Forecasting Foreign Exchange Rate Movements with k-Nearest-Neighbour, Ridge Regression and Feed-Forward Neural Networks

Milan Fičura

University of Economics, Prague

Faculty of Finance and Accounting

Department of Banking and Insurance

Winston Churchill Square 4, 130 67, Prague 3, Czech Republic

E-mail: milan.ficura@vse.cz

Abstract

Three different classes of data mining methods (*k*-Nearest Neighbour, Ridge Regression and Multilayer Perceptron Feed-Forward Neural Networks) are applied for the purpose of quantitative trading on 10 simulated time series, as well as real world time series of 10 currency exchange rates ranging from 1.11.1999 to 12.6.2015. Each method is tested in multiple variants. The *k*-NN algorithm is applied alternatively with the Euclidian, Manhattan, Mahalanobis and Maximum distance function. The Ridge Regression is applied as Linear and Quadratic, and the Feed-Forward Neural Network is applied with either 1, 2 or 3 hidden layers. In addition to that Principal Component Analysis (PCA) is eventually applied for the dimensionality reduction of the predictor set and the meta-parameters of the methods are optimized on the validation sample. In the simulation study a Stochastic-Volatility Jump-Diffusion model, extended alternatively with 10 different non-linear conditional mean patterns, is used, to simulate the asset price behaviour to which the tested methods are applied. The results show that no single method was able to profit on all of the non-linear patterns in the simulated time series, but instead different methods worked well for different patterns. Alternatively, past price movements and past returns were used as predictors. In the case when the past price movements were used, quadratic ridge regression achieved the most robust results, followed by some of the *k*-NN methods. In the case when past returns were used, *k*-NN based methods were the most consistently profitable, followed by the linear ridge regression and quadratic ridge regression. Neural networks, while being able to profit on some of the time series, did not achieve profit on most of the others. No evidence was further found of the PCA method to improve the results of the tested methods in a systematic way. In the second part of the study, the models were applied to empirical foreign exchange rate time series. Overall the profitability of the methods was rather low, with most of them ending with a loss on most of the currencies. The most profitable currency was EURUSD, followed by EURJPY, GBPJPY and EURGBP. The most successful methods were the linear ridge regression and the Manhattan distance based *k*-NN method which both ended with profits for most of the time series (unlike the other methods). Finally, a forward selection procedure using the linear ridge regression was applied to extend the original predictor set with some technical indicators. The selection procedure achieved limited success in improving the out-sample results for the linear ridge regression model but not the other models.

AMS/JEL classification: C45, C63, G11, G14, G17

Keywords: Ridge regression, *k*-Nearest Neighbour, Artificial Neural Networks, Principal Component Analysis, Exchange rate forecasting, Investment strategy, Market efficiency

Acknowledgments

This paper has been prepared under financial support of a grant “Advanced methods of financial asset returns and risks modelling” IGA VŠE F1/23/2015 which the author gratefully acknowledges.

Introduction

Forecasting the conditional mean of financial time series returns is among the most challenging and elusive tasks in financial econometrics, while at the same time being at the centre of the attention of many market participants such as hedge funds, commodity trading advisors as well as individual speculators. As there is a relative lack of linear dependencies in the financial time series returns (apart from the ultra-low and the ultra-high frequencies as mentioned in Christoffersen and Diebold 2003) and the possible non-linear dependencies are often viewed as too complex to be specified analytically, the academicians and practitioners often turn their focus to artificial intelligence and data mining methods, which do not require the analyst to define the relationships in the time series in advance, but instead identify them on their own. Wide variety of data mining methods have been applied for financial time series forecasting, such as neural networks, support-vector-machines, k-nearest neighbour, k-means clustering, classification trees, random forests, etc. (for a survey see Krollner, Vanstone and Finnie 2010, or Calvacante et. al. 2016). While most of the performed studies report that their data mining models outperformed the utilized benchmark (80% of the studies based on the survey in Krollner, Vanstone and Finnie 2010), as authors of the survey mention, the performed studies often do not consider real world trading costs such as spreads and slippage. Furthermore, it is difficult to assess if the methods beat the market on a risk-adjusted basis, including the model risk, and it is arguably even more difficult, to evaluate to what degree are the results of the published studies influenced by issues such as selection bias and publication bias, causing the results to be fitted to the noise in the out-sample period (for an approach of how to cope with this kind of overfitting during the research, see Bailey, Borwein, Prado and Zhu 2015).

