t^T = \big(CS_{t-2},\ CS_{t-1},\ CS_t,\ \Delta_{t-2,t-1},\ \Delta_{t-1,t}\big) \tag{4.15}$$

To complete the description of a particular pattern it is also required to provide information about the pattern's context (i.e., the current trend state). Since candlestick patterns are only concerned with a rather short-term situation, it is sufficient to use a short sequence of preceding values in order to determine this trend state. As before, a very short weighted moving average (WMA_3, based on 3 days) is used. An upward movement is assumed if this moving average has been strictly monotonic increasing for at least two days before a pattern was formed. Accordingly, a downward movement is assumed for a strictly monotonic decreasing average. If neither condition is fulfilled the context does not exhibit a clear trend. Thus the trend state T_t for a certain pattern CP_t is defined as

$$T_t = \begin{cases} 1 & \text{if } WMA_3(t-4) < WMA_3(t-3) < WMA_3(t-2) \\ -1 & \text{if } WMA_3(t-4) > WMA_3(t-3) > WMA_3(t-2)\,. \\ 0 & \text{else} \end{cases} \tag{4.16}$$

Since patterns are treated differently depending on the corresponding trend state, these states can be used to classify the observations before feeding them into the network. As a result, instead of having one network for all patterns, three separate instances of the same network are used such that each instance handles patterns of a certain context. This pre-classification results in a significant speedup of the inference process especially since processing time of the clustering algorithm described below increases exponentially with larger data sets.

Prognosis

Since the field of candlestick pattern analysis is a classical task of pattern matching, it is expedient to separate the reasoning process into two subtasks:

First, it is determined whether a certain observation E_i resembles an instance of some known pattern. A discrete clustering variable P is introduced to denote the set of known patterns. Since it is obvious that not every observed candlestick sequence resembles a known pattern, it is not eligible to enforce the classification of each observation into a known pattern. To enable the forecasting module to disregard any instances, an additional "absorbing pattern" is introduced, i.e., a pattern that servers as a placeholder for all sequences that cannot be classified into one of the known patterns. Hence, for a number of n known patterns, the clustering variable P can take on the values $1...n+1$.

After classifying the evidence sequence, the actual prognosis concerning future price developments can be inferred. Experiments with various modelings of the forecast have shown that this domain allows for neither a quantification of future price changes, nor a prediction of particular candlestick characteristics but instead it only facilitates a binary prognosis concerning the direction of a pending candlestick's midpoint. Thus for a prediction of p days into the future, the forecast h_t^p is defined as[4]

$$h_t^p = \operatorname{sgn}\big(\Delta_{t,\ t+p}\big). \tag{4.17}$$

[4]Strictly speaking, this definition would yield three different forecasting states because next to rising and falling days it may also occur that the midpoint positions are exactly equal. However, since a pair of candles virtually never exhibits exactly the same midpoint values in practice, the hypotheses space is modeled binary. In order to achieve an exhaustive definition, the forecast is implemented with the assumption $\operatorname{sgn}(0) = -1$. Due to the practical irrelevance one could also keep the third unneeded state or use other assumptions as well without impacting the results.

4.4.2 Pattern Recognition

Before any evidence instances can be matched with a known pattern, it is required to somehow acquire descriptions of relevant patterns. Different variants have been explored within this work with the goal of obtaining a system that is able to identify any relevant pattern autonomously. While the task of inferring predictions based on unknown pattern memberships resembles the process of learning memberships of unknown mixture components (as described for example in [RN02]), this domain bears the additional challenge that the number of mixture components (i.e., the number of significant patterns) is unknown a priori, too. As explained in the previous chapter, Bayesian networks are not able to cope with variable sizes and thus it is necessary to perform a separate process of pattern recognition in order to obtain information about the number of patterns prior to defining the network.

Evaluation of Possible Approaches

A rather simple approach of acquiring this knowledge is proposed in [CHYL97]: the characteristics of significant patterns are identified by a domain expert and implemented through a set of static rules. The situation with this approach is similar to the alternatives discussed in section 4.3.2 and therefore bears the same disadvantages. Again, instead of entrusting the reasoning component with the task of learning significant patterns, the system has to rely on the knowledge of a human expert and consequently its results are strictly limited by the quality of the predefined rules. It is especially not possible that the system can find any supplementary patterns unknown to the domain expert and therefore the machine learning task is reduced to merely an evaluation of the quality of the expert's knowledge. In fact, [CHYL97] explicitly states that its AI component only serves as some kind of quality assurance as it double-checks the decisions obtained through the use of static rules. While it may be useful to have such a quality assurance, it is obvious that this approach is not able to identify unknown patterns autonomously and therefore lacks the means of fully exploiting the potential of a machine learning process.

An alternative approach that offers unsupervised learning of candlestick characteristics has been described in an internet blog.[5] This article proposes the categorization of individual candlesticks into various types with distinguished shapes by using the *k-means clustering algorithm* and a construction of pattern definitions based on these types. K-means clustering is a simple center-based clustering procedure and with a predefined number of clusters k it works as follows:

1. The k cluster centers are distributed randomly.
2. Each data point is assigned to the cluster with the shortest distance to its center (i.e., the cluster with the nearest mean, hence the name).
3. The cluster centers are recomputed such that they form the mean of all containing data points.
4. Based on the new cluster centers, the procedure continues with step 2 again. This procedure is performed repeatedly until either a predefined number of iterations has been performed or until an iteration does not alter cluster memberships of any point.

For more information on this procedure and other clustering methods see [MW07].

An implementation of this proposed clustering procedure revealed two significant short-

[5] http://intelligenttradingtech.blogspot.com/2010/06/quantitative-candlestick-pattern.html, the mentioned article has been posted under the title "Quantitative Candlestick Pattern Recognition (HMM, Baum Welch, and all that)" on June 10, 2010. Unfortunately the article's author remains anonymous.

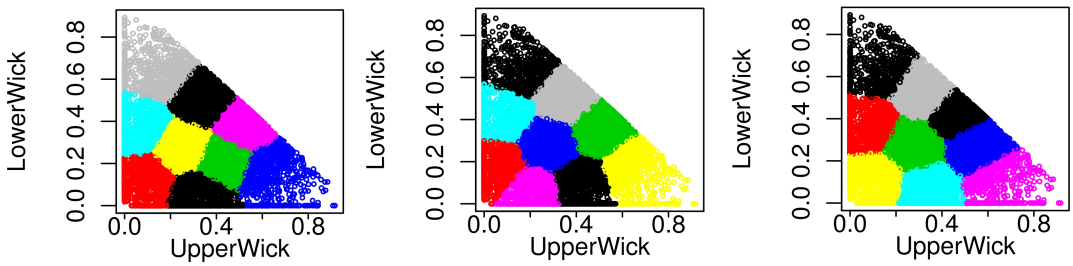


Figure 4.15: Clusters of candlestick shapes obtained from various executions of the k-means clustering algorithm, the lengths of the wicks are given as a fraction of the whole candlestick's size (the cluster colors are only used for a better visualization and do not depict any correlations).

comings of the k-means clustering algorithm. To illustrate these shortcomings, figure 4.15 depicts the resulting clusters for three executions of the attempt of categorizing candlesticks into nine clusters with respect to their proportionate wick sizes.

A comparison of the results immediately reveals one disadvantage of k-means clustering: due to the arbitrary initialization of cluster centers, the results are not reproducible but instead may vary with each execution of the algorithm. Although this problem can be attenuated by executing the algorithm several times and averaging the results, it is hardly possible to construct any feasible pattern descriptions on the basis of varying clusters. An alternative solution to reach reproducible results is the replacement of random initialization with predefined initial cluster positions. This would lead to fairly reproducible results but bears the disadvantage that a suboptimal manual initialization will have a strong impact on the results. The other problem of employing the k-means clustering algorithm in the candlestick domain is the fact that it always results in clusters of spherical shape and is therefore not able to capture the truly significant shapes of particular candlesticks. For instance, as it was explained in section 2.4, the diminutive size or complete absence of a shadow carries valuable information. To reflect these characteristics in the classification of candlesticks, the results depicted in figure 4.15 would be required to exhibit long and narrow clusters at the graph's edges. Obviously such a cluster cannot be obtained if clusters are limited to spherical shapes.

To resolve the problem of spherical clusters, an alternative clustering procedure has been explored, namely *Density Based Spatial Clustering of Applications with Noise (DBSCAN)*. This algorithm is able to find clusters of arbitrary size based on the density of data points in certain regions. A thorough description of this algorithm can be found in [MW07]. However, an examination of the data points in figure 4.15 shows that they are nearly even distributed and therefore any clustering approach based on density is deemed to fail. In fact, an application of DBSCAN to this dataset leads in essence to a single large cluster (with the exception of some outliers).

