SEARCHING SIGNIFICANT PATTERNS IN FINANCIAL TIME SERIES DATA USING MACHINE LEARNING TECHNIQUES

Research done by Nikita Naychukov

Faculty advisor: Dr. Igor Goncharenko

Abstract

Financial data, as time series of stock prices and similar, have permanently been assessed as very noisy, disorganized, and non-stationary and, therefore tend to be unpredictable by most conservative models in the light of which financial time series do not contain any patterns to follow. Moreover, most modern researches attempting to reveal some patterns found only weak evidences of patterns existence. Hence, the question whether financial data contain any significant regularities is not resolved yet by using traditional and modern methods which provokes a need in more advanced search instruments. In recent times advanced techniques in time series pattern recognition and image classification become more and more popular among scientists which can also be adapted for research in financial time series. These modern techniques include “shapelets” recognition, Dynamic Programming-based partial matching algorithms, up-to-date similarity-invariant pictures’ descriptors which allows to compare objects invariant to rotation, translation and scale, and fast k-medoids clustering. By reasonably applying them it is possible to test whether these tools could help improve regularities search in financial time series.

In this thesis, financial time series data are examined on the purpose of containing significant patters. The main tool for this purpose is an automated algorithm for searching significant patters which uses new approach to treat financial time-series as sketch-based pictures, and compare them by applying machine learning tool pack. The results we obtain support our belief that a combination of modern advanced methods allows to extract statistically significant patterns from financial time series.

Table of Contents

Introduction

The problem of retrieval significant patterns from financial time series stays rather actual and becomes increasingly important in the light of practical applications and academia.

On the hand of practical usefulness, it is worth to mention technical analysis (TA) which is predicting future stock movements relying on historical prices, volumes and other market data. Generally, a technical approach to financial time series analysis exhibits an idea that prices follow a particular shape determined by investors' characteristics which are formed from their attitudes to different economic, social and political factors (Pring, 2002). The greater part of technical analysis consists of determining a currently realizing particular pattern in financial time series which commonly represents some kind of a figure, a shape such as "head-and-shoulders" or "double tops". Basing on such patterns investors can predict further movements of prices which opens a huge possibility to make profit.

However, the main issue is to correctly detect such a pattern. There are some significant works in this sphere where researchers evaluate TA using computing algorithms and statistical tests to capture specific technical patterns and prove their existence in financial time series (Lo, Hasanhodzic, 2010). The findings of these studies provide an incentive for further investigation because it states that there are several patterns which statistically distinguishable in financial data. In addition, there are a lot of work can be done on optimizing pattern recognition algorithms in order to increase the efficiency of regularities identification. Speaking about traditional non-scientific TA methods which include a large amount of visual tools to discover a realizing pattern, most of them are based on a subjective judgement of an investor which is not scientifically accepted.

On the hand of academic finance, we can pay attention to classical methods for analyzing financial time series such as regressions of various types (linear regression models, non-liner regressions, non-parametric regressions, autoregressive models such as AR, MA, ARMA, ARMIA, autoregressive models to model time-series with time-varying volatility clustering such as ARCH, GARCH, etc.) which provides a possibility to fit financial time series accounting for different factors (Tsay, 2005), (Mills, Markellos, 2008). However, these models are not specifically used for the case of pattern retrieval, they are generally applied in order to account for some factors in future price movements. Moreover, these models are rather difficult to practically implement when dealing with indeed huge datasets because of a restrictive nature of

their assumptions and a large amount of possible influencing factors. Other traditional and common models to account for inefficiencies in financial time series are CAPM and APT. These models are closer than statistical ones in resolving our task of revealing patterns in financial data in a sense that they also indicate kind of patterns in a form of capturing risk-free profits and inefficiencies in financial markets. However, these models fail to be rather applicable due to restrictive assumptions and consideration of too noisy end-of-day prices data (Fama, French, 2004).

Hence, taking into account abovementioned information, the research of a new method of financial time series patterns retrieval has a practical value and a scientific importance. It potentially will increase current regularities search efficiency which opens a possibility for investors to make profit using retrieved patterns for further predictions. The applications for such an algorithm are widespread enough including active trading players such as hedge funds, proprietary trading departments of banks and millions of passionate individual investors. With respect to academia, the research of a model that efficiently captures market inefficiencies identifying patterns in financial time series data leads to a significant step in finance because traditional methods fail to account for any regularities in data or even they are not targeted at this goal, and most modern current methods are not efficient enough to distinguish many statistically significant patterns.

In this paper we try to test the hypothesis that financial time series contain statistically significant patterns by means of building an automated algorithm for patterns retrieval that uses a novel approach of patterns identification. This approach combines a bunch of machine learning tools which allows the algorithm to distinguish patterns and separate them into clusters without human interruption. We use a dataset that consists of thirty stocks from Dow Jones Industrial Average index, and compare obtained patterns with regularities ejected by the same algorithm from another dataset of random prices that follow Brownian motion.

The paper is built as follows. In chapter one we provide a brief overview of relevant literature concerning the foundations of technical analysis and automating it by means of computational algorithms, time series clustering using “shapelets” and their various modifications, and recent methods of pattern recognition and classification of images. Chapter two gives a detailed overview of the dataset, and preparing it to the research, because we treat selection of “right” data as very important aspect in considering pattern recognition task. Chapter three introduces the model and associated methods in data mining. The further chapter presents the application of our model to pure noisy dataset based on Brownian motion in order

to test whether our model is capable to extract significant patterns. Finally, the last part of the paper summarizes the results of the research.

In the annexes we provide the coding tips for people willing to make research on this topic or to further develop this study, using R programming language.

Chapter 1. Literature Review

Technical analysis was invented a long time ago, approximately in XVIII century in Japan by Munehisa Homma, in a form of representing rice prices by candlestick chart. Munehisa Homma explored some patterns in prices such as they go down after bullish market period and go up after bearish market period. That allowed him to make significant profit and to become a financial adviser. After that moment, next significant step in TA development played Dow's axioms published at the end of XIX – Charles H. Dow stated that prices reflect their fundamental value and in order to make any predictions about their future movements it is enough to look at current market trends. In XX century TA became relatively popular among investors who actively practice their methods on real markets which provoked emergence of works that gathered all TA concepts of that time in one place and formed first rules of TA (Schabacker, 1930). After that point TA became a widespread instrument for many investors worldwide and agents using TA worked out many techniques which determined current TA approaches. Generally, there are three types of TA (Bechu, Bertrand, 1999): a traditional one which is based on visual perception of data and study of shapes in financial time series; a contemporary TA which relies greatly on quantitative methods, such as various forms of a moving average, in order to determine a realizing pattern and predict future price changes (Taylor and Allen, 1992); a social one which tries to describe price movements through agents' behavior. Mostly, it is useful to combine different TA types together in order to correctly capture a figure, a trend, and forecast how it will behave further. For that purpose, many practitioners nowadays use computer programs which are rather popular: for example, UBS proposes to use their online service 'KeyInvest" for applying TA online, or there is a well-known cloud service "TradingView" to try and share new tools of TA.

Despite the fact of the existence of various online platforms and that TA had many applications during XX century and various investors types, including brokers, dealers, investment funds, hedge funds, individual traders, used TA for trading purposes on all kinds of

markets, including currency (FOREX), commodity, and stock markets, (Lui, Mole, 1998), (Cheung, Chinn, 2001), there are not really many researches on that topic which probably might be connected to a high portion of skepticism devoted to TA in academia. This trend is mainly attributed to an academically determined notion of market efficiency according to which current prices already contain all possible information and, therefore reflect their fundamental values (Malkiel, Fama, 1970). Furthermore, the concept of market efficiency was narrowed to three forms of market efficiency: weak form efficiency when prices reflect all past prices historical information, semi-strong form efficiency when prices reflect all current public and past prices historical information, and strong form efficiency when prices reflect all available information including any private sources (Jensen, 1978). Hence, according to an academic view, if prices are fairly determined which means markets are efficient, there cannot be any inefficiencies in financial times series such as retrieved patterns by means of which it is possible to make risk-free profits. Therefore, it is seen how tight the connection between our task of pattern recognition in financial time series and the concept of market efficiency. However, our work is not aimed at rejecting the market efficiency hypothesis because the possibility of pattern identification tells only about that at some particular point in time regulators fail to make market conditions such that prices contain all possible information and financial time series inefficiencies occur.

Nevertheless, there are some researches in this field. One of the first significant works "The Predictive Significance of Five-Point Chart Pattern" was done in 1971 by Robert A. Levy who tested forecasting power of thirty-two five-point chart patterns based on extrema identification in financial time series. The dataset for this work included prices of five hundred stocks for the period from 1964 to 1969 traded on New-York Stock Exchange (NYSE). The results were academically predicted in a sense that patterns retrieved were not significant so that further trading strategy based on them were not able to produce profits. The next noticeable work in the field was a research of Chang and Carol L. Osler in 1999 on graphical models applications in exchange rate forecasting. They stated that exchange rates were formed irrationally which created a possibility for identifying of patterns. They decided to describe their financial time series using the graphical pattern "head-and-shoulders", where their dataset consisted of daily spot exchange rates of six currencies – Canadian dollars, British pounds, German marks, Japanese yens, francs, and Swiss francs – for the period since 1973 to 1994. Chang and Osler used bootstrap method (Efron,1982) to test whether their retrieved patterns were significant enough. This method includes simulating some amount of random time series

with the same parameters as the original dataset has (length, mean, variance, skewness and kurtosis) and applying the model which were originally applied to a random dataset. In other words, Change and Osler tried to retrieve "head-and-shoulders" pattern from the original dataset and from a random one, and compare two distributions, using some statistical goodness-of-fit test. The results indicated that there was a significant difference in two distributions only in two cases for German marks and Japanese yens. Another similar research was conducted by Caginalp and Laurent in 1998 where five hundred stocks from S&P500 index were evaluated for the period from 1992 to 1996. Authors searched not only one pattern "head-and-shoulders", but all reversal patterns (after pattern is realized prices follow a reverse direction) in the dataset and found a significant difference in patterns distributions between the real dataset and a random one.

The main research on academic approach to TA is "Foundation of Technical Analysis: Computational Algorithms, Statistical Inference, and Empirical Implementation" written by Lo, Mamaysky and Wang in 2000. This was the first work on the topic of TA which considerable changed the perception of TA by academia. In this work Authors tested a vast amount of TA patterns by means of automated algorithm which recognizes a bunch of regularities ("head-and-shoulders" and "inverted head-and-shoulders", "broadening tops/bottoms", "triangle tops/bottoms", "rectangle tops/bottoms", and "double tops/bottoms"), in financial time series. Lo, Mamaysky and Wang proposed a more structured approach to financial time series pattern recognition using nonparametric kernel regression and their algorithm on the dataset of U.S. stocks from 1962 to 1996. To verify the results, they used Kolmogorov-Smirnov test on two distributions: the unconditional empirical distribution of daily stock returns to the distribution conditioned on specific pattern such as "head-and-shoulders" or "double tops". They found that five (head-and-shoulders", "broadening bottoms", "rectangle tops/bottoms", and "double tops") of ten considered technical patterns are contained in financial data indeed and it can provide incremental information for investors which can be used for making trading strategies.

To sum up, the abovementioned researches proved that the topic of patterns recognition in financial time series is worth to notice. Moreover, many studies retrieved some patterns from the datasets they used which indicates that further research in this filed can potentially bring fruitful results. Although many studies approved regularities existence in financial data, each particular result is restricted to specific markets, bunch of patterns, and a period of investigation. Furthermore, most results are weak enough to conclude that the difference in pattern distributions between original datasets and random ones is significant.

Therefore, in this paper we apply more advanced methods in order to try to retrieve patterns from financial times series data and prove that these patterns are significantly different from a random noisy dataset. By means of latest innovations in the fields of image classification and time series pattern recognition which vastly use machine learning tools we want to increase efficiency of patterns retrieval algorithm which allow to avoid the need to specify which particular patterns we look for, and which markets and periods of time we investigate.