These two variants illustrate an elementary problem of all "flat" clustering algorithms (opposed to hierarchical methods as explained subsequently) for this domain: either the data set is equally partitioned in clusters of fixed shape, which are not able to capture the candlesticks' key properties, or no meaningful clusters can be found at all due to the nearly even distribution of data points.

Before proceeding to the description of a feasible clustering method, it should be noted that the approach of determining particular candlestick types and constructing pattern descriptions based on these types - even though it may appear promising at first - did not yield feasible results. This is mainly due to the tremendous number of possible patterns: a

meaningful classification of individual candlesticks should contain at least three types for each shadow's size (diminutive, average, large) and five types for the candle's body (large red, small red, diminutive, small green, large green). By analyzing sequences of three candlesticks this leads to $(3 \cdot 3 \cdot 5)^3 = 91125$ possible patterns even without taking the relative position of each candlestick into account. Thus, the number of possible patterns is significantly larger than the size of the training set and as a result this approach did not find any reoccurring patterns in the training data. Instead of clustering individual candlesticks, the following algorithm is applied to complete descriptions of candlestick sequences according to equation 4.15 such that the resulting clusters directly represent meaningful patterns.

Hierarchical Clustering

Since flat clustering approaches are not able to yield feasible results in this domain, a hierarchical method is used instead. Hierarchical clustering methods construct an ordered structure of clusters such that the distance according to some distance measure between respective cluster members increases with the hierarchical order of a cluster. To illustrate this, figure 4.16 depicts an example of a hierarchical cluster structure: in a complete structure, the top-most cluster contains all elements of the data set and its children are split such that they form two clusters with a minimal distance between all respective cluster elements.

In general, there are two approaches to construct a hierarchical clustering structure:

Agglomerative hierarchical clustering ("bottom-up clustering") starts with assigning each element to a separate cluster. Afterwards, in each iteration the two clusters with minimal distance are merged into a new cluster so that successively larger clusters are constructed. This process continues until all data points are merged into a single cluster.

Divisive hierarchical clustering ("top-down clustering") works in the opposite direction. It is initialized with a single cluster containing all data points and in each step the cluster is split such that two clusters with a minimal intra-cluster distance are created. This process continues until all data points are assigned to separate clusters. More information on hierarchical clustering methods can be found in [MW07].

The agglomerative approach has an advantage in this domain because after finding desired clusters (according to conditions explained below), the clustering process may terminate early without constructing the complete structure. Since the clustering process is very time-consuming, this early termination offers a significant speedup.[6]

A useful distance measure to define similarities between different observations in this context is the euclidean distance. Hence, the distance between two patterns x and y (of the form described in equation 4.15, containing 11 properties) is defined as:

$$d(x, y) = \|x - y\|_2 = \sqrt{\sum_{i=1}^{11} (x_i - y_i)^2}$$

After merging two observations into a cluster, the feature values of the resulting cluster are set to the respective average values of both merging candidates. Applying agglomerative hierarchical clustering with this distance measure to the training set of input patterns[7]

[6] The early termination is only a performance optimization that does not alter the functionality. Thus in the following a full construction of the hierarchical structure is assumed in order to simplify explanations.

[7] As explained before, the training set is actually partitioned into three subsets according to the corre-

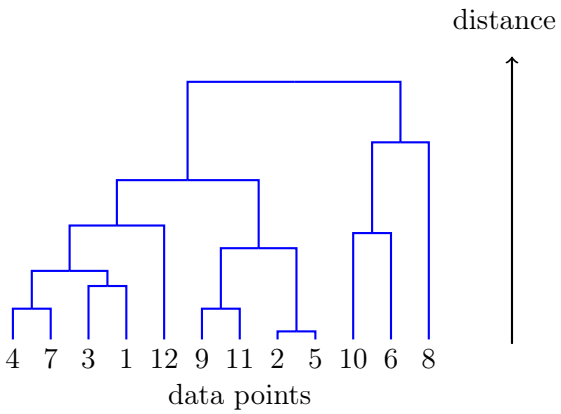


Figure 4.16: Schematic depiction of a hierarchical cluster structure (this type of graph is called *dendogram*)

results in a structure that provides information about the similarity of different observations and therefore can be used to identify reoccurring patterns.

Once this structure has been constructed, it can be easily used to query for significant patterns. To identify significant patterns, two properties for each cluster (i.e., each possible pattern) are defined: First, it is necessary to ensure that a cluster indeed represents a reoccurring pattern and not only some random effect. In order to ensure reoccurrence, a cluster must contain at least a certain minimum number of members (representing instances of a particular pattern). This is denoted by the instance count I_P of a pattern P and simply defined as

$$I_P = \#\text{ instances for a pattern } P = \#\text{ elements in the corresponding cluster.}$$

Also, a pattern should exhibit a certain reliability R_P, i.e., a certain ratio of all cluster members need to be followed by the same subsequent development. The reliability of a pattern P is defined as

$$R_P = \frac{\max\ \#\text{ instances with the same outcome}}{I_P}.$$

This stresses the importance of considering the instance count: each of the initial clusters only contains a single instance and thus exhibits a reliability of 100%. Without requiring a certain number of occurences, all of these clusters would be considered significant.

Based upon these properties a pattern is considered significant if both values exceed a certain threshold (i.e., $I_P \geq I_{\min}$ and $R_P \geq R_{\min}$). Note that the parameter $I_{\min}$ cannot be set universally but instead must be adapted to the size of the training set. Obviously, this parameter must be set the higher the larger the training set is.

Search for these pattern starts at the bottom of the hierarchical structure. As soon as a cluster is found that meets the specified criteria, it is added to the set of significant patterns and its parents are omitted from continuation of the search. This is because if a pattern is already considered as significant, an addition of further instances bears the danger of diluting the pattern's properties by considering auxiliary insignificant instances.

Result of this procedure is a set of autonomously detected significant patterns, which can be used to train the parameter P described in the previous section.

sponding trend states and therefore the clustering is performed three times for subsets of the training data. Due to the clustering procedure's exponential complexity, this is the step that profits most (in terms of runtime requirements) from partitioning the input set.

4.4.3 Prediction Network

Once the information about significant candlestick patterns has been obtained through agglomerative hierarchical clustering, the design of a feasible network is fairly simple.

As depicted in figure 4.17, the evidence (i.e., the pattern description according to equation 4.15) is used to determine its pattern membership. Note that, next to all known patterns, P also provides the option for a missing pattern membership, as explained before. The conditional probabilities $P(e|p)$, denoting that a particular observation e is due to a certain pattern p, are again modeled as gaussians. Hence, the probability of this observation is a mixture of gaussians:

$$\begin{aligned} P(e) &= \sum_{p=1}^{n+1} P(p)P(e|p) \\ &= \sum_{p=1}^{n+1} w_p \cdot \mathcal{N}(\mu_p, \sigma_p^2) \end{aligned}$$

w_p denotes the mixing weight for each component and can easily be determined by the prior $P(p)$ which in turn is determined by the ratio of p's number of occurences and the size of the training set.

Usually the task of learning mixtures of gaussians is used for unsupervised learning (as explained for example in [RN02]), i.e., inference of a set of cluster memberships, while only the number of clusters but no information about their properties is provided. However, in this scenario the subtask of learning the properties of the mixtures is not truly unsupervised as information about the clusters is provided from the results of the agglomerative hierarchical clustering (although by considering the whole learning process with included pre-clustering, the task of learning candlestick patterns is indeed unsupervised).