Hence, it is necessary to introduce some relevant articles from these fields we use for our research. One of the first motivating articles for this research was "Time Series Shapelets: A New Primitive for Data Mining" written by Eamonn Keogh and Lexianh Ye published in 2009. Authors introduces a novel approach for classification of time series, they introduce the notion of time series "shapelet" which is a subsequence of a time series which is distinguishably representing of a whole class of similar time series. They provide an interesting example of the task to classify the leaves of two species. The leaves are almost identical – they have a very similar form and the same color – so considering the entire time series is useless and the only method to classify them is to find some distinguishable piece of their shape which is different between two species. Hence, authors depict each leave into one-dimensional representation such that they obtain a time series of a shape. And then they found a small subsequence of this time series which is the same for all leaves of one species and different for all leaves of the other species. They call such this subsequence a "shapelet" and it is a small subsequence which is a distinct feature of a whole class. The main novelty of the method is in that they do not consider the entire time series of the shape to classify the leaves. Authors test their model on various datasets including historical documents such as coats of arms or heraldic shields, projectile points (arrowheads), and many more objects which are possible to cluster according to their shape. As a result of using "shapelets" for clustering tasks authors obtain significant reduction in space and time needed, more accurate classification results and interpretable results in comparison to other state-of-the-art classifiers.

This article captures our interest in several following aspects: Keogh and Ye research shape retrieving and classification which is similar to our task of financial time series pattern recognition because patterns are also shapes; authors consider investigation of very noisy data which is similar to financial time series data; they provide a framework for identifying "shapelets" proposing an algorithm for "shapelets" extracting and further time series classification which we partially can adapt to our research. However, our task is slightly different in that we do not classify the entire time series by identifying a small "shapelet" in it,

our purpose is to retrieve some repeating subsequences of a distinguishable form from a whole dataset which are generally called patterns. Hence, we do not adapt the entire model from this article, but gain some valuable insights of clustering algorithms and dataset preparations, which will be discussed further.

The other prominent work in time series clustering which affected our research is "Clustering Time Series using Unsupervised-Shapelets" written by Jesin Zakaria, Abdullah Mueenm, and Eamonn Keogh published in 2012. This work continues time series clustering model introduced in previous article but presents a new notion - "u-shapelet". "U-shapelet" is the same subsequence as "shapelet" in all aspects except for that it might be used without prior knowledge of classes into which objects should be clustered, so "U" stand for "unsupervised". This feature is incorporated by using a distance map of deliberate "u-shapelets" to all time series in a dataset which allows to divide objects into corresponding clusters according to comparing their distance, while "u-shapelets" choice for such a procedure is determined by maximizing gap between classes such that objects from one class are much closer to "u-shapelet" of this class than to "u-shapelet" of the other class. Authors demonstrated that their method is again more accurate among other clustering models with unlabeled original data.

The last work on time series clustering which influenced our research is "Scalable Clustering of Time Series with U-Shapelets" written by Liudmila Ulanova, Nurjahan Begum and Eamonn Keogh published in 2015. Authors introduced a new approach to speed up a "u-shapelet" clustering process by two orders of magnitude. They present "scalable u-shapelet" (SUSh) – a hash-based algorithm that allows to make the model faster.

The next area which is extensively used in this paper is image pattern recognition and classification. Hence, it is important to describe a recent work in this sphere ideas from which we apply in this research. The article "Sketch-based Image Retrieval from Millions of Images under Rotation, Translation and Scale Variations" written by Sarthak Parui and Anurag Mittal published in 2015. We appeal to this article because our main task is to identify patterns which can be geometrically, visually presented as a particular figure or a shape. Therefore, we can treat them as images or namely, as sketch-based pictures. Hence, we need to know most up-to-date methods of pictures retrieval and classification. The article proposes a retrieval method which is invariant to rotation, translation, and scale (invariant to all similarity transformations) that exactly satisfies our need because we want to extract similar patterns like "inverted head-and-shoulders and head-and-shoulders" and attribute them to one cluster. Authors suggest to represent object boundaries (an object is an image) as chains of connected segments that can be

easily described analytically by similarity-invariant length descriptors. Using these descriptors authors can calculate a similarity score between any two objects even of different lengths using Dynamic Programming-based partial matching algorithm. Then they propose to classify objects according to obtained distances using k-medoids machine learning clustering algorithm. The results that authors got applying their method for image recognition and classification demonstrate superiority over rival state-of-the-art methods.

The proliferation of modern pattern recognition and clustering techniques and their success in many fields convince us to take advantage of using them for our task. Furthermore, taking into account the conceptual similarity of data described in the articles and financial time series data it is reasonable to apply these methods in financial pattern recognition. Considering that most of these modern methods were not applied to financial time series before and weakness of current results using automated TA models, we tried to compile them into our model. However, building the model which combines updates from the articles without modifications do not satisfy our endeavors. Hence, we found out that the data used in the article (Lo, Mamaysky and Wang, 2000) which include only stock returns is too noisy for pattern recognition task and their restriction to use only ten particular TA figures significantly limits the test whether financial time series generally contain any patterns. The model proposed in the articles (Keogh, 2009, 2012, 2015) produces poor recognition and classification results when used as a standalone one due to extra noise peculiarity of financial data and probable absence of any patterns incorporated in time series, moreover, we want to compare "shapelets" of different length (patterns in our case) not to original time series but between each other, and for this reason we need some more advanced distance measure than the Euclidean one. Finally, advanced image processing instruments proposed in (Mittal, 2015) are not directly applied for times series comparison due to the fact that they measure similarity score (not a distance), and the Mittal's score functions are exponentially defined which brings too large order of numbers for our task, and the author's Dynamic Programming-based partial matching algorithm is defined for non-contiguous shapes, hence, it needs further modification for our task of comparing contiguous figures. Therefore, we build the model which uses high and low prices which are significantly less noisy and more informative, the algorithm that do not restrict patterns search in terms of a shape or an amount, the modified version of Mittal's distance measure that is a distance (not a score), that is linearly defined, which allows to compare objects of different length unlike the Euclidean one, and uses Dynamic Programming matching algorithm for contiguous objects. Generally speaking, our model for financial time series pattern

recognition uses machine learning techniques that learns to find and cluster regularities in time series without human interruption.

Chapter 2. Description of Data

In this research we use thirty U. S. stocks which are constituents of Dow Jones Industrial Average Index for the period of nine years from the 1st March of 2007 to the 13th June of 2016. The source we use to download the dataset is Yahoo Finance which is conveniently reachable through R packages which allows to download the dataset right to the working environment. The chosen companies represent the most prominent industries of the U.S. economy and their stocks are of different liquidity such that we can gather most reliable results because liquid stocks rarely contain patterns, while illiquid ones contain them more often.

The initial dataset consists of daily data of six parameters: opening prices, high prices, low prices, closing prices, volume, and adjusted close prices for the abovementioned period of time. As it was briefly mentioned in previous sections in this paper we use high and low prices because these parameters more accurately reflect the real price movements. They contain significantly less noise than closing prices or stock returns do. Moreover, another reason for this choice is accounting for volatility in the process of the dataset normalization which will be discussed next.

When the dataset is obtained, the next step is to normalize it and to obtain final objects which we will use for the pattern recognition algorithm. Only high and low prices are of our interest, that is why we do all further manipulations mainly with them and treat them as high and low prices time series (H and L time series).

Firstly, we need to make H and L time series adjusted for any corporate actions or distributions as it was already done with closing prices which helps us avoid abrupt price movements. Then, we need to get subsequences of different lengths from H and L time series for further pattern investigation. The length of a subsequence which can potentially capture a pattern is thirty-five trading days (Lo, Mamaysky, Wang, 2000). We decided to investigate three different lengths of ten, twenty and thirty-trading days which is more reasonable for active traders and modern TA methods. Subsequences are obtained by sliding a window of length l_i, where $l_i = \{10, 20, 30\}$ for each corresponding $i = \{1, 2, 3\}$, across H and L time series (Ye,

Keogh, 2009). Hence, we get $\sum_{i=1,2,3}(N - l_i + 1)$ subsequences, where N – is a size of H or L time series.

Then, in order to make these subsequences comparable to each other we apply simple z-score normalization process to every subsequence, for example, for high prices subsequences (H subsequences) $z = \frac{x-\bar{y}}{\hat{\sigma}}$, where x – is high prices in a subsequence, $\bar{y}$ – is the mean of adjusted closing prices in a corresponding subsequence, $\hat{\sigma}$– is the Parkinson volatility estimator which is depicted on the expression 1. The mean of adjusted closing prices is used because we need to subtract the same value from both H and L prices in order to save the relation that high prices are always higher that low ones. We consider the Parkinson estimator for volatility which uses high and low prices instead of closing prices because it is five times more efficient than the close-close estimator.

$$\hat{\sigma} = \sqrt{\frac{1}{4Nln(2)} \sum_{i=1}^{N} \ln\left(\frac{h_i}{l_i}\right)^2} \quad (1)$$

After normalization process we are ready to combine H and T subsequences into final objects which we will research. Analytically, any object is represented as a vector of numbers (high prices, low prices), for example, for the case of twenty days' subsequence, $v_1 = (60.09, 58.23 \ldots 57.85, 56.91 \ldots)$. Graphically, the same object is depicted below, where a black line is a H subsequence, while a red line is a L subsequence:

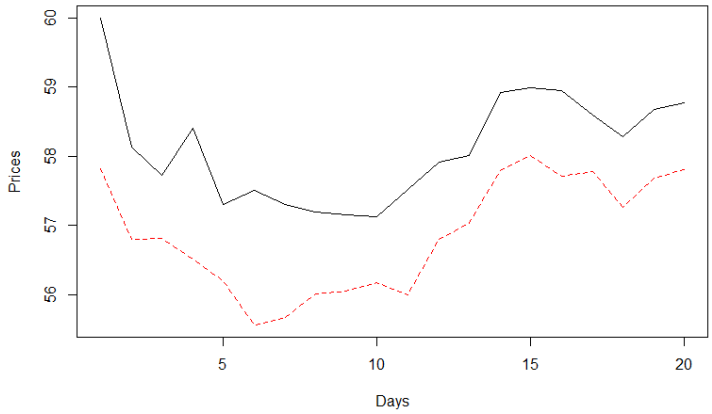

Figure 1: The example of a final object

Hence, we acquire a number of such objects of different lengths (10, 20, and 30 trading days) and store them in a dataset.

Finally, as we are interested only in complicated patterns like TA uses, we need to clean our dataset from simple trends which represent almost straight lines in different rotations like:

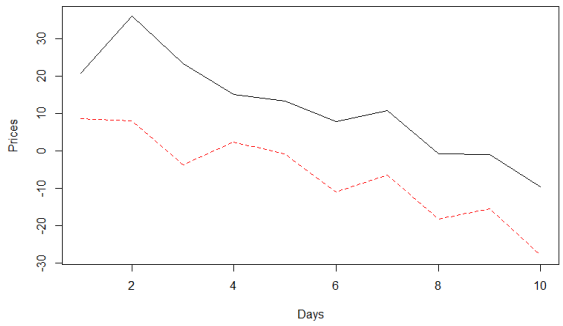

Figure 2: The example of a simple trend

For this purpose, we use a simple cleaning algorithm which uses a linear approximation function that fits an object with some degree of accuracy known as R^2. Namely, we make linear approximation of every object in the dataset with $R^2 = 0.95$ which empirically shows good results in separation sophisticated objects from simple trends. Then, for every object, we look how many points are left after linear approximation, if there are less than seven points, we remove such object and vice versa. Actually, the simplest trend such as a straight line can be described by four points like on the Figure 2, but there are many ones which contain more than four points after our linear approximation process. Graphically, the linearization process is depicted as follows:

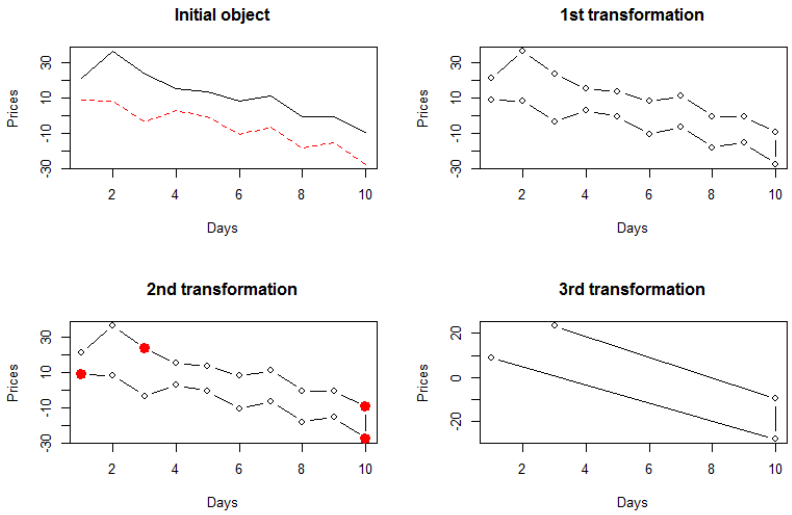

Figure 3: Linear approximation process

Our cleaned dataset contains 197338 objects of the Figure 1 type, while 14060 objects were considered as simple trends and removed from our dataset. Eventually, we obtain the dataset which contains on the greater part sophisticated subsequences which potentially can contain "head-and-shoulders"- like patterns and so on.