One might wonder why the Bayesian network is still tasked with inferring relations between evidence and clusters even though cluster memberships have already been determined through the previous clustering process. Instead, the evidence node could be omitted and the network could be used solely for inferring relationships between certain patterns and their respective prognoses. While this too would lead to valid results, the incorporation of the evidence node is mainly due to performance reasons: the hierarchical clustering structure cannot be easily augmented with additional evidence because new nodes cannot be simply appended but each new instance of evidence could possibly change the clustering structure. Hence, using the hierarchical cluster structure for determining pattern memberships of the test data would lead to repeated recalculations of the clustering hierarchy and, as explained before, these calculations are rather time-consuming. Determining the pattern membership within the Bayesian network as depicted in figure 4.17 proved to be significantly faster and an evaluation of both variants showed that they are able to produce results of equal quality. However, as explained before it is not possible to dispose of the pre-clustering completely because the gaussian mixture learning

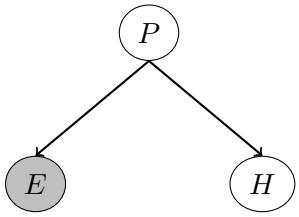


Figure 4.17: Bayesian classifier for candlestick pattern matching

trend context	total # instances	reliability ($R_{\min}$)	# identified pattern types	# matched pattern instances	matching ratio	success rate
up	6262	60 %	75	2914	0.47	0.66
		70 %	48	1914	0.31	0.69
		80 %	32	1232	0.20	0.72
		90 %	10	387	0.06	0.82
		100 %	2	70	0.01	0.95
down	5129	60 %	52	2086	0.41	0.66
		70 %	33	1243	0.24	0.72
		80 %	22	842	0.16	0.73
		90 %	10	413	0.08	0.79
		100 %	1	35	0.01	0.86
none	3609	60 %	35	1387	0.38	0.68
		70 %	22	804	0.22	0.69
		80 %	16	616	0.17	0.72
		90 %	6	244	0.07	0.84
		100 %	0	0	0	-

Table 4.5: Evaluation of the candlestick analysis module: the table depicts the results for a one-day forecast with $I_{min} = 30$ and varying $R_{\min}$ parameters sorted by the corresponding trend context. For each parameter $R_{\min}$ the number of identified pattern types, the total number of observation instances that could be matched to one of the patterns, the corresponding matching ratio (i.e., the fraction of all evidence instances that were identified as significant patterns), and the resulting success ratio of the forecasts are listed.
Note that a reliability of 100 % merely denotes that all observed instances of a pattern were followed by the same outcome (i.e., the data set did not contain any counterexamples) and must not be confused with a prognosis of absolute certainty.

requires a predefined number of mixture components and therefore cannot be used before determining this number through the hierarchical clustering.

Once the pattern membership of a particular instance has been determined, the actual forecast is inferred based on the hidden pattern. Since both the pattern and the forecast are discrete variables, the corresponding conditional probability $P(H|P)$ can be easily defined through a conditional distribution table.

4.4.4 Learning and Results

Learning and testing with this module has been performed in the same way described before with a sliding test window and alternating training and test procedures.

Experiments with varying query parameters $R_{\min}$ and $I_{\min}$ have shown that $I_{\min} = 30$ provides a feasible threshold for the described DJI data set. The corresponding results for a one-day forecast with varying $R_{\min}$ parameters are listed in table 4.5. A comparison of the used reliability thresholds and the according success rates of the forecasts shows that the resulting success rate reflects a rather close tracking of the respective reliability parameter. Thus, by deciding upon a certain $R_{\min}$, one can tune the success rate nearly arbitrarily. Although at first glance it may appear desirable to increase the success rate to

the highest amount possible, an analysis of the listed results shows that it not necessarily the best choice to aim for a maximized success rate. For instance, a success rate of 95 %, as it has been obtained with $R_{\min} = 1$ in the uptrend context, can certainly be considered an amazing result. However, taking the corresponding matching ratio into account, as well, reveals that this result is not so pleasant after all. A matching ratio of 0.01 reflects roughly two occurences of such a highly reliable pattern per year. This means that a signal is generated only approximately twice per year and is concerned with only forecasting movements of the following day. Obviously such a sparse existence of very short-term signals renders this result virtually useless in practice, although its quality is outstanding. If instead one would for example use a reliability of 70 % (and therefore accept occasional false signals), the resulting success rate would still be very satisfactory but, on average, signals are generated more than once per weak and thus the overall gain of any trading strategy employing this parameter would result in significantly higher gains.

Tests with forecasts of various durations were able to confirm the statements from section 2.4 regarding the forecasting ability: a forecast three days into the future with $R_{\min} = 0.7$ resulted in a success rate of roughly 60 % while the success rate of a four-day forecast was not able to exceed 50 % significantly and therefore can hardly be of any use in practice. Hence, this module should be used for forecasts with a maximum of three days into the future.

5 Inference of Trading Strategies with Reinforcement Learning

The separate forecasting modules described in the previous chapter are able to provide valuable information about future price movements, but their results are not yet sufficient to completely define a trading strategy. Especially due to the variety of information obtained from the probabilistic reasoning processes, it is not always possible to extract an obvious mapping from forecasting results to expedient trading decisions. There are still several issues which have to be resolved when handling the forecasting results. For instance, questions that still remain unanswered are:

- What is the best decision if separate forecasting modules differ in their respective prognoses?
- How should one react when forecasts vary for different time horizons?
- How can separate forecasts be combined in order to find optimal points in time for a trading action?

To answer these questions, another learning module is presented in this chapter, which aims at learning an optimal trading strategy based on the results of the previously described forecasting modules. This learning module is based on *reinforcement learning* and is described in the following.

5.1 Introduction to Reinforcement Learning

This section provides a summary of the key features of reinforcement learning as described in [RN02] and [SB98]. In order to illustrate the principles of reinforcement learning, [RN02] gives the following description:

> "Imagine playing a new game whose rules you don't know; after a hundred or so moves, your opponent announces, 'You lose.' This is reinforcement learning in a nutshell."

More specifically, reinforcement learning separates the learning task into two entities, a learning and decision-making entity called *agent* and the thing an agent interacts with (basically everything outside the agent), called the *environment*. The agent and environment interact with each other at discrete time steps $t = 1, 2, 3...$ The environment has a set of possible states $\mathcal{S}$ and for each time step t, the agent receives information about the current environment's state $s_t \in \mathcal{S}$. Based on the environment's state, the agent selects an action $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ denotes the set of actions available for the state s_t. The agent communicates his selection to the environment and, in response, receives an numerical reward $r_{t+1} \in \mathbb{R}$ together with a new state description s_{t+1}. This communication cycle forms the basis for reinforcement learning and is depicted in figure 5.1.

For the scenario described in the quotation above, the player corresponds to the agent and everything else (i.e., the opponent and the rules of the game) forms the environment. Playing the game once corresponds to a single time step and yields a binary reward through either winning (which is good and should therefore lead to a positive reward) or losing

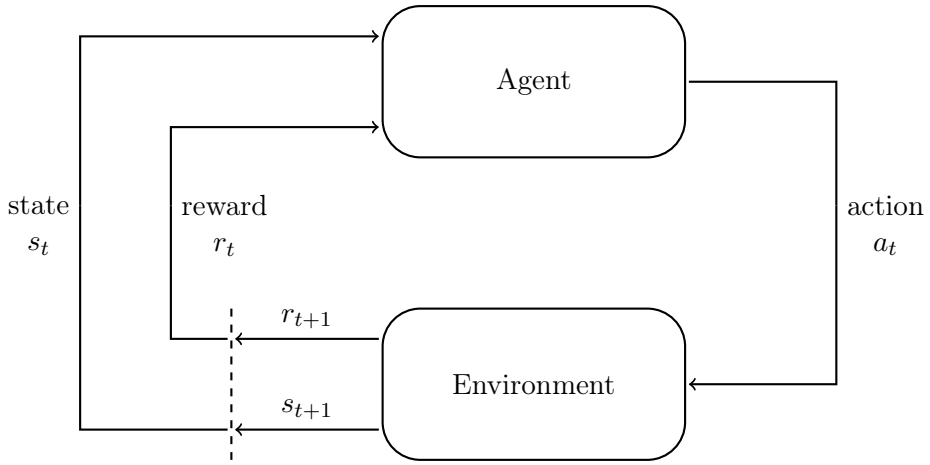


Figure 5.1: Cyclic proceedings in reinforcement learning (according to [SB98]): the agent receives a state description s_t and a reward r_t from the environment, selects an action a_t and in turn the environment replies with a successive state description s_{t+1} and reward r_{t+1}.

(which is bad and should lead to a negative reward). Initially, the agent has no idea how the game works and thus is not able to aim at winning. Through repeated playing he will (hopefully) observe how his actions lead to certain results and how these results will eventually conclude in a victory or defeat. By the use of rewards he should also learn that winning is better than loosing and thus after playing the game for various times, the agent is expected to choose his actions such that they are most likely to lead to a victory. Consequently, the agent is expected to learn the game through trial and error, which is essentially the approach of reinforcement learning. The name of this approach stems from the fact that "good" actions receive high rewards and are thereby reinforced while "bad" actions are punished and thus attenuated.

5.1.1 Formalization of the Learning Task

To formalize the task of reinforcement learning, a utility function is introduced in order to measure the quality of an agent's actions. The most simple concept of a utility function can be obtained by simply accumulating the rewards for a sequence of states:

$$U(s_0, s_1, ..., s_n) = r_1 + r_2 + ... + r_{n+1} \tag{5.1}$$

Obviously, an agent's task in reinforcement learning is to choose his actions such that they lead to states with high rewards and therefore its ultimate goal is to maximize this utility function. If an agent was tasked with learning a maximum for this function, its aim would be to choose each action so that it results in a successive state sequence with highest rewards and therefore would lead to a globally maximized utility. While it is of course desirable to reach such an optimal utility, this function introduces a computational problem in practice: as it can be clearly seen from figure 5.1, a single action a_t will influence all future states $s_{t+1}, s_{t+2}, ..., s_{t+n}$ and all future rewards $r_{t+1}, r_{t+2}, ..., r_{t+n}$. Hence, before the agent can know which is the most desirable state s_{t+1} (and therefore know which is the most desirable action a_t to cause this state), it would be required to evaluate all successive alternatives $s_{t+2}, s_{t+3}, ...s_{t_n}$. Such an evaluation will become quickly intractable if the sequence of states is not limited to a rather short length. This problem can be illustrated by considering the target domain of inferring trading strategies: in this domain it is reasonable to provide the agent with three different actions for each state (buy, sell,

do nothing). If considering the data set described in chapter 4 (comprising data for 20588 trading days), before an agent can truly determine the globally best choice for a_1, it would have to test the potential rewards for all possible successive sequence. Thus, before the best action a_1 can be found, $3^{20588}(\approx 10^{9823})$ alternatives would have to evaluated, which is obviously a futile task. Also, it should be noted that the assumption of a single action influencing all future states is all but meaningful in the financial domain. As it will be discussed below, the described trading agent does not truly influence future states $s_{t+\tau}$ by his actions but merely the rewards $r_{t+\tau}$ associated with tuples of states and actions. Hence, the influence of any particular action will vanish as soon as the agent executes another different action in the future.

To overcome this intractability problem, it is common to model an agent such that it prefers current rewards over future rewards. This is done by introducing a discount factor γ, which leads to a decay of future rewards when calculating the utility. This also introduces the benefit that time series of infinite length (for simplicity denoted by $S = s_0, s_1, ...$ subsequently) can be handled:

$$U(S) = r_1 + \gamma r_2 + \gamma^2 r_3 + ... = \sum_{t=0}^{\infty} \gamma^t r_{t+1} \tag{5.2}$$

With these considerations, the agent is first tasked with learning the environment's transition model (i.e., the agent should learn how the environment evolves). This transition model is described through a set of probabilities $\mathcal{T}^a_{ss'}$ denoting the probability of reaching the successive state s' if an action a is selected in some state s:

$$\mathcal{T}^a_{ss'} = P(s_{t+1} = s' \,|s_t = s \wedge a_t = a) \tag{5.3}$$

Simultaneously with learning the transition model, the agent should also learn what action $a \in \mathcal{A}(s_t)$ to take in a particular state $s = s_t$ such that it receives a maximum reward r_{t+1} and therefore he should learn an expedient mapping between states and action. Such a mapping is called *policy* and is usually denoted by π (consisting of mappings $a = \pi(s)$). With respect to a certain policy, the utility of some state s can be expressed as

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi,\ s_0 = s\right] \tag{5.4}$$

A policy is called optimal and denoted by π^* if it yields the highest expected utility:

$$\pi^* = \arg\max_\pi \ E\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi\right] \tag{5.5}$$

Hence, the optimal policy for a certain step s is:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{T}^a_{ss'} \cdot U^{\pi^*}(s') \tag{5.6}$$

As explained in [SB98], assuming that an agent selects optimal actions leads to the so-called *Bellman equation*, which gives the utility of an individual state $s = s_t$ (consisting of the immediate reward for being in state s (denoted by $R(s)$) and the expected discounted utility of the next state s'):

$$U(s) = R(s) + \gamma \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{T}^a_{ss'} \cdot U(s') \tag{5.7}$$

By solving this equation for all possible states $s \in \mathcal{S}$, the agent could determine an optimal policy and thereby solve the learning problem.

5.1.2 Learning Procedures

Temporal Difference and Q Learning

Although the previously explained equation 5.7 provides an option to find the optimal policy π^*, solving the resulting system of equations for all possible states s may still be intractable for large state spaces $\mathcal{S}$. An approximative solution to these equation can be obtained through *temporal difference learning (TD learning)*. Instead of explicitly computing equation 5.7 for each state, TD learning uses (randomly initialized) utility estimates for all states and incrementally updates these estimates with observed rewards from samples. These updates are formed by the difference of a state's current utility estimate and observed rewards from successive states.

To illustrate this concept, consider first the task of simply determining the utility of a specified fixed policy (this task is also called *passive reinforcement learning*) with regard to the differences between two successive states (this is called *TD(0) learning*, the reason for this name will become apparent with subsequent explanations). In this scenario, utility estimates are updated as following:

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha\big(R(s') + \gamma U^\pi(s') - U^\pi(s)\big) \tag{5.8}$$

New observations are weighted with $0 \leq \alpha \leq 1$. This parameter is called the learning rate and determines the influence of new samples, i.e., $\alpha = 0$ would mean that new observations are not considered at all while $\alpha = 1$ would lead to a complete replacement of previous knowledge. This update step is also called *sample backup* because it looks ahead to a successor state, uses the successor's value and the reward to compute a backed-up value, and changes the value of the original state accordingly.

Since the ultimate goal of the agent is to learn an optimal policy, the utility estimates from TD(0) learning can be replaced with estimates of the rewards of state-action pairs, denoted by $Q(s, a)$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(R(s') + \gamma \max_a Q(s',a) - Q(s,a)\big) \tag{5.9}$$

The resulting process is called *Q-learning* and it can be shown (see [SB98]) that the learned Q estimates approximate the optimal state-action pairs Q^*, independently of the policy being followed (provided that the policy visits all states). Consequently, the resulting Q estimates can be used directly to form the optimal policy π^*.

TD(λ) and Q(λ) learning

The TD(0) procedure described above is only concerned with looking one step ahead and thereby uses 1-step backups. In order to achieve more accurate results, one can also extend the lookahead n steps into the future (leading to an n-step backup). The reward $R_t^{(n)}$ of such an n-step backup is defined as:

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^n U(s_{t+n})$$

and is called n-step return. This Reward $R_t^{(n)}$ comprises a sequence of $n-1$ actually observed rewards and terminates with a utility estimate according to equation 5.8 for the state s_{t+n} . Therefore this expression is also called *corrected n-step truncated return.*

Instead of considering only a particular n-step return, one can also consider an average of various weighted n-step returns for varying n. Such a combination of varying n-step returns

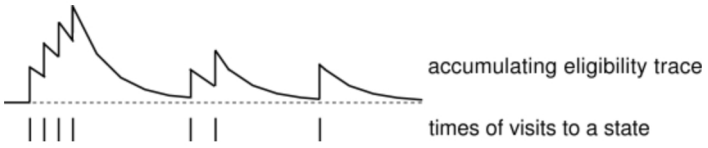


Figure 5.2: Schematic development of an eligibility trace (from [SB98])

is called *complex backup* and can be constructed from an arbitrary set of returns, as long as the mixing weights are positive and sum to 1. A common approach of constructing such a complex backup is to use all possible n-step rewards and assign exponentially decaying weights:

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_t^{(n)} \tag{5.10}$$

with a decay factor $0 \leq \lambda \leq 1$. The additional normalizing constant $(1-\lambda)$ is used to ensure that all weights sum to 1. A temporal difference procedure which incorporates such a complex backup with weights decaying by some factor λ is called *TD(λ)*. The choice of λ governs the influence of successive states, i.e., with a high value of λ a large proportion of the rewards is obtained from more distant states. For $\lambda = 0$ this procedure is exactly the one-step TD learning method described before.

As explained in [SB98], the above considerations cannot be implemented directly because equation 5.10 is *acausal*, meaning that at each step it uses knowledge of what will happen many steps later. To allow for an implementation of this procedure, an additional memory variable called *eligibility trace* is associated with each state. The eligibility trace for a state s at a time t is denoted by $e_t(s) \in \mathbb{R}^+$. When performing the learning task, these eligibility traces are decayed by $\gamma\lambda$ in each step t and the eligibility trace for the visited step is is incremented by one:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases} \tag{5.11}$$

These eligibility traces record which states have recently (defined in terms of $\gamma\lambda$) been visited and therefore they indicate whether a state is *eligible* for changes in the case of reinforcement events. To illustrate this, figure 5.2 schematically depicts the development of an eligibility trace.