Chapter 3. Model

The entire work is coded in R programming language using "R studio" and can be intuitively divided into several parts as they follow in our code: downloading data from Yahoo Finance, creating the dataset, cleaning the dataset with the linear approximation process, and applying the clustering algorithm to the dataset. The first parts were already described above because they are relatively fast and easy and mainly devoted to data preparation tools, while the last part is the main concern of this paper which deals with searching patterns in financial time series using machine learning techniques.

To begin and be clear, we treat the clustering algorithm and the model as the same thing, and firstly, let us consider a simple description of the model to understand its structure in order to follow it for the further detailed exploration. In simple words, the model works as follows: it takes our dataset of objects, represents them in a special format, measures distances, and clusters them according to obtained distances. Strictly speaking, the clustering algorithm is a bit more complex and works slightly different, but at this point we want to discuss its basic elements in order to create a clear vision of the model, and at the end we will give a detailed description of how this algorithm actually works. The flow of algorithm's operation is not random. Namely, to cluster really similar objects we need to compare them by some criterion as for example, distance. This distance should be small for similar shapes and large for different ones because our task is to search for patterns, figures. Moreover, the distance should be similarity-invariant which means that it can detect the same object even if it is rotated, translated or changed in scale. However, in order to calculate such a distance, the objects should be represented in a special format which is invariant to translation, rotation and scale.

Hence, the first element of the algorithm is representation of objects in the special format. This idea is adopted from Mittal's work of 2015. This format is an object's representation in a form of a chain, which almost looks like as our initial object except that now it is a unite shape:

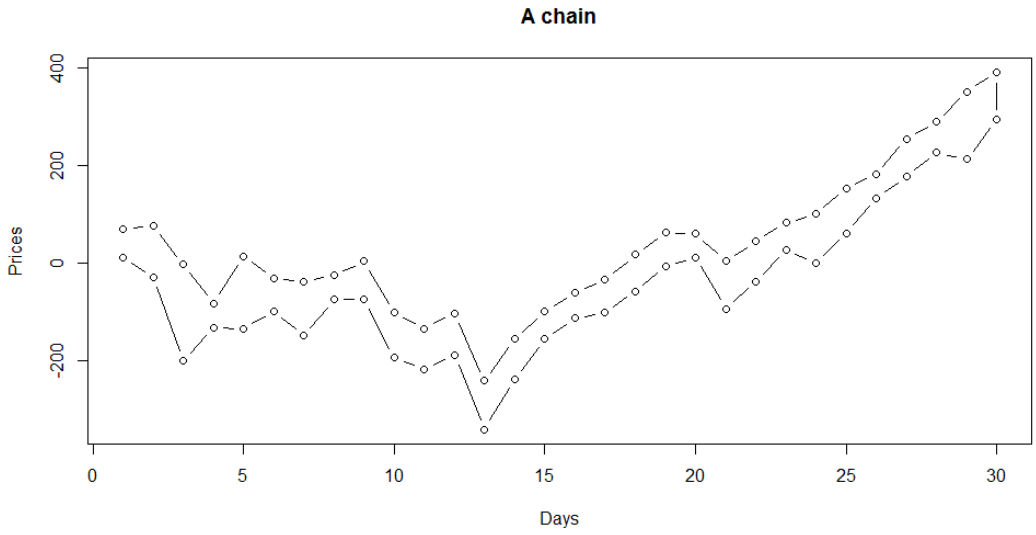

Figure 4: A chain

Then we need to describe the object's shape such that rotation, translation and scale change do not affect our description. For this reason, we separate the chain into small pieces called joints. Graphically:

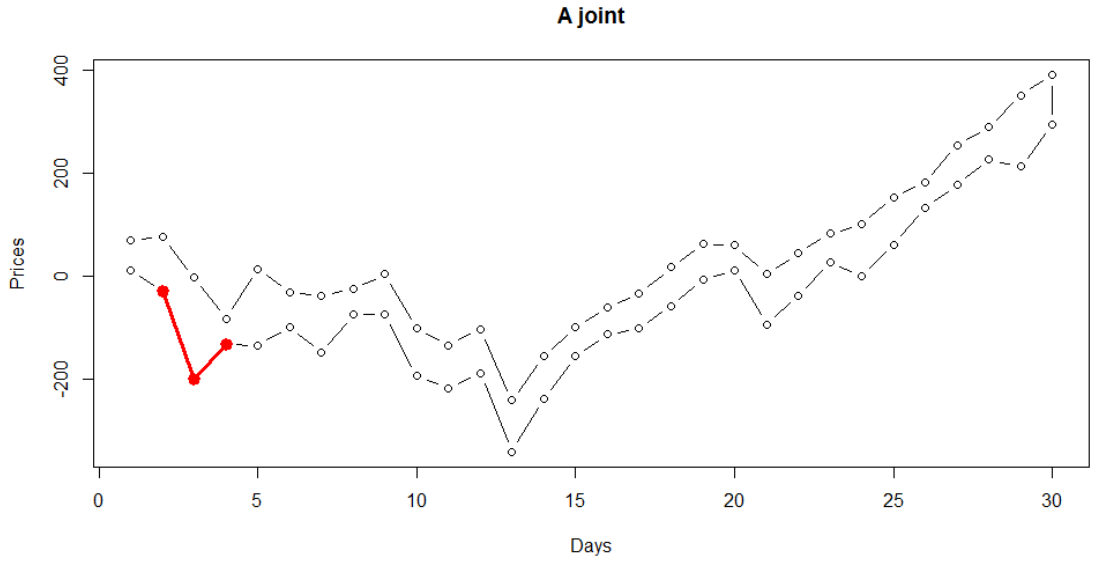

Figure 5: A joint

A joint, in turn, can be encoded or, in other words, described exactly in the similarity-invariant way by the segment ratio $\gamma_i = \frac{l_{seg_{i+1}}}{l_{seg_i}}$, where l – is the Euclidian distance, and the inner anti-clockwise angle θ_i in a range [0, 2π] between the adjacent pair of segments seg_i and seg_{i+1}:

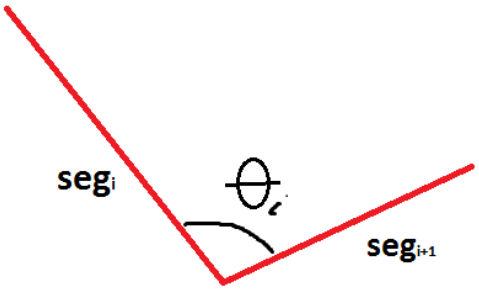

Figure 6: Joint's measures

Hence, this way we can describe the whole chain by storing these two measures for each joint of the chain. Finally, we obtain the descriptor for the chain $\Psi = \left\{\gamma_i = \frac{l_{seg_{i+1}}}{l_{seg_i}},\ \theta_i |\ i \in (1, 2, 3, \dots, 18)\right\}$. In general form it is:

$$\Psi_j = \left\{\gamma_i = \frac{l_{seg_{i+1}}}{l_{seg_i}}, \qquad \theta_i |\ i \in (1, \dots, N-2)\right\} \tag{2}$$

The second element of the model is to calculate the distance between two chains which is equivalent to determining how similar two original objects from our dataset. However, the distance algorithm and distance functions are two different things in our model. We adopted a general ideas of Mittal's article but changed the functions and the distance algorithm itself. At this point we want to introduce the distance function which compares two joints (x^{th} and y^{th} joints of the two descriptors for chains C_1 and C_2) in terms of similarity of their $\gamma's$ and angles:

$$D_{jnt}(x, y) = D_{lr}(x, y) + D_{ang}(x, y) \tag{3}$$

$$D_{lr}(x, y) = \lambda_{lr}(\Omega(\gamma_x{}^{C_1}, \gamma_y{}^{C_2}) - 1), \text{ where } \Omega(a, b) = \max(\frac{a}{b}, \frac{b}{a}) \tag{4}$$

$$D_{ang}(x, y) = \lambda_{ang}|\theta_x{}^{C_1} - \theta_y{}^{C_2}| \tag{5}$$

The expression (4) determines the distance between two joints in terms of the relative length measure, where $\Omega(a,b) \in [1,\infty)$ measures how similar two gammas, and λ_{lr} measure the weight of relative lengths similarity in the total distance. In turn, the expression (5) represents the distance between two joints in terms of the angle measure, where λ_{ang} measures the weight of relative angle similarity in the total distance. The expression (3) gives a total distance between two joints summing up (4) and (5). Because these functions use such measures as relative length ratios and angles, they are also invariant to translation, rotation and change of scale meaning they are as well similarity-invariant. Hence, given the possibility to calculate distance between any pair of joints, it is possible to calculate the distance between two chains by cumulatively summing (3) for each pair of joints. However, as it was mentioned in previous sections, we have a need to calculate the distances between objects of different length. For this case we have a special measure called sharpness which simply increases the distance between objects of different lengths by the following rule:

$$D(x) = \lambda_{sh}|\pi - \theta_x{}^{C_i}|) \tag{6}$$

The expression (6) returns a value based on the fact that if the angle of the joint is significantly different from π, then the object's shape is significantly changed and therefore, became very different from past realizing figure. Graphically:

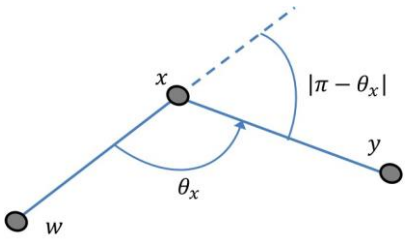

Figure 7: A sharpness measure. Source: (Parui, Mittal, 2015).

Hence, if we compare two objects of different lengths, the distance between them will be increased by the sharpness measure of the last joint of the longer object.

After the main considerations on the distance measures, we are ready to introduce the distance algorithm which uses Dynamic Programming-based partial matching method which is

based on Mittal's work of 2015 but changed for our needs. The interesting thing in this algorithm is that it calculates next value from previous ones or, in other words, each time it produces marginal summation. This feature makes this algorithm a dynamic programming one. Hence, this algorithm takes descriptors for two chains as inputs and produces the distance between chains as an output, it works as follows:

Step 1: It produces $(x + 1)\ by\ (y + 1)$ matrix of nulls (zeros), where x – the length of the first descriptor Ψ_1 , y – the length of the second descriptor Ψ_2.

Step 2: It checks three conditions: 1) $x = y$, 2) $x < y$, 3) and $x > y$.

Step 3: For the first condition, it simply calculates the distance between each two joints of Ψ_1 and Ψ_2 using (3) and marginally fits the diagonal of the matrix summing the previous diagonal element to the next one. For the second condition, it makes the same calculation as before and additionally sums the last diagonal element to the sharpness measure of the last additional joint of Ψ_2. For the third condition, the algorithm works vice versa.

Step 4: It returns the final distance, which lies in the last element of the matrix, between two chains.

However, we noticed that the distance algorithm can produce incorrect distances which are too large for the visually similar objects of different lengths. We found out that this happened due to the starting point in the chain of joints – the algorithm always looks at the first elements of each descriptor and compare them step-by-step summing up with the sharpness measure for the excess joints. Logically, this is not the right way because the objects can be similar not only at their beginning but somewhere in the middle or at the end. Therefore, we code an additional function which uses the same distance algorithm, but tries all possible portions of the descriptor when comparing joints of the objects of different lengths and returns the minimum distance obtained. That update ensures that any similar parts of two objects will be identified and the objects will be considered as similar as a result.

Summing up abovementioned information we introduced a novel method to represent times series as geometric objects which allows to apply similarity-invariant distance functions and Dynamic Programming-based partial matching algorithm to identify similar time series. The very important point to consider with respect to the abovementioned content is that out distance function and algorithm make possible to capture visually similar time series as real person does it. Any technical analyst uses his mind in the same way as our functions work when identifying shape similarity in objects (time series). Real traders simply look for any similarities

in any part of time series comparing each portion of the shape to each other portion of the shape marginally accounting for any differences. Our algorithm does the same. In this sense we produced a machine learning tool for revealing similar objects because we teach the computer to behave like a person. However, most TA practitioners rely on their subjective view or empirical methods in determining whether the patterns they see are significant or it is simply a disturbing noise. Hence, some readers may ask the question why it is important to teach a computer to see the similar time series, if traders already can simply do it, and moreover, this skill works very poor when TA practitioners try to extract some significant patterns. The answer to this question is very pleasant – when computer can identify similarities, it can store them and cluster in a mathematically proper manner such that we obtain a scientific rigorous prove of the patterns existence or their absence. Hence, let us proceed to the final part of the model – clustering method.

The foundation of our clustering algorithm is a popular k-means technique which learns to classify objects and with each iteration the algorithm makes it better and better. The only parameter which is user defined is the number of clusters. The classic version of k-means algorithm works as follows: let we have $X = \{x_1, \dots, x_n\}$ which is the set of objects which are generally vectors for multi-dimensional space or points for two dimensions; then k-mean algorithm minimizes the objective function:

$$F = \sum_{i=1}^{k} \sum_{x \in K_i} \left|\left|x - \mu_i\right|\right|^2 \tag{7}$$

In order to cluster n objects from X into k clusters $K_1, \dots, K_k$, where μ_i is the centroid of cluster K_i. The algorithm does this procedure iteratively by reassigning centroids by averaging the elements in obtained clusters until it converges and stops clusters' modification. We do not use the k-means algorithm for our task because it uses the Euclidian distance and recalculates centroids by averaging all objects in a cluster. When the goal is to cluster shapes this method cannot be used because it is not correct to create some average object as a centroid of a cluster because this will lead to bad results of mixing completely different objects. Moreover, in our environment of chains as representatives of time series we cannot use the Euclidian distance as

a criterion to define objects to clusters. Therefore, we use the slightly changed version of another popular clustering technique called k-medoids. The difference of these algorithms is that in k-medoids the center of a cluster is not created by averaging, but chosen from existing objects within the cluster. Furthermore, this algorithm works with arbitrary distance metrics between objects. Our upgraded k-medoids algorithm works generally the same as the original one except for the method of centers recalculation. In our clustering tool, an object becomes a new center, if it was on average closer to the previous center than all other objects. As inputs the algorithm takes the dataset vectors like $v_1 = (60.09, 58.23 \dots 57.85, 56.91 \dots)$ which was depicted on the Figure 1, and as output it produces a classification number for each object of a dataset which defines to which cluster this object belongs . The detailed procedure of the algorithm is as follows:

Step 1: It takes the objects from the dataset and randomly assigns centers to k of them, where k is the user-defined amount of clusters.

Step 2: It calculates distance, using our distance functions and distance algorithm, for each object to each cluster center and divide the dataset into clusters. Is also uses a linear approximation with $R^2 = 0.95$ for every considered object before distance calculation.

Step 3: It reassigns cluster centers by calculating an average distance of objects within a cluster to a cluster center and picking for a new center such an object that is on average closer to a cluster center that others. Formally, it picks such an object that has a minimum distance to a cluster center after the mean distance is subtracted.

Step 4: It repeats again from the step 1.

As it can be clearly seen that our algorithm has no natural convergence and even a synthetic convergence criterion unlike a standard k-medoids or k-means. That is because we work with very unusual objects as chains which represent time series and this environment has a non-standard distance measure. Although our distance measure definitely satisfies non-negativity condition or separation axiom $D_{jnt}(x, y) \geq 0$, identity of indiscernibles $D_{jnt}(x, y) = 0 \Leftrightarrow x = y$, symmetry $D_{jnt}(x, y) = D_{jnt}(y, x)$, subadditivity or triangle inequality $D_{jnt}(x, z) \leq D_{jnt}(x, y) + D_{jnt}(y, z)$ can be further inspected in more details. There is a vast space for further investigation of this properties which can qualify as a serious standalone work. Our space environment is rather difficult, that is why the violation in triangle inequality might happen sometimes. This way, among objects A, B, and C, A can be very similar to B, B is very similar to C, while A and C may be completely different. This is a potential point for further

investigation of our distance measures or other environments which can safely adopt our modern functions without breaking a law. Hence, that is why our clustering method has no convergence and therefore, it demands to be bounded by the number of iterations. We propose to limit the algorithm by the number of iterations in the following way: ten iterations form step 1 to step 3 and five iterations of those ten iterations. This is done for the purpose to randomly reassign new centers without repetition for each of five global iterations in order to capture the best result. Hence, we apply a standard measure of quality of the clustering result which is the sum of all objects distances within a cluster to their cluster center for all clusters, analytically: $\sum_{i=1}^{k} \sum_{x \in K_i} ||x - \mu_i||$.

Summing up, the framework of our model is the upgraded k-medoids tool and the core of our model is our novel distance functions and algorithm. Together they search for patterns in financial time series data identifying similar objects and storing them in order to cluster into groups. The model does this operation of pattern retrieval pretty much the same as it does an expert of TA. Moreover, the clustering algorithm learns from iteration to iteration how to better group the objects in order to obtain significant patterns. Overall, our model should produce results better than other state-of-the-art methods in financial time series pattern retrieval like automated algorithms in the work (Lo, Mamaysky, Wang, 2000). In order to test the significance of the patterns found by our algorithm we consider random noise dataset generated by Brownian motion. We expect to see that on noisy dataset our model will find mainly small or unity clusters containing only its centers which proves patterns retrieval power of our model and patterns existence in financial time series.

Chapter 4. Brownian Motion

In order to test whether our model is capable to extract significant patterns, we test it on a purely random dataset generated by Brownian motion. Formally, Brownian motion is a stochastic model where each new point of data is generated randomly from a normal distribution with null mean and variance of $\sigma^2 \Delta t$ which means that it linearly increases as time passes. In order to realise such a model we code a function which generates random values from normal distribution $N(0,1)$ and them cumulatively sum them obtaining a vector of a particular length which represents Brownian motion properties. We also do not want to have negative values in our simulated dataset because prices are always greater than null, so we shift up the entire vector by adding to it some constant to make all numbers positive. We use such type of test dataset because the mathematical model for Brownian motion is often used for the purposes to simulate such noisy data as stock market fluctuations. Furthermore, the absence of patterns in such a simulation is pre-defined because generated dataset is purely random walk. That is exactly what we need to check whether our model is helpful in searching significant pattern or whether financial time series contain any patterns at all.

To make a comparable dataset based on Brownian motion we do not need to generate excess amount of data or even the same amount as our original dataset contains, because any additional time series generated from Brownian motion adds no valuable information due to its randomness. Hence, we simulated a thousand objects for test purposes. However, it is necessary to obtain high and low prices from simulated data in order to construct the same objects that suits our functions and that were originally investigated, and it is also necessary to repeat the same manipulations which we previously did with real financial time series. The method to obtain high and low prices is rather simple: we generate some amount of intraday data for high and low prices determination and repeat this process for number of days we need. High prices are obtained by picking maxima for each day from intraday simulation, while low prices are obtained vice versa. Eventually, we got five hundred observations (vectors of high and low prices) for further manipulations. Then, we proceed to normalization and data cleaning procedures which prepares our simulated dataset to the clustering algorithm. As a result, before the clustering model, we acquire the dataset containing objects generated from Brownian motion with different lengths of 10, 20 and 30 days in the same proportion as our original dataset have. We pick some graphical examples for visual consideration:

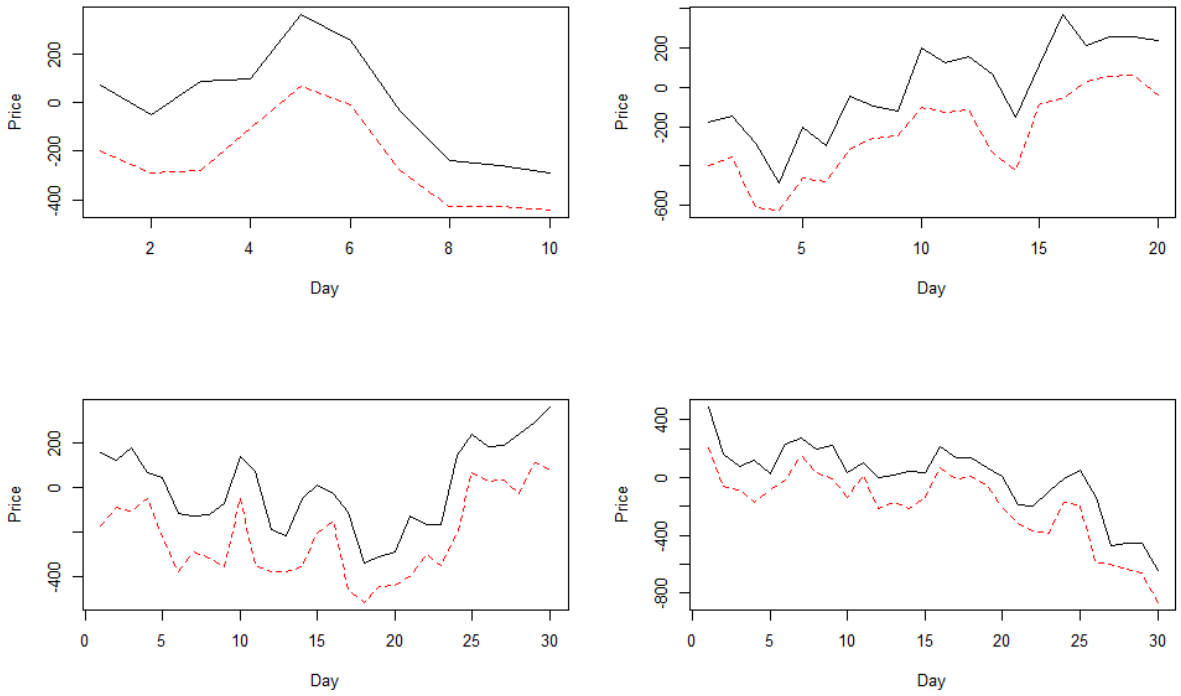

Figure 7: Brownian motion times series

Chapter 5. Results

Given that out algorithm seeks for any possible patterns, we are in a difficult situation regarding which statistical test to use. Logically, we understand that patterns are sophisticated figures (like on the Figure 1) which are met frequently enough across our real dataset, while if we have a situation where each cluster contain only one time series or too many different ones, that means our algorithm cannot extract patterns or real financial data simple do not contain them from the perspective of our model. Hence, we can qualitatively assess the significance of patterns retrieved by simply looking at clusters obtained, and we can quantitatively compare several characteristics presented further.