These eligibility traces are used to quantify how much a utility estimate is changed at some time t:

$$\Delta U_t(s) = e_t(s) \cdot \alpha \cdot \big(r_{t+1} + \gamma U(s_{t+1}) - U(s_t)\big) \tag{5.12}$$

Note that recently visited states exhibit a high eligibility trace and therefore they are subject to rather large changes while estimates of states which have not been visited recently remain virtually unchanged due to their negligible eligibilities. As a consequence, the use of eligibility traces facilitates the decaying weighting scheme from equation 5.10 and therefore yields a correct implementation of the TD(λ) procedure, as it has been shown in [SB98].

Again, TD(λ) offers an analogous procedure (called $Q(\lambda)$[1])to learn estimates of state-action pairs. Eligibility traces are used in a similar fashion as described before, with the exception that they are set to zero whenever an exploratory action is selected. The use of exploratory actions is explained below, for now it should only be noted that an agent

[1]In fact, different learning methods have been proposed under the name $Q(\lambda)$. The method described here refers to the one proposed in [Wat89]

usually selects actions with the expected highest rewards but may also select other actions in order to augment its knowledge. To simplify notation, eligibility traces e_t for Q(λ) learning are denoted as two separate components e^1 and e^2:

$$e_t^1(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } Q_{t-1}(s_t,a_t) = \max\limits_a Q_{t-1}(s_t,a) \\ 0 & \text{else (exploration action)} \end{cases}$$

$$e_t^2(s,a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t \\ 0 & \text{else} \end{cases}$$

$$e_t(s,a) = e_t^1 + e_t^2$$

Using these eligibility traces, updates of the Q estimates are performed in analogy to equations 5.9 and 5.12:

$$\Delta Q_t(s,a) = e_t(s,a) \cdot \alpha \cdot \left(\gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)\right) \tag{5.13}$$

Performing these update steps iteratively will eventually lead to convergence towards the optimal state-action pairs Q^*.

The main advantage of TD(λ) and Q(λ) methods compared to the previously described temporal difference and Q learning methods is that they exhibit a faster convergence. However, it should be noted that it is common to start reinforcement learning with a rather high frequency of exploratory actions (as explained in the next section) and since the Q(λ) eligibility traces are reset whenever selecting such an action, there is only little advantage of using eligibility traces when starting the learning process.

5.1.3 State Space Exploration

The previous section explained how an agent is able to infer knowledge about its best actions, but it did not yet provide any actual criteria for deciding upon a particular action.

At first sight it may seem obvious to simply pick the action that corresponds to the highest expected gain (with respect to the current knowledge). This approach is called a *greedy strategy* or *exploitation strategy*. However, always picking the greedy action would most likely cause the agent to quickly stall in local optimum without ever being able to learn the actual globally optimal policy π^*. This is due to the fact that such a greedy agent is unwilling to take the risk of exploring unknown states, i.e., as soon as agent has learned that in some state s, a particular action a will result in a positive reward, he will always choose this action a if he arrives in state s subsequently. However, there may be alternative but yet unexplored actions which could lead to significantly higher rewards. Since these actions are not explored yet they are not associated with a positive expected reward and consequently they will never be chosen by the agent.

The complete opposite of a greedy agent would be a pure exploring agent, i.e., an agent who always chooses actions randomly without ever considering previously learned knowledge about expected rewards. The corresponding strategy is therefore called *exploration strategy*. In theory, such an agent is able to learn the globally optimal strategy eventually after exploring the complete space of state-action pairs. However, as explained in section 5.1.1, the exhaustive exploration of the space of state-action pairs is intractable for most domains. Even if this exploring agent was somehow able to learn the optimal policy despite the tremendous computational requirements, this knowledge would be of no use as long as the agent keeps its exploring policy and remains ignorant of his knowledge.

parameter	name	meaning
$0 \leq \alpha \leq 1$	learning rate	displacement weight of new knowledge over previous knowledge
$0 \leq \gamma \leq 1$	discount factor	preference of future rewards over current rewards
$0 \leq \lambda \leq 1$	trace decay parameter	decay of a particular state's influence over time (through its eligibility trace)
$0 \leq \epsilon \leq 1$	exploration parameter	rate with which explorative actions are selected

Table 5.1: Parameters for reinforcement learning

A common solution to this problem is a so-called *ϵ-greedy strategy*. Such a strategy is defined with respect to some $0 \leq \epsilon \leq 1$, which provides a ratio of exploitative and explorative actions. For a proportion $1 - \epsilon$ of the trials, the agent is supposed to choose the greedy action while for the remaining proportion ϵ he is asked to explore alternative actions. More formally, the agent chooses greedy actions a_g and explorative actions a_e with the probabilities

$$P(a_g) = 1 - \epsilon$$
$$P(a_e) = \epsilon$$

Consequently, $\epsilon = 0$ denotes a pure greedy agent and $\epsilon = 1$ denotes a pure exploration agent. In practice usually a parameter $\epsilon \ll 1$ is selected, though the choice may vary with the corresponding domain.

5.1.4 Summary

As it can be seen from the previous discussions, a reinforcement learning agent is neither required to have a model of the domain nor does it has to know how to select actions initially. This is why reinforcement learning is also called *model-free learning*. Instead, the learning task is only defined through the previously introduced learning parameters, which are summarized again in table 5.1. Consequently, a RL agent is a general purpose agent which - after being implemented once - can be applied to various domains. Note that the discussions of this section are limited to the use of this work. There are also alternative approaches of reinforcement learning which may be specified with different parameters. For more information see [RN02] and [SB98], for instance.

5.2 State Space Encoding

The previously described methods of reinforcement learning require that the pairs of states and actions can be represented in a tabular form. Thus it is necessary to have a set of distinct states which consequently requires a discrete state space. If a domain exhibits states with continuous values, these values have to be discretized by some means before they can be used with reinforcement learning. The most simple approach of performing such a discretization is a partitioning of the continuous state space into intervals of a certain step width and assign each continuous value to its corresponding interval (hence, an implementation of a classical analog-to-digital converter). However, such a simple discretization method has the problem of resulting in a rather large number of discrete

states for many domains. To illustrate this, consider the continuous forecasts obtained from the trend line module described in section 4.1. Using a long-term forecast for the previously described DJI data set yields a set of forecasts in the range $[-31.53, 70.05]$ (percentaged value changes). Even if one used a step size of 1 (which leads to a rather coarse resolution), roughly 100 steps would be required to discretize the input values. Since the goal of this reinforcement learning module is to use forecasts of different time scales, the size of the state space virtually explodes when incorporating forecasts of varying time scales with this discretization method.

Such an overdimensioned state space bears a variety of problems. First, there are obvious practical issues: as explained before, the agent is tasked with learning the values for each state-action pair, and thus, a large state space will require the exploration of many different options, which in turn will lead to a very low convergence speed. Since each particular state has to be explicitly represented with a corresponding utility value and an eligibility trace, the memory requirements of a large state space can quickly exceed the available hardware resources.

Even if one is able to overcome the practical problems, there remains another - even more severe - issue. With a training set not being considerably larger than the state space size, it is very likely to happen that each instance of the training data will be represented through a unique state. As a consequence, the agent would learn how to react for each specific observation contained in the training data, but it would not be able to apply this knowledge to any new observations, since it would be unable to detect similarities between different states. This effect is called *overfitting* and hinders the facilitation of an expedient learning process. As a result, one must find a state space encoding of significantly smaller size than the training data in order to enable a *generalization* of the learning task, i.e., the states must be encoded such that an agent is able to apply knowledge of previously visited states to ones that have never been seen before.

In this work, the state space generalization is performed through *tile coding*. As explained in [SB98], tile coding groups the input space into an exhaustive set of partitions, so that each element represents a single binary feature. Such a partitioning is called *tiling* and each element of a tiling is called a *tile*. This is best illustrated with a two-dimensional example, as depicted in figure 5.3: the input space in this example consists of two continuous variables $x, y \in [0, 10]$ and is separated into a tiling with 16 tiles (marked as the black grid). Through using this tiling, each point (x, y) can be specified with a binary vector of length 16 with each vector element denoting the membership of a particular tile. Hence, all but element of this vector will be 0.

Note that, due the the large step size, the chosen tiling provides a rather coarse approximation of the input space. A significant improvement of the coding resolution can be achieved by using several instances of the same tiling, each offset by a random amount (depicted through the blue and red grid in figure 5.3). Mappings of points from the input space to specific tiles are then weighted according to their memberships of each tiling instance. The weights for each tile can be obtained by summing over the corresponding vector elements of all tiling vectors. For example, as shown in figure 5.3, the point p is member of tile 10 with weight $w_{10} = 2$ and of tile 11 with weight $w_{11} = 1$ (and obviously membership to all remaining tiles is weighted with 0). The use of several equal tilings with random offsets according to this weighting scheme allows for a local generalization while resulting in a rather small state space. As described in [SB98], random offsets of the tilings exhibit a significantly better performance than offsetting each tile by the same increment.