The first quantitative measure we want to introduce is a cluster size. In order to prove patterns existence in real data, cluster size should be on average moderate in comparison to the original dataset, while for our simulated dataset it should be on average too small or too large due to the fact that there naturally no patterns incorporated and each cluster should contain only outliers (objects that do not similar to any other ones). So, for simulated data there will be greater part of clusters with small amount of objects and a few clusters with larger amount of objects which are simply different and stuck together because of user-defined cluster limit. Hence, we should remove the outlier clusters, which contains more than ten percent of the dataset, and compare the mean amount of cluster size in rest of the data for real financial times series and the mean of clusters size in rest of the data in simulated time series, using Mann Whitney U Test due to the distribution of cluster sizes is not normal. According to our hypothesis the cluster size mean for real dataset should be higher than for simulated one. The second quantitative measure is based on the qualitative perception of patterns obtained: we can selectively pick patterns that are obtained in real dataset and compare their amount to the patterns obtained in simulated dataset which gives us the relative measure of patterns retrieved.

In turn, we also can qualitatively asses the patterns presence looking at obtained clusters and determining whether the time series there are really similar to each other or even more, similar to some popular patterns like “head-and-shoulders”.

But for the beginning, let us look at the first results of our model we obtain when test it on a small sample of objects. The setup of the model was three clusters and ten random objects of the dataset described in previous chapter. We want to remind that our algorithm retrieves patterns invariantly to rotation, translation, and scale, therefore, in order to visually compare

two objects, it is necessary to account for this factor. Here the graphical representation of obtained clusters:

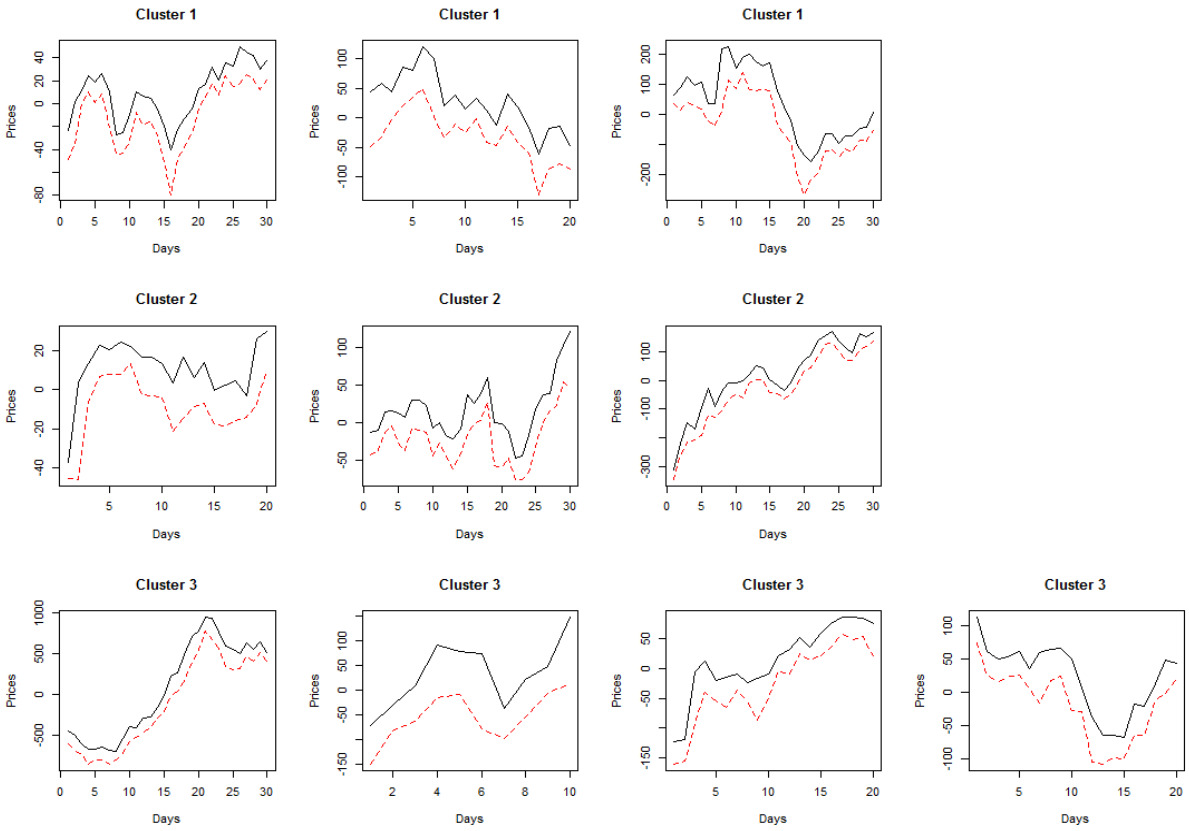

Figure 8: Example of clustering 10 objects into 3 clusters

Hence, as we can see, the algorithm works rather well for the cluster one: all pictures are similar indeed taking into account that in order to see the similarities in rotation, translation and scale-invariant way, it is necessary to reverse the first picture from right to left. However, for second and thirds clusters pictures seem to be randomly defined. It is seen from the graph that all pictures in last two clusters are different and perhaps, all of them should be defined in each separate cluster. For the sake of speed, we do not run our algorithm with eight clusters for the same ten objects in order to prove our hypothesis, because we can prove it considering only two clusters instead. For the case of two clusters the right classification would be to put all different objects in one cluster and three first objects in the other cluster. This method is possible because the number of clusters is user-defined and we always get some large noisy clusters of objects that are completely different from each other. Here we present the graph:

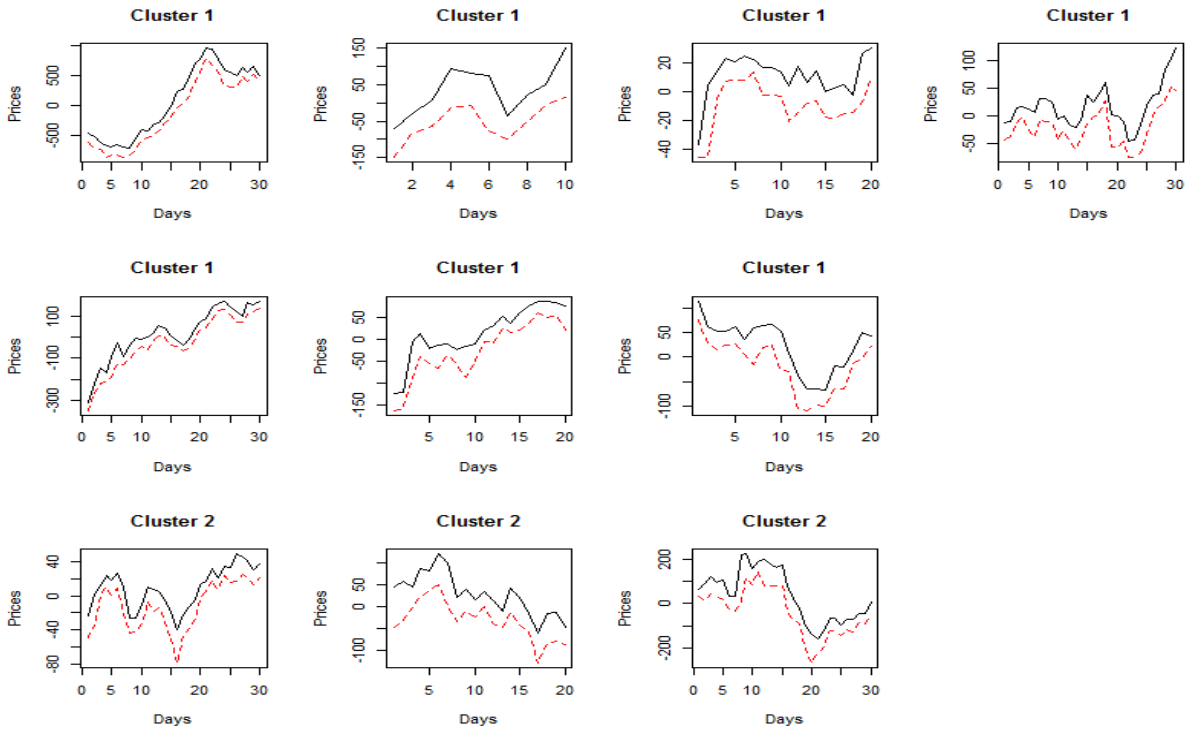

Figure 9: Example of clustering 10 objects into 2 clusters

As we expect, we obtain the right classification which proves that our algorithm can correctly determine the similarity between objects, but it does not prove the presence of patterns in real financial time series at all due to we consider only small sample of objects.

Now, let us consider the dataset we use for all our tests and work which, unfortunately, is not the dataset described in chapter two, because the algorithm works too long when considering 197338 objects. The model works forty seconds and a half for the case of ten objects and two clusters which implies that for 197338 objects and two clusters it will take around three months. However, for such great amount of objects it would not be enough to consider only two clusters and in this case we would be compelled to add more clusters which would make processing time more by several times (the processing time is a multiple of the number of clusters). Therefore, our work dataset contains of a random sample of size 1000 from the original dataset described in chapter two and we consider ten clusters (empirically, good clustering produces the amount of clusters which is smaller than the considered dataset by two orders of magnitude). Let us inspect clusters we obtained. Firstly, it is worth to mention the amount of objects in each cluster we acquired, so there three clusters of large amount (180-300 objects), three clusters of moderate amount (60-90 objects), and four small clusters (9-20 objects). Let us

try qualitatively asses the patterns retrieved. For this purpose, we pick the best cluster obtained and look into its figures. The cluster contains twenty objects from which we represent first fourteen:

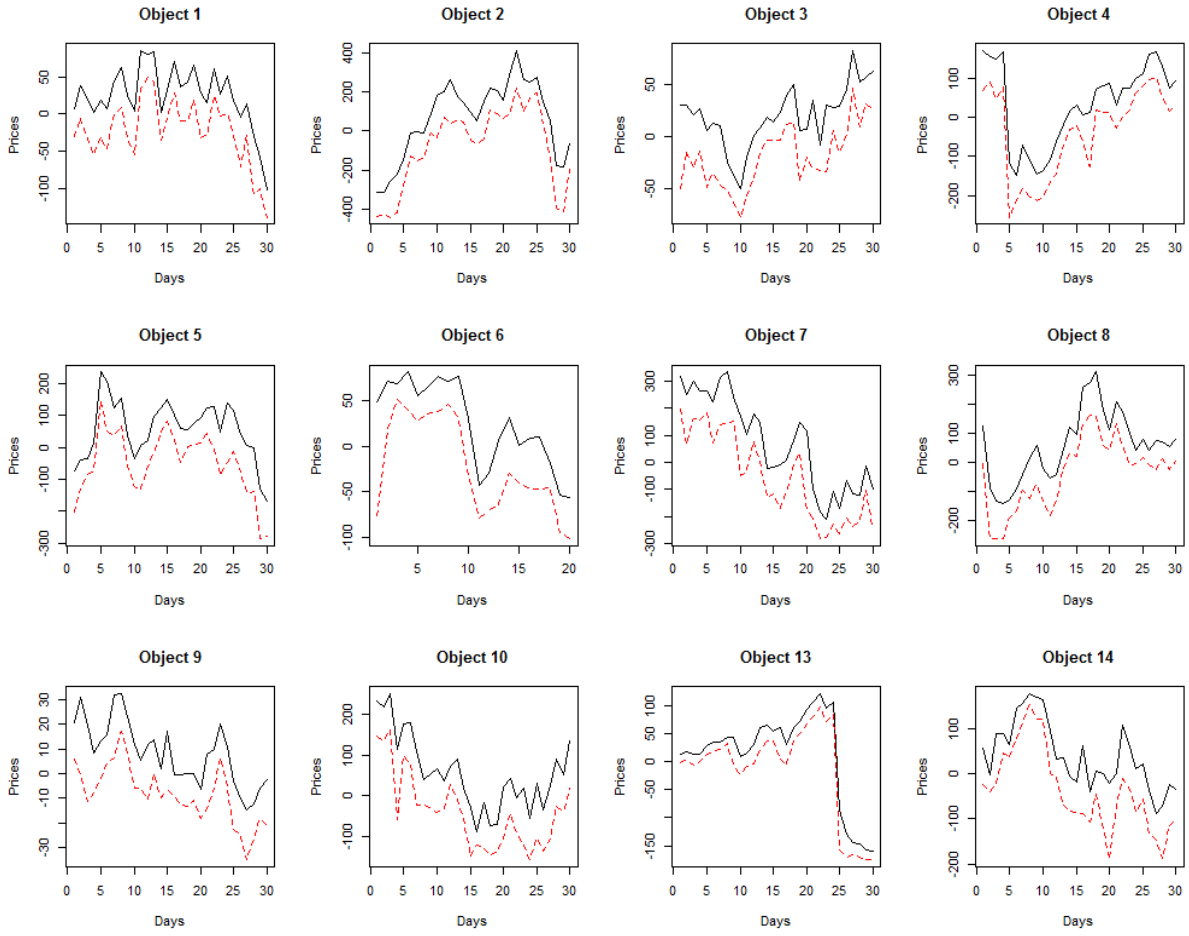

Figure 10: Cluster 1 “Double Top”-similar

For the first look, it may seem that objects are not similar. This is partially true due to that we have a clustering with user-defined number of clusters which is a significant drawback of our model and therefore, the algorithm sometimes defines different excess objects to the cluster of similar objects. Nevertheless, detailed look at the Figure 10 (do not forget about similarity-invariant feature) allows to see that, on average, objects, for a small exception of abovementioned outliers, are similar to the “double top” pattern or almost “double top” where two tops are within a small range from each other. This pattern is classically defined as:

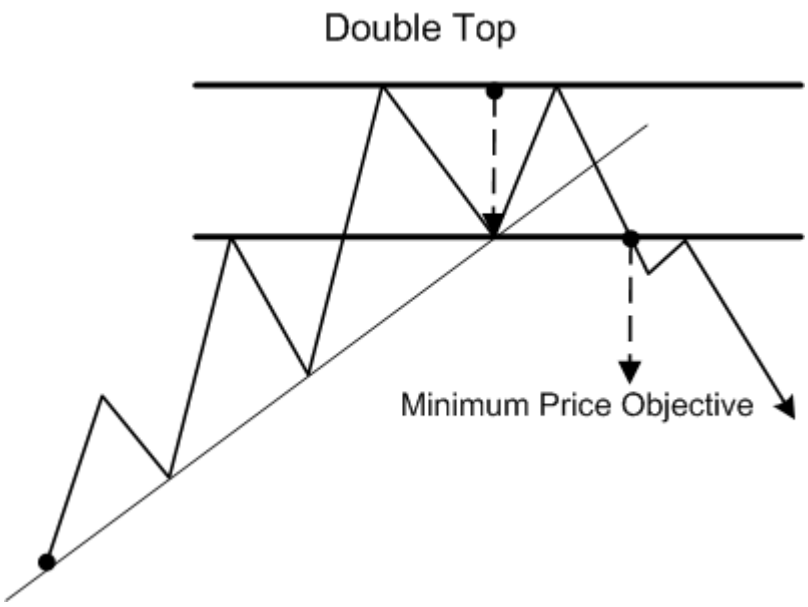

Figure 11: TA classical double top. Source: 2000freepatterns.ludug.com.

We also cannot forget about that our model searches and retrieves all possible patterns comparing the similarity of all objects in the dataset. Hence, it also can find previously non-existent patterns like in the following cluster (this cluster contain ninety-two objects in total):

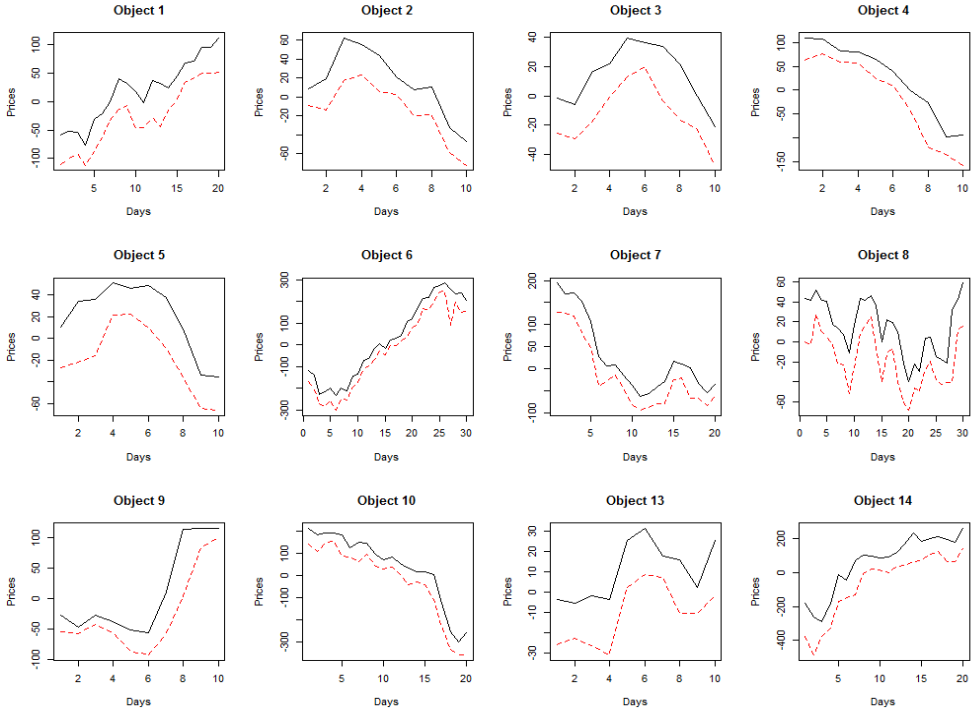

Figure 12: Cluster 2 of mountain-like figures

This cluster, for a small exception of excess objects, consists of figures which are similar to mountains with a single top or the parts of mountain which lead to this top. The other clusters we obtain consists of objects that are rather different from each other and we cannot devote them to patterns. On the greater part they are represented by the following bunch of time series:

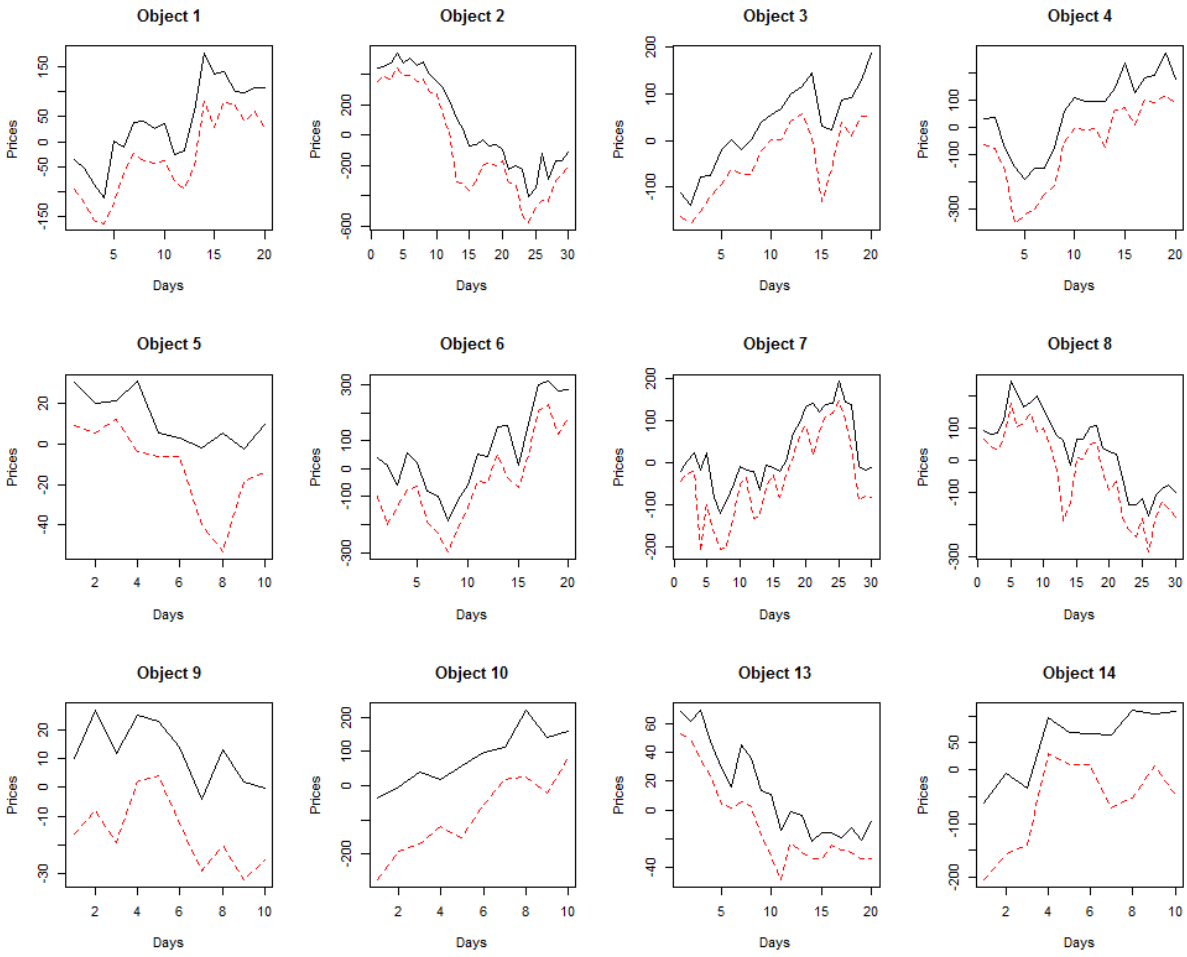

Figure 13: Cluster of different objects

Hence, as a result we obtain two patterns ("Double Top"-like and a mountain-like) which is rather good for the case with a thousand of objects.

With respect to the simulated dataset, we obtained exactly what we expected, let us introduce the cluster sizes distribution: five large clusters (100-250 objects), three moderate clusters (70-100 objects), and two small clusters of 9 and 21 objects. The graphical representation of the cluster containing 9 objects approves its randomness:

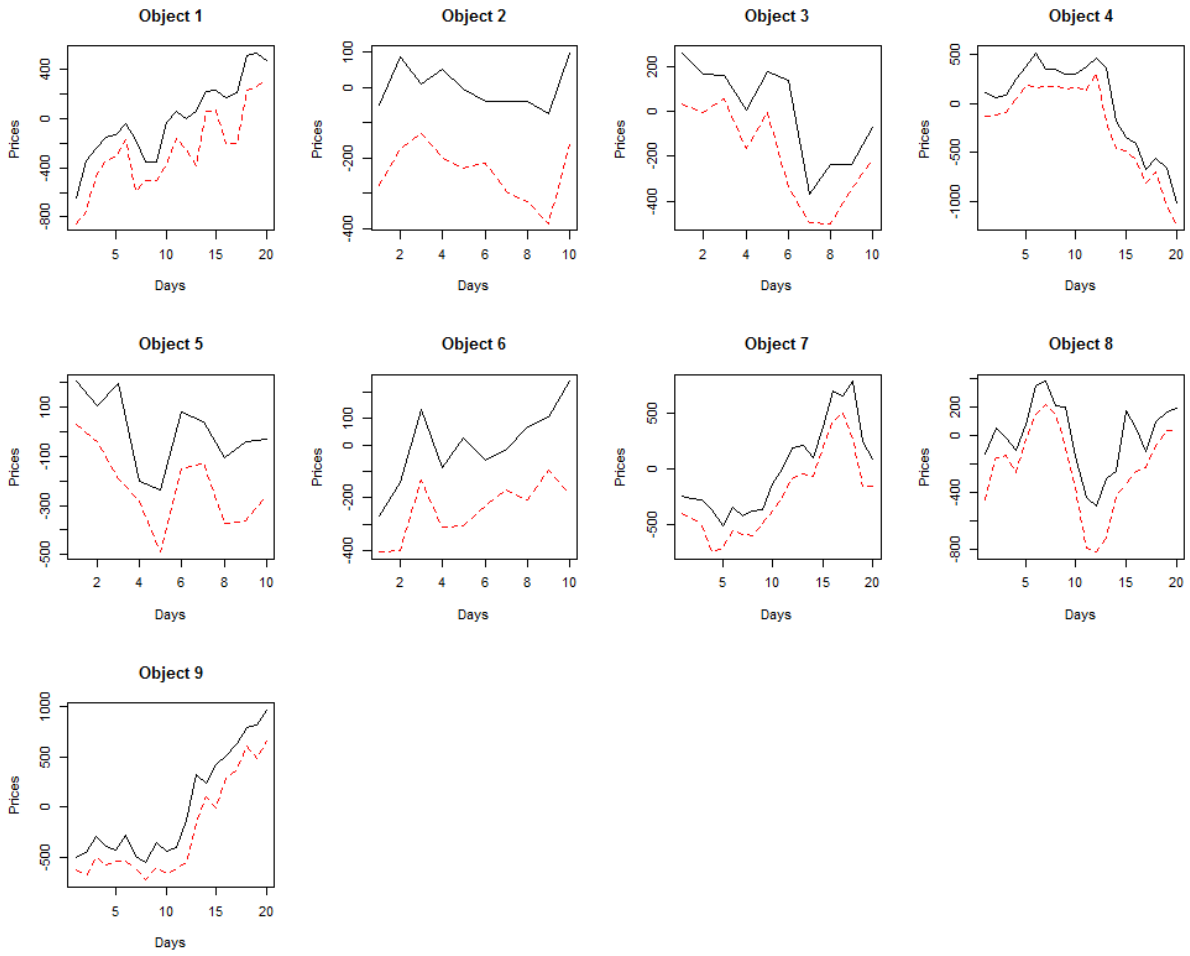

Figure 14: Cluster of simulated objects

Next we can produce the non-parametric Mann Whitney U Test which can help to check the equality of means in two independent samples. We use this test because the samples we have are not normally distributed and the samples' sizes are rather small (we have ten clusters). As it was mentioned at the beginning of this chapter, we need to prepare our samples by making the following procedure: remove outlier clusters (containing more than ten percent of the original dataset) from our consideration and take only the rest one. According to our supposition a mean cluster size for real data should be higher than for simulated one because synthetic data contains only different objects (the probability to obtain similar objects from Brownian motion is 0.01%) and greater part of them should be concentrated in outlier clusters leaving all other clusters with small amount of objects. So this test helps us to check whether financial time series contain any pattern or it is equivalent to random walk represented by the dataset simulated by Brownian motion. The hypothesis for the Mann Whitney U Test is as follows:

$$H_0: two\ population\ distributions\ are\ equal$$

$$H_1: two\ population\ distributions\ are\ not\ equal$$

This is equivalent for standard hypotheses of t-test: H_0: $\mu_1 = \mu_2$ and H_1: $\mu_1 > \mu_2$. The statistic for the Mann Whitney U Test is called U and it represents the smaller of U_1 and U_2 defined as follows:

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$

$$U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2, where\ n_1, n_2 \quad (8)$$
$$- the\ 1^{st}\ and\ 2^{nd}\ sample\ sizes, R_1, R_2$$
$$- sum\ of\ the\ ranks\ for\ 1^{st}\ and\ 2^{nd}\ groups$$

Our sample sizes cleaned from outlier clusters are seven for real dataset and five for simulated one. Therefore, the one-tailed U critical value for 5% significance level is 6. So, we obtained U equal to 3 and rejected H_0 which proves our initial supposition and the presence of patterns in financial time series.

Eventually, it is important to state directions for further work. Firstly, as it was mentioned at the end of chapter three, there is a need in more detailed exploration of distance functions represented in (Parui, Mittal, 2015) and our updated distance functions and distance algorithm. The further investigation should mainly concern the issue of analytical prof for the last distance property called the triangle inequality or subadditivity for the case of our distance functions. This could help better understand the environment of shape distances and work out the better clustering algorithm which works a lot faster and define clusters significantly better.

The other significant space for further research is clustering algorithm which we use. Essentially, it is not logical to use clustering algorithm with user-defined amount of clusters when the task is to classify unlabeled data. Because of this drawback, our final clusters contain noisy excess objects which disturb the attention when looking at other normal cluster objects. But the main reason we use our k-medoids algorithm is that it saves a significant amount of time in comparison to brute-force-like algorithms and produces comparatively good results when we

define the amount of clusters that is closer to the real one. The good extension of this work would be to try DBSCAN clustering algorithm which is the most cited in scientific literature till then.

It is also important to extent our work by developing the trading algorithm which would use the retrieved patterns for price forecasting. Such algorithm would help in research of the topic of how possible to earn excess returns using our pattern recognition algorithm.

The last obvious direction of work is to research larger datasets containing at least the amount of objects described in chapter two. Moreover, it would be the good idea to add the stock label and the time label to the dataset in order to determine to which company and period of time patterns belong.

Conclusion

In this paper we presented the automated algorithm for searching significant patterns in financial time series data using machine learning techniques. We apply our algorithm on the dataset of thousand random objects generated from the original dataset described in chapter two, and test the model on the dataset simulated by Brownian motion. The hypothesis of whether financial data contain any statistically significant patterns is confirmed with respect to instruments of our research.

We believe that our model could be significantly extended by considering the suggestions at the end of chapter five, and it would significantly increase the efficiency of patterns retrieval from financial time series data.

List of References

1. Andrew W. Lo, Jasmina Hasan Hadzic. “The heretics of finance: Conversations with leading practitioners of technical analysis.” 2010.
2. Bechu, Bertrand. “Laanalyse technique: pratiques et methods.” 1999.
3. Caginalp, Laurent. “The predictive power of price patterns.” Applied Mathematical Finance, 1998.
4. Chang, Carol L. Osler. “Methodical Madness: Technical Analysis and the Irrationality of Exchange-rate Forecasts.” The Economic Journal1, 1999.
5. Cheung, Chinn. “Currency traders and exchange rate dynamics: a survey of the US market.” Journal of international money and finance, 2001.
6. Efron. “The jackknife, the bootstrap and other resampling plans.” 1982.
7. Fama, French. “The capital asset pricing model: Theory and evidence.” Journal of Economic Perspectives, 2004.
8. Malkiel, Fama. “Efficient capital markets: A review of theory and empirical work.” The journal of Finance, 1970.
9. Jensen. “Some anomalous evidence regarding market efficiency.” Journal of financial economics, 1978.
10. Lo, Mamaysky, Wang. “Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation.” The journal of finance, 2000.
11. Lui, Mole. “The use of fundamental and technical analyses by foreign exchange dealers: Hong Kong evidence.” Journal of International Money and Finance, 1998.
12. Mills, Markellos. “The econometric modelling of financial time series.” 2008.
13. Parui, Mittal. “Sketch-based Image Retrieval from Millions of Images under Rotation, Translation and Scale Variations.” 2015.
14. Pring. “Technical analysis explained: The successful investor's guide to spotting investment trends and turning points.” 2002.
15. Schabacker. “Stock market theory and practice.” 1930.
16. Taylor, Allen. “The use of technical analysis in the foreign exchange market.” Journal of international Money and Finance, 1992.
17. Tsay. “Analysis of financial time series.” 2005.