This technique can be applied to input spaces of arbitrary dimension and leads to the representation of each tile as a hyperrectangle. Note that the tilings do not necessarily need to form uniform grids but instead the tiling resolution can be set independently for

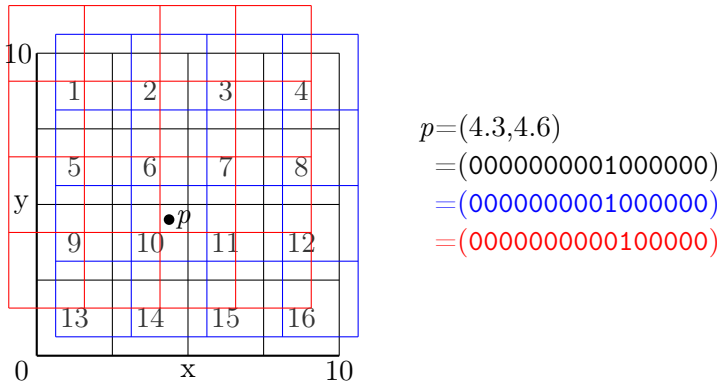


Figure 5.3: Tile coding example for two continuous variables $x, y \in [0, 10]$. The respective binary tiling vectors are shown for an exemplary point p, indicating a membership of tile 10 with weight $w_{10} = 2$ and of tile 11 with weight $w_{11} = 1$

each dimension so that each dimension of the input space can be approximated with the required precision.

5.3 Environment Model for the Trading Domain

Before explaining the details of the environment model, it should be noted that the trading agent has a set of three obvious actions for every time step t at its disposal: it can buy ($a_t = 1$), sell ($a_t = -1$) or remain neutral ($a_t = 0$). The actions in this model do not directly compare to actual trading actions: usually a trading agent would have the options of buying, selling, or doing nothing. The actual states of the agent then depend on its previous actions: if he has bought a security before and remains inactive subsequently he is still engaged in a trade. This dependency on previous actions complicates the model, but can be removed easily without altering the agent's functionality, while at the same time leading to a simpler model: instead of denoting actual trading activities, the agent's action are modeled so that they represent the desired trading state of the agent. For instance, if the agent chose a buying action yesterday and wants to keep its securities today, this would be denoted by a buying action again. If instead the agent wants to sell the securities, this would be denoted by a neutral action (because a buy followed by a sell results in a neutral state). Both views lead to an equivalent model of the trading agent, but denoting desired trading states instead of actual activities simplifies description and implementation of this model.

The goal of the reinforcement learning agent in this work is the inference of a suitable trading strategy based on previously obtained forecasts such that it will gain high profits. The input state space comprises predictions from all forecasting modules described in the previous chapter for various time scales. The forecasting periods for each module have been determined through various tests such that they return useful and reliable information in practice. The resulting components of the input space are listed in table 5.2. Note that forecasts from the candlestick module only result in probabilities of binary directions. Thus, the corresponding forecast can be reduced to a single probability value because it contains the opposite direction's probability implicitly. Prognoses from the indicator module are limited to their discrete values without considering the associated probabilities because, as explained in section 4.3.3, the resulting probabilities tend to be so high for nearly all forecasts that they do not provide any additional information.

These input values are used to form a suitable state space through tile coding. The

forecasting module	forecasting periods (days)	provided values
trend lines	5, 30	predicted minimum and maximum percentaged change (continuous value)
indicators	5, 15	predicted direction (discrete value)
candlestick patterns	1, 2, 3	probability of upward movement (continuous value)

Table 5.2: Input space for the RL environment

exact parameter choices for this tile coding will be discussed in the following section.

The reward function for this model is rather obvious: since the agent's goal is to maximize its profits through trading, its daily rewards are determined from the daily percentaged price change $C(t)$ of a security. Whether the agent yields profits or losses obviously depends also on his trading actions. Hence the reward is defined as:

$$r_{t+1} = a_t \cdot C(t) \tag{5.14}$$

$$\text{with } C(t) = \frac{\text{closing price}(t) - \text{closing price}(t-1)}{\text{closing price}(t-1)}$$

This completes the specification of the learning environment: for each time step t the agent receives a tile encoded state description s_t, decides on one of the three available actions a_t and receives a corresponding reward s_{t+1} along with a new state description s_{t+1} afterwards. Note that this problem differs somewhat from common reinforcement learning tasks, because the agent is not able to truly influence the environment. The environment's next state will always be the same no matter what action is chosen by the agent, but the learning problem is not affected by this since the agent is still tasked with learning the optimal state-action pairs.

5.4 Learning and Results

An evaluation of the reinforcement learning process proved to be more complex than evaluating the individual forecasts. First of all, the procedure is highly sensitive to the choice of learning parameters. The parameters that proved to perform best in practice are listed in table 5.3. The most important parameters are the ones related to the tile coding: these parameters determine the size of the environment's state space and are therefore crucial to the learning success. The selected parameters result in a rather coarse coding scheme but tests with varying parameters confirmed the previous considerations about the state space size: a finer coding allows the agent to yield significantly higher rewards when it is trained with repeated iterations on the same training set. However, as soon as the agent is tasked with applying his knowledge to new observations, it performs so poorly that any positive overall reward can be considered lucky. Obviously this finer encoding of the state space yields to an overfitting so that the agent is not able to generalize its knowledge in order to apply it to new situations. Smaller values on the other hand result in such a small state space that the agent is no longer able to grasp key properties of specific observations and thus will never learn a profitable strategy. As discussed before, it is important to set the discount factor γ to a value < 1, because otherwise the agent can determine any action's value only after examining all possible subsequent states which would result in a virtually

parameter	value
α	0.4
γ	0.6
λ	0.5
ϵ	0.15
tiling resolution	line forecasts: 7 indicator forecasts: 3 candlestick forecasts: 3
# tilings	5
convergence criterion	no improvement within 20 iterations

Table 5.3: Selected parameters for the reinforcement learning task

never-converging learning process. The remaining parameters did not exhibit a significant influence on the learning procedure, but rather can be used for fine-tuning the process (e.g., a higher value of ϵ does not alter the results significantly but will lead to a slower convergence). To determine whether the learning process has achieved its maximum, the agent keeps track of the highest previous rewards for a certain learning set. If it is not able to exceed this reward for 20 iterations (i.e., it was not able to learn any new valuable information in that period), the learning process is assumed to be converged and the agent proceeds with the next learning sequence.

In order to evaluate the learned policy's merits, it is necessary to disable the exploration (i.e., set $\epsilon = 0$) while testing with unknown data sets. Otherwise the agent would deliberately choose sub-optimal actions occasionally and therefore the results would be less than optimal. A comparison of testing periods of varying length have shown that it is expedient to limit them to a rather short duration. If test periods are too long, there may be many previously unvisited states contained within and disabling the exploration hinders the learning of any new information for these states. Test periods with a length of 125 days (reflecting approximately half a year, since only business days are considered) resulted in a satisfying performance. After testing a sequence of instances, the same sequence is used to continue learning. It turned out that the agent is able to learn significantly better when he is only tasked with learning on small subsets. This is due to the fact that he can quickly explore various combinations of states if only small sets are considered, while it takes many learning iterations if the whole set is used for learning. On the other hand, if only small time slices are considered for learning, it may happen that a single instance of a certain state in the subset will replace knowledge about the corresponding state-action estimate learned from many previous instances. This could result in a local overfitting and to avoid this effect, the complete set of previously learned instances is considered for updating the learning process every once a while. The complete evaluation proceeds by starting with some initial training, testing of a subset, using the same subset for learning subsequently until the convergence criterion is reached, and then proceeding to the next subset. After each learning sequence of ten subsets, the procedure performs an additional learning task with the complete set so far. This procedure is schematically depicted in figure 5.4.

Note that the resulting evaluation comprises only a subset of the data described in chapter 4. This is due to the fact that the initial 5000 data sets used for training the forecasting modules cannot be employed for the reinforcement learning task, because they do not have inferred forecasts associated. Before the inferred strategies from the reinforcement learning agent can be evaluated, it is also necessary to provide a training set of certain size because otherwise the agent would have no means of exploring the state space sufficiently. Again, an initial training set of 5000 instances proved to be sufficient to allow for a thorough

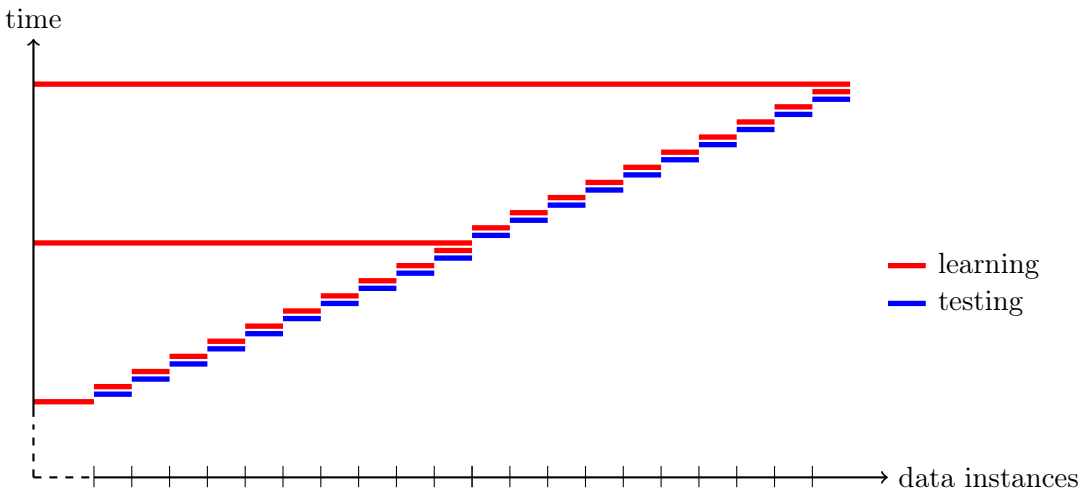


Figure 5.4: Schematic RL evaluation procedure

training of the agent. As a result, the data set used for evaluation is reduced by 10,000 instances compared to the data described in chapter 4. Still, the remaining data covers the time period from 1965 to 2010 and is therefore large enough to show that the agent is able to generalize its knowledge. Testing of only short time sequences bears the danger of producing misleading results because an agent may incidentally perform well in certain situations without being able to adapt to other scenarios. Various tests with different approaches throughout the development of this work have shown that even inadequate procedures may yield profits if tested with distinct short-term periods (i.e., periods of a few years at most), although they will fail to perform well consistently. This can be best illustrated by considering recent developments: Since financial markets worldwide started recovering from the economic crisis in early 2009, nearly all securities experienced magnificent gains recently. By analyzing only the short time period from early 2009 on, one could have selected securities for buying virtually at random and still yield major profits. Obviously, this could lead to impressive gains for many trading strategies, although they might be incapable of performing well in changing market phases, and therefore an analysis of such a short time period might lead to a distortion of the results. The resulting evaluation set presented in this work still comprises roughly 45 years of historical price movements and is therefore large enough to show a general applicability of the inferred strategy.

The result from applying the inferred trading strategy to the test data of the Dow Jones Industrial Average Index is depicted in figure 5.5. This figure shows the accumulated percentaged profits from applying this strategy compared to the percentaged price change of the underlying security. These underlying price changes denote the profits one could have obtained by buying the security at the beginning of the test period and then simply holding on to it (called *buy-and-hold strategy*). This comparison is commonly used to evaluate trading strategies: a strategy is only considered useful, if it is able to outperform the simple buy-and-hold strategy. As it can be seen from the figure, the results from this work are clearly able to meet this criterion and therefore they may be considered useful. Special attention should be paid to the recent years of the test period: it can be seen from the figure that the beginning of the US subprime crisis (marked with the red dashed line) lead to a significant drop in the underlying values while it produced tremendous profits with the inferred strategy at the same time. This clearly shows how the agent was able to learn a correct interpretation of the forecasts and thereby exploit the heavy fluctuations to gain large profits. To explain the fact that the strategy's gain has a significantly larger magnitude than the underlying decrease, one has to keep in

mind that positive accumulated gains grow exponentially and that figure 5.5 only has a rather coarse resolution. In fact, the actual movements exhibit many small fluctuations (as were depicted in various figures before), which were exploited by the trading agent such that its resulting gain has a larger overall percentaged change than the underlying prices. Further analysis of the results reveals that profiting from crises is no isolated incident for the learned strategy. Previous crises (the Black Monday Crash from October 1987, marked with the brown line, and the New Economy Crisis from the early 2000s, marked with the turquoise line) also resulted in significant increases of the trading agent's profits. Obviously, the learned trading strategy performs especially well in crisis situations. This can be explained through the fact that such situations comprise short-term drops of the underlying price movements with a significantly higher magnitude than usual changes. If the pending direction was predicted correctly, the trading agent is therefore able to score major profits with few actions.

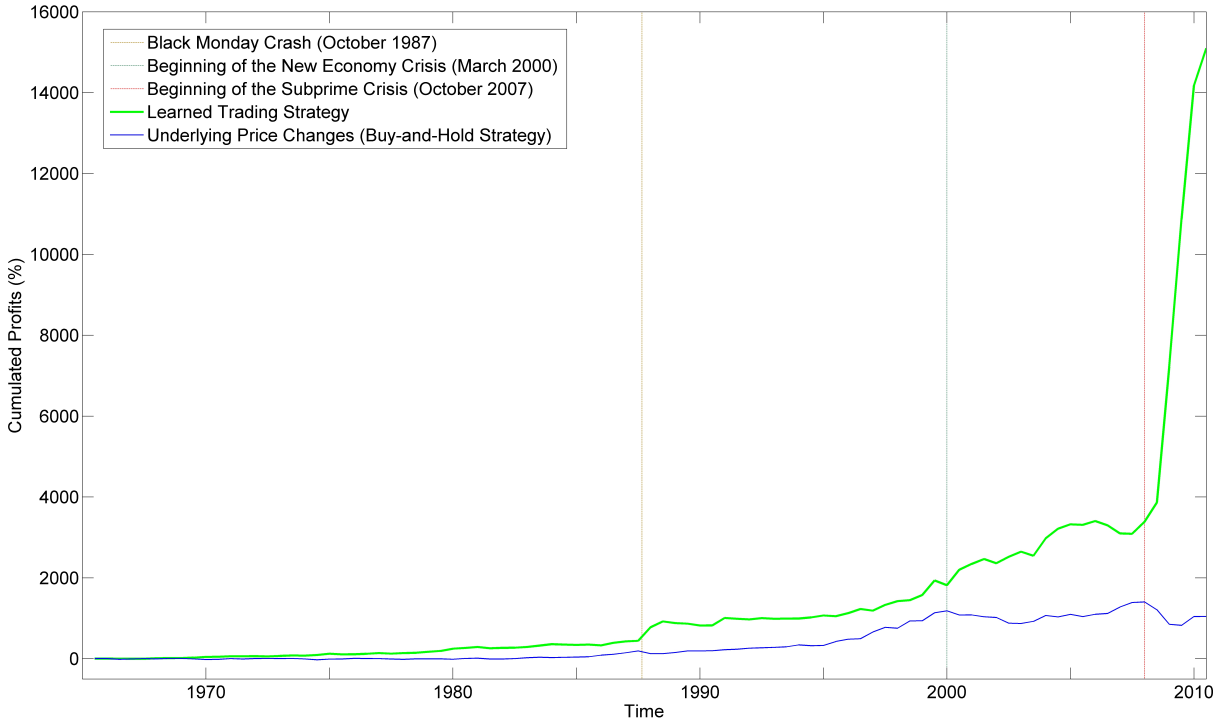


Figure 5.5: Resulting profits for the Dow Jones Industrial Average Index with the inferred trading strategy. For comparison, the actual price changes (representing a buy-and-hold strategy) are also depicted.

6 Conclusion

6.1 Summary

Goal of this work was an investigation and subsequent implementation of machine learning approaches to facilitate a technical analysis of financial markets. The results from chapter 4 clearly show that it is in general possible to successfully employ machine learning processes in order to obtain forecasts of future price movements, although it was discovered that not all analysis methods are equally well-suited for all purposes. The analysis of trend lines proved to yield useful forecasts with respect to direction and magnitude of future price changes and exhibits a satisfactory precision. Prognoses based on an indicator analysis are able to predict future directions with an impressive success rate but are less informative compared to the line-based forecasts due to the lack of magnitude information. An analysis of candlestick patterns proved to yield forecasts with an almost arbitrary scalable success ratio but it has been shown that obtaining prognoses with the highest reliability does not necessarily result in maximum gains. The only analysis method that proved to be unfeasible for a statistical learning process is the theory of point patterns.