18. Ulanova, Begum, Keogh. “Scalable Clustering of Time Series with U-Shapelets.” Conference on Data Mining (SDM 2015), 2015.
19. Ye, Keogh. “Time series shapelets: a new primitive for data mining.” Proceedings of the 15th ACM SIGKDD international, 2009.
20. Zakaria, Mueen, Keogh. “Clustering time series using unsupervised-shapelets.” Data Mining (ICDM), 2012.

```
 ## Download needed packages
install.packages("Quandl")
install.packages("quantmod")
install.packages("XML")
library(XML)
library(Quandl)
library(quantmod)
            ## Define all needed functions
#The Parkinson volatility estimator
parkinson.value <- function(h,l){
      ratio <- log(h/l)
      vol <- sqrt(sum(ratio^2)/4/log(2))/sqrt(length(l))
  return(vol)
}
#Obtain subsequences from long time series
getsubsequences <- function(price.vec,subseq.length){
  subsequence.mat <- matrix(0,subseq.length,length(price.vec)-
subseq.length+1)
  sliding.window <- 1:subseq.length
  for (i in 1:(length(price.vec)-subseq.length+1)){
    subsequence.mat[,i] <- price.vec[sliding.window]
    sliding.window <- sliding.window + 1
  }
  return(subsequence.mat)
}
#Transform time series into descriptors
edge.feature.chain.new <- function(hl.line){
  l <- length(hl.line)
  n <- l/2
  ### 2d coordinates of edges ###
```

```
  edge.coordinates <-
rbind(cbind(1:n,hl.line[(1:n)+n]),cbind(n:1,hl.line[n:1]))
  relative.length.vec <- rep(1,l-2)
  relative.angle.vec <- rep(1,l-2)
  for (i in 2:(l-1)){
    im <- i-1
    ip <- i+1
    x <- c(edge.coordinates[im,1],edge.coordinates[im,2])
    y <- c(edge.coordinates[i,1],edge.coordinates[i,2])
    z <- c(edge.coordinates[ip,1],edge.coordinates[ip,2])
    relative.length.vec[i-1] <- relative.distance.2d(x,y,z)
    relative.angle.vec[i-1] <- relative.angle.2d.v2(x,y,z)
  }
  return(cbind(relative.length.vec,relative.angle.vec))
}
edge.feature.chain.any <- function(points){
  l <- dim(points)[1]
  n <- l
  ### 2d coordinates of edges ###
  edge.coordinates <- points
  relative.length.vec <- rep(1,l-2)
  relative.angle.vec <- rep(1,l-2)
  for (i in 2:(l-1)){
    im <- i-1
    ip <- i+1
    x <- c(edge.coordinates[im,1],edge.coordinates[im,2])
    y <- c(edge.coordinates[i,1],edge.coordinates[i,2])
    z <- c(edge.coordinates[ip,1],edge.coordinates[ip,2])
    relative.length.vec[i-1] <- relative.distance.2d(x,y,z)
    relative.angle.vec[i-1] <- relative.angle.2d.v2(x,y,z)
  }
  return(cbind(relative.length.vec,relative.angle.vec))
}
```

```r
#Associatited elements of above functions
eucledean.distance <- function(x,y){
  return(sqrt(sum((x-y)^2)))
}
relative.angle.2d <- function(x,y,z){
  phi1 <- 180*atan((y[2]-x[2])/(y[1]-x[1]))/pi
  phi2 <- 180*atan((z[2]-y[2])/(z[1]-y[1]))/pi
  return(-phi1+phi2+180)
}
relative.angle.2d.v2 <- function(x,y,z){
  v1 <- y-x
  v2 <- z-y
  cross.product <- sign(v1[1]*v2[2]-v1[2]*v2[1])
  if (cross.product==-1){
    coef <- 180
  } else{
    coef <- 0
  }
  angle <- sum(v2*v1)/sqrt(sum(v1^2))/sqrt(sum(v2^2))
  angle <- min(1,angle)
  angle <- max(-1,angle)
  theta <- acos(angle)*180/pi
  if (cross.product==1){
    theta <- 180-theta
  }
  result <- theta+coef
  if   ( (result==0) || (result==360) )  {
    result <- 180
  }
  return(result)
}
relative.distance.2d <- function(x,y,z){
  v1 <- y-x
```

```
  v2 <- z-y
  return(sqrt(sum(v2^2))/sqrt(sum(v1^2)))
}
#Distance functions
cost_joint <-
function(joint1,joint2,lambda_len=1,lambda_angle=5){
  cost <- 0
  cost_length <- lambda_len * (
max(joint1[1]/joint2[1],joint2[1]/joint1[1])-1 )
  cost_angle <- lambda_angle*abs(joint2[2]-
joint1[2])*(2*pi)/360
  cost <- cost_length + cost_angle
  return(cost)
}
sharpness_joint <- function(joint,lambda=10){
  return( lambda * abs(pi-joint[2]*(2*pi)/360) )
}
#Distance algorithm
matchDistance.new <- function(x,y){
  x.len <- dim(x)[1]
  y.len <- dim(y)[1]
  matchDistance.mat <- matrix(0,x.len+1,y.len+1)
  if (x.len == y.len){
    for (i in 2:(x.len+1)){
      matchDistance.mat[i,i] <- matchDistance.mat[i-1,i-1] +
cost_joint(as.vector(x[i-1,]),as.vector(y[i-1,]))
    }
  }
  if (x.len < y.len){
    for (i in 2:(x.len+1)){
      matchDistance.mat[i,i] <- matchDistance.mat[i-1,i-1] +
cost_joint(as.vector(x[i-1,]),as.vector(y[i-1,]))
    }
```

```
    for (j in (x.len+2):(y.len+1)){
      matchDistance.mat[x.len+1,j] <-
matchDistance.mat[x.len+1,j-1] + sharpness_joint(as.vector(y[j-
1,]))
    }
  }
  if (x.len > y.len){
    for (i in 2:(y.len+1)){
      matchDistance.mat[i,i] <- matchDistance.mat[i-1,i-1] +
cost_joint(as.vector(x[i-1,]),as.vector(y[i-1,]))
    }
    for (j in (y.len+2):(x.len+1)){
      matchDistance.mat[j,y.len+1] <- matchDistance.mat[j-
1,y.len+1] + sharpness_joint(as.vector(x[j-1,]))
    }
  }
  return(matchDistance.mat[x.len+1,y.len+1])
}
matchDistance.invariant <- function(x,y){
  y.len <- dim(y)[1]
  x.len <- dim(x)[1]
  if (y.len == 1 | x.len ==1){
      return(matchDistance.new(x,y))
  }
  benchmark <- c(1:y.len,1:y.len)
  distance.vec <- rep(0,y.len)
  for (i in 1:y.len){
    shift <- benchmark[1:y.len + i-1]
    y.shift <- y[shift,]
    distance.vec[i] <- matchDistance.new(x,y.shift)
  }
  return(min(distance.vec))
}
```

```
#Compiled distance function
my_dist <- function(x, y, rkvad = 0.95)  {
    d1 <- rbind(cbind(1:(length(x)/2),x[ 1:(length(x)/2) +
(length(x)/2) ]),cbind( (length(x)/2):1,x[ (length(x)/2):1] ))
    d2 <- rbind(cbind(1:(length(y)/2),y[ 1:(length(y)/2) +
(length(y)/2) ]),cbind( (length(y)/2):1,y[ (length(y)/2):1] ))
    sub_d1 <- unname(shape.linear.approx(d1,rkvad))
    sub_d2 <- unname(shape.linear.approx(d2,rkvad))
    distance <- 0
    if ( (dim(sub_d1)[1] > 2) && (dim(sub_d2)[1] > 2)  ){
        distance <-
abs(matchDistance.invariant(edge.feature.chain.any(sub_d1),
edge.feature.chain.any(sub_d2)))
    } else{
        distance <-
abs(matchDistance.invariant(edge.feature.chain.new(x),
edge.feature.chain.new(y)))
    }
    return(distance)
#Obtain Dow Jones stock names
DJIcomponents <- function() {
    if (!("package:XML" %in% search() ||
require("XML",quietly=TRUE))) {
        stop("Please install the XML package before using this
function.")
    }
    djicomp <-
readHTMLTable('http://finance.yahoo.com/q/cp?s=^DJI+Components'
)
    dow <<-data.frame(djicomp[[grep("Symbol", djicomp)]])
}
#Linear approximation function
shape.linear.approx <- function(points,r.sq.threshold){
```

```
196       lin.approx <- vector()
197       x.vec <- points[,1]
198       y.vec <- points[,2]
199       lin.approx <- c(x.vec[1],y.vec[1])
200       i=1
201       while (i <= (dim(points)[1]-3) ){
202           #cat(i, " ")
203           rsq <- rep(0,dim(points)[1])
204           for (k in (i+2):dim(points)[1]){
205               index <- seq(i,k,by=1)
206               y <- y.vec[index]-y.vec[i]
207               x <- x.vec[index]-x.vec[i]
208               beta <- sum(y*x)/sum(x*x)
209               rsq[k] <-  1-sum((y-beta*x)^2)/sum(y^2)
210           }
211           #cat("\n",rsq,"\n")
212           rsq[is.nan(rsq)] <- 1
213           if (max(rsq) > r.sq.threshold){
214               index.to <- tail(which(rsq == max(rsq)),1)
215               lin.approx <- rbind(lin.approx,
216   c(x.vec[index.to],y.vec[index.to]) )
217               i = index.to
218           } else{
219               lin.approx <- rbind(lin.approx,
220   c(x.vec[i+1],y.vec[i+1]))
221               i=i+1
222           }
223       }
224       return(lin.approx)
225   }
226               ##Obtain raw data
227   DJIcomponents()
228   dow_JI <- as.vector(dow$Symbol)
```

```
snp_rawdataset <- new.env()
getSymbols(dow_JI, env=snp_rawdataset)
            ##Create dataset of objects
dataset <- list()
dataset_global <- list()
for (i in 1:length(dow_JI)){
    h <-
as.vector(snp_rawdataset[[dow_JI[i]]][,2]*snp_rawdataset[[dow_J
I[i]]][,6]/snp_rawdataset[[dow_JI[i]]][,4])
    l <-
as.vector(snp_rawdataset[[dow_JI[i]]][,3]*snp_rawdataset[[dow_J
I[i]]][,6]/snp_rawdataset[[dow_JI[i]]][,4])
    c <- as.vector(snp_rawdataset[[dow_JI[i]]][,6])
    window.len <- c(10,20,30) ### to be changed to 5:100 if
time allows to make full scale test
    cat(i," ")
    for (j in 1:length(window.len)){
        output <- vector()
        h.sub <- getsubsequences(h,window.len[j])
        l.sub <- getsubsequences(l,window.len[j])
        c.sub <- getsubsequences(c,window.len[j])
        mean.vec <- apply(c.sub,2,mean)
        for (u in 1:dim(h.sub)[2]){
            vol <- parkinson.value(h.sub[,u],l.sub[,u])
            # apply z-score
            h.mod <- (h.sub[,u] - mean.vec[u]) / vol
            l.mod <- (l.sub[,u] - mean.vec[u]) / vol
            output <- rbind(output,c(h.mod,l.mod))
        }
        dataset[[j]] <- output
    }
    dataset_global[[i]] <- dataset
}
```

```
names(dataset_global) <- dow_JI
#Create dataset for k_means
dataset_kmeans <- list()
counter <- 1
for (j in 1:length(dataset_global)){
    cat('# GLOBAL', j, '\n')
    cat(' ', '\n')
    # Sys.sleep(1)
    for (l in 1:length(dataset) ){
        cat('# local', l, '\n')
        cat(' ', '\n')
        # Sys.sleep(0.3)
        for (i in 1:dim(dataset_global[[j]][[l]])[1]){
            # cat('# row', i, '\n')
            # cat(' ', '\n')
            dataset_kmeans[[counter]] <-
dataset_global[[j]][[l]][i,]
            counter <- counter + 1
        }
    }
}
#Cleaning dataset with linear approx
rkvad=0.95
dataset_kmeans_clean <- list()
avoided<-c()
for (i in 1:length(dataset_kmeans)) {
    cat('# iteration', i, '\n')
    cat(' ', '\n')
    x<- dataset_kmeans[[i]]
    len <- length(dataset_kmeans[[i]])/2
    d1 <- rbind(cbind(1:len,x[(1:len)
+len]),cbind(len:1,x[len:1]))
    sub_d1 <- unname(shape.linear.approx(d1,rkvad))
```

```
295       if (dim(sub_d1)[1] >= 7) {
296           dataset_kmeans_clean[[i]] <- x
297       }else{
298           avoided<-c(avoided,i)
299       }
300   }
301   nulls <- c()
302   for (i in 1:length(dataset_kmeans_clean)){
303       if (is.null(dataset_kmeans_clean[[i]])){
304           nulls <- c(nulls, i)
305       }
306   }
307   dataset_kmeans_clean <- dataset_kmeans_clean[-nulls]
308   length(dataset_kmeans) - length(dataset_kmeans_clean)
309                ##K_means algorithm
310   dataset_in_use <-
311   dataset_kmeans_clean[sample(length(dataset_kmeans_clean),1000)]
312   # to take random objects from out dataset, but restrict their
313   amount to 2500
314   k <- 10
315   indices <- list()
316   index.centers <- sample(length(dataset_in_use),k)
317   rsq <- 0.95
318   clusters_final <- list()
319   output <- matrix(0,10,length(dataset_in_use))
320   cost.mat <- matrix(0,10,5)
321   centers_all<- matrix(0,5,k)
322   ptm <- proc.time()
323   for(iter_glob in 1:5) {
324       cat(" ", '\n')
325       cat("GLOBAL #", iter_glob, '\n')
326       cat(" ", '\n')
```

```
    if (any(as.vector(apply(centers_all, 2, function(y) {y %in%
index.centers})))){ # to check that new random centers in
global iterations are defferent each time
        if (length((1:length(dataset_in_use))[-
unlist(as.list(centers_all))[unlist(as.list(centers_all)) !=
0]])< k){
            index.centers <- sample(length(dataset_in_use),k)
        }
        index.centers <- sample((1:length(dataset_in_use))[-
unlist(as.list(centers_all))[unlist(as.list(centers_all)) !=
0]],k)
    }
    centers_all[iter_glob, ] <- index.centers
    centers <- dataset_in_use[index.centers]
    for (iter_num in 1:10){
        cat("iteration #", iter_num, '\n')
        cat(index.centers,"\n")
        d.mat <- matrix(0,length(dataset_in_use),k)
        for (i in 1:length(dataset_in_use)) { # calculate
distance of each point to each center and separate them into
corresponding clusterts
            dst <- lapply(centers, function(y)
{my_dist(dataset_in_use[[i]], y, rsq)})
            d.mat[i,] <- unlist(dst)
            indices[[i]] <- which.min(dst)
        }
        cost <- sum(apply(d.mat,1,min))
        cost.mat[iter_num,iter_glob]<-cost
        cat( cost ,"\n")
        output[iter_num,] <- unlist(indices)
        for (i in 1:k){ # recalcualte centers in current
clusters
            initial.index <- which(indices==i)
```

```
            d.cl <- d.mat[indices==i,i]
            new_cent <- initial.index[which.min(abs(d.cl-
mean(d.mat[indices==i,i])))]
            centers[[i]] <- dataset_in_use[[new_cent]]
            index.centers[i] <- new_cent
        }
        if (cost.mat[iter_num,iter_glob] %in%
rev(cost.mat[,iter_glob][cost.mat[,iter_glob] != 0])[-1][1:3]
){
            break # to check if we stuck into the same two
changing centers - then cost
            #in the iteration is repeating with period 3 or
less,
            #if so we do not want to stay in the iteration and
go to the next global
        }
    }
    clusters_final[[iter_glob]] <- output
}
proc.time() - ptm
            ##Look at results
p<-which(round(cost.mat) ==
round(min(unlist(as.list(cost.mat))[unlist(as.list(cost.mat))
!= 0])), arr.ind = T)
cost.mat
needed_cluster <- p[1,]
final_indices<-
clusters_final[[as.numeric(needed_cluster[2])]][as.numeric(need
ed_cluster[1]),]
# look into length of clusters
for (i in 1:k){
    print(length(dataset_in_use[final_indices == i]) )
}
```

```
# Loon at specific cluster
cluster_we_want <- 1 # choose cluster you want in a range of
1:k
main_cluster <- dataset_in_use[final_indices ==
cluster_we_want]
dev.off()
par(mfrow=c(3,4)) # represent in a widnow to the right by
grpahs of 3 rows by 4 columns
for (e in 0:10){   # draw 12 graps of objects for 10
iterations, needs to be adjusted for case you want
    for (q in ( (1:12) + e*10 ) ) {
        x <- main_cluster[[1]]
        y <- main_cluster[[q]]
        l <- length(y)/2
        d <- my_dist(x, y, rsq)

matplot(cbind(y[1:l],y[(1:l)+l]),type="b",main=paste(q,round(d,
2)))

    }
    Sys.sleep(2)
}
```