The candlestick module performs best with very short-term forecasts, the indicator module is able to infer forecasts on a short-term to mid-term range, and, while the trend line module does not perform well on very short-term forecasts, it is able to extend forecasts farther into the future compared to the indicator module. As a result, the forecasts obtained from the different modules supplement each other and should be analyzed simultaneously in order to decide on trading actions. Chapter 5 has shown that such a combination of forecasts from different analysis modules and for varying time scales can indeed be used to facilitate learning of a profitable trading strategy.

6.2 Outlook

A highly useful extension to the presented model would be some kind of risk estimation. From the presented forecasting processes, the candlestick analysis module is the only one that is informative with respect to its reliability and therefore to the risk associated with reaching decisions based on its forecast. The trend line module is able to give risk estimates to a very limited extend as it yields forecasts with respect to both minimum and maximum expected change in price values. By comparing both forecasts one can obtain a rudimentary estimation of the associated risk, i.e., if both forecasts exhibit similar values the associated risk can be considered small while large deviations obviously reflect uncertainty and therefore a rather high risk. However, this is far less informative than information about the certainty associated with each forecast. The indicator module lacks any information on potential risk because it tends to yield forecasts with associated probabilities that indicate nearly certainty. An augmentation to the presented modules such that useful information on all forecasts' associated risks is provided would provide the reinforcement learning agent with another dimension of information that possibly might enhance its profits significantly by avoiding risky trades.

Another improvement might be obtained by employing further technical analysis meth-

ods. The research presented so far has been limited to a selection of popular methods, but there are several additional methods available which have not been explored due to the limited time frame of this work. For instance, one could search the historic data for cyclic movements. It has been observed that markets are subject to certain cycles and therefore if one could identify characteristic cycles in the past and, by matching the current situation with such a known cycle, one could potentially forecast future movements. An example of a cyclic movement could be the assumed correlation between American financial markets and the US presidential elections: it has been observed that prices exhibit rather strong increases in the pre-election year while they tend to decline in the post-election year. Due to the modular structure presented in this work, additional forecasting modules can be easily added by implementing them separately and include their resulting forecasts into the input space of the reinforcement learning environment model. Such an approach could possibly also explain (and eventually resolve) the observation from section 4.3.3 that forecasts from the indicator module exhibit certain time periods with lower success rates.

All considerations so far were only concerned with analyzing individual securities. A completely different dimension could be introduced by analyzing the correlations of various securities. To name a few examples of such an analysis, one could for instance compare the stocks of similar companies. If these companies are active in the same area of business and have exhibited similar stock price movements in the past, one could use these similarities to infer forecasts. If stock prices of these companies diverge at some point there is reason to assume that they will converge again in the near future and therefore one would predict the higher-priced stock to decline and the lower-priced stock to rise. This concept has been employed in practice for several decades and is called *statistical arbitrage*. Another example is the analysis of inter-market correlations. For instance, stock markets exhibit a considerable risk while gold is considered to be a very reliable investment. Hence it can be observed that an increasing insecurity in stock markets leads to a rise in gold prices. This may be used as an early indicator as a rising gold price might hint at future declines of stock markets and vice versa. Enabling a machine of learning such correlations autonomously obviously requires rather complex and sophisticated models and is therefore a task that offers plenty of future research opportunities.

Bibliography

[Bao08] Depei Bao. A generalized model for financial time series representation and prediction. *Applied Intelligence*, 29:1–11, 2008. 10.1007/s10489-007-0063-1.

[BCG03] Christophe Biernacki, Gilles Celeux, and Gérard Govaert. Choosing starting values for the em algorithm for getting the highest likelihood in multivariate gaussian mixture models. *Computational Statistics & Data Analysis*, 41(3-4):561 – 575, 2003.

[Bil01] Jeff Bilmes. Graphical models and automatic speech recognition. Technical Report UWEETR-2001-0005, University of Washington, Department of Electrical Engineering, 2001. `https://www.ee.washington.edu/techsite/papers/refer/UWEETR-2001-0005.html`.

[BS73] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[BY08] Depei Bao and Zehong Yang. Intelligent stock trading system by turning point confirming and probabilistic reasoning. *Expert Systems with Applications*, 34:620–627, 2008.

[CGH96] Enrique Castillo, Jose M. Gutierrez, and Ali S. Hadi. *Expert Systems and Probabilistic Network Models*. Springer, 1 edition, December 1996.

[Cha95] Tushar S. Chande. The time price oscillator. *Technical Analysis of Stocks and Commodities*, 13(9):369–374, 1995.

[CHYL97] Seng Chou, Hsien Hsu, Chau Yang, and Feipei Lai. A stock selection dss combining ai and technical analysis. *Annals of Operations Research*, 75:335–353, 1997.

[Den08] Min Deng. On the Theoretical Foundation of Technical Analysis: Market Action Discounts Everything. *SSRN eLibrary*, 2008.

[DP97] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss, 1997.

[DVR07] José G. Dias, Jeroen K. Vermunt, and Sofia Ramos. Analysis of heterogeneous financial time series using a mixture gaussian hidden markov model. *Preprint submitted to Elsevier Science*, 2007.

[enc09] *Encyclopaedia Britannica 2010*. Encyclopedia Britannica, 15 edition, September 2009.

[FW94] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48(3):533 – 551, 1994.

[GLL07] Xinyu Guo, Xun Liang, and Nan Li. Automatically recognizing stock patterns using rpcl neural networks. *Proceedings of the 2007 International Conference on Intelligent Systems and Knowledge Engineering*, October 2007.

[HW99] Jeff Harrison and Mike West. *Bayesian Forecasting and Dynamic Models*. Springer, Berlin, 2. auflage. edition, April 1999.

[HY01] David J. Hand and Keming Yu. Idiot's bayes - not so stupid after all? *International Statistical Review*, 69(3):385–398, 2001.

[JL95] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. *Proceedings of the Eleventh Conference on Uncertainty in Artifical Intelligence*, 1995.

[Lam83] Donald R. Lambert. Commodity channel index: Tool for trading cyclic trends. *Technical Analysis of Stocks and Commodities*, 1(5):120–122, 1983.

[Leo06] Kong Cheong Leong. the empirical study of applying technical analysis on dji, hsi and taiwan stock market. Master's thesis, National Sun Yat-Sen University, No. 70, Lienhai Rd., Kaohsiung 80424, Taiwan, R.O.C., June 2006.

[Man63] Benoit Mandelbrot. The variation of certain speculative prices. *Journal of Business*, 36:394, 1963.

[MK96] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. Wiley-Interscience, 1 edition, November 1996.

[Mor06] Gregory L. Morris. *Candlestick Charting Explained: Timeless Techniques for Trading Stocks and Sutures*. Mcgraw-Hill Professional, 3rd ed. edition, May 2006.

[Mur99] John Murphy. *Technical analysis of the financial markets : a comprehensive guide to trading methods and applications*. New York Institute of Finance, New York, 1999.

[Mur02] Kevin Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, 387 Soda Hall, Berkeley CA 94720, July 2002.

[Mur07] Kevin Murphy. How to use the bayes net toolbox. `http://bnt.googlecode.com/svn/trunk/docs/usage.html`, 2007.

[MW07] Guojun Gan; Chaoqun Ma; and Jianhong Wu. *Data Clustering: Theory, Algorithms, and Applications*. SIAM, Society for Industrial and Applied Mathematics, May 2007.

[Nis91] Steve Nison. *Japanese Candlestick Charting Techniques: A Contemporary Guide to the Ancient Investment Techniques of the Far East*. New York Institute of Finance, March 1991.

[Nis03] Steve Nison. *The Candlestick Course*. Wiley & Sons, June 2003.

[Pat10] Scott Patterson. *The Quants: How a New Breed of Math Whizzes Conquered Wall Street and Nearly Destroyed It*. Crown Business, February 2010.

[Pea88] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[PR02] Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002.

[PWZP00] Chang Shing Perng, Haixun Wang, Sylvia R. Zhang, and D. Stott Parker. Landmarks: a new model for similarity-based pattern querying in time series databases. In *ICDE*, pages 33–42, 2000.

[Ris01] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI-01 workshop on "Empirical Methods in AI"*, 2001.

[RN02] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2 edition, December 2002.

[Rud08] Dr. Tamas Rudas. *Handbook of Probability: Theory and Applications*. Sage Publications, Inc, 1 edition, February 2008.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The Mit Press, May 1998.

[Voi06] Michael Voigt. *Das große Buch der Markttechnik : Auf der Suche nach der Qualität im Trading*. FinanzBuch-Verlag, München, 2006.

[Wat89] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge,England, 1989.

[Wil78] J. Welles Wilder. *New Concepts in Technical Trading Systems*. Trend Research, June 1978.

List of Figures

List of Tables