Overfitting is among the most significant problems in financial time series forecasting, due to their limited length (especially for the daily and weekly frequencies), non-linearity of the underlying relationships (leading to a need of more complex models to capture them) and the low signal-to-noise ratio in these time series (i.e. high levels of noise), coupled with a highly persistent stochastic volatility of the random component. Another problem associated with the application of data mining methods for financial time series forecasting is the strong dependency of the performance of these methods on the choice of the meta-parameters of the models (the number of hidden layers, hidden neurons, learning rate, iterations and the activation function for neural network models, the number of nearest neighbours and the neighbourhood function for the *k*-nearest neighbour method, etc.). A caution is thus needed in the study design to not base the selection of these meta-parameters on the performance of the models in the same sample as the one used for results evaluation as this would lead to overfitting to the noise to this simple and thus lead to biased results.

Among efficient methods used to cope with the meta-parameter selection in the study design is the partition of the data sample into a training sample, validation sample and a testing sample (the first being used for parameter estimation, the second for meta-parameter optimization and the third for results evaluation), the use of a k-fold, or of a leave-one-out cross-validation and cross-verification (Martens and Dardenne, 1998), or the newly proposed combinatorically symmetric cross-validation (CSCV) devised in Bailey, Borwein, Prado and Zhu (2015). In our study, the partition of the analyzed time series into a training sample, validation sample and testing sample will be used.

Another problematic issue of financial time series forecasting is the time-varying nature of the dependencies in these time series, caused by the evolving character of the markets towards even greater levels of information efficiency (in line with the adaptive market hypothesis proposed by Lo, 2004). As the relationships identified by large enough fraction of market participants tend to get arbitraged away by their actions, the quantitative investors and hedge funds willing to keep their edge and achieve above-average risk-adjusted returns, need to constantly innovate their algorithms,

coming up with ever more sophisticated trading strategies. This raises a question to what degree are the positive results achieved in the past studies, performed even just several years ago, still relevant to the current market situation, as well as how far are the current markets from reaching the weak for of market information efficiency (proposed by Fama, 1970), which would make all attempts of financial time series forecasting based on their past history futile.

In order to evaluate if the standard data mining tools can predict the movements of the current financial markets, we apply three well known data mining methods, namely the k-Nearest Neighbour classifier, the Ridge Regression and the Multilayer Perceptron Feed-Forward Neural Networks in order to forecast the currency exchange rate time series of 10 major currency exchange rates, in the one-day horizon, in the period from 1.11.1999 to 12.6.2015, with the testing sample covering the period after 13.5.2011. The testing will be performed in a statistically robust way, using the partition of the data into a training sample, validation sample and testing sample, in order to avoid the look-ahead bias. The methods will be applied firstly by using a simple set of explanatory variables comprising of the exchange rate movements in the last 5 days. In next part of the study a selection procedure will be devised in order to extend the dataset with a series of technical indicators, acting in the terminology of data mining research as non-linear feature extractors. In all of the cases, apart from the direct application of the methods to the predictors, dimensionality reduction of the variable set is performed via the Principal Component Analysis (PCA) method, with an optimized number of principal components used as predictors instead of the original variables, in order to cope with the curse of dimensionality and the multicollinearity in the explanatory variable set.

Furthermore, in addition to the empirical study on the 10 foreign exchange rate time series, a detailed simulation study is performed, in which the utilized data mining methods are applied to 10 realistically simulated time series of foreign exchange rate returns, with different non-linear dependency patterns added, governing the evolution of their conditional mean. The simulation of the financial time series is performed via a realistic Stochastic-Volatility Jump-Diffusion (SVJD) model with self-exciting jumps, whose parameters were estimated on the EURUSD time series with MCMC algorithm. The purpose of the simulation study is to evaluate, to what degree are the utilized data mining methods able to learn and predict random non-linear patterns occurring in simulated time series under realistic market conditions of high levels of noise, persistent stochastic volatility, as well as self-exciting jumps, representing an additional, discontinuous noise component.

As to our knowledge, this study is the first to evaluate the performance of data mining methods to model patterns in realistically simulated financial time series with stochastic volatility and jumps.

The rest of the paper is organized as follows. In Section 1 we present the data mining methods used in the study, specifically the *k*-Nearest Neighbour method, the linear and polynomial Ridge Regression and the Multilayer-Perceptron Feed-Forward neural networks. In addition to that, we explain the Principal Component Analysis, used for the purposes of dimensionality reduction of the predictor matrix, as well as the utilized technical indicators used as feature extractors and predictors in the models. In Section 2 is the simulation study in which the presented data mining methods are applied to recover a set of 10 different non-linear patterns from a realistically simulated financial time series using a SVJD model with self-exciting jumps. Several different variants of the models are tested (*k*-NN with 4 different distance functions, FFNN with 1 to 3 hidden layers, etc.), in a version with and without the additional technical indicator based predictors, in both cases in a either with or without the dimensionality reduction performed via the PCA method. In Section 3, the same methods are applied, in order to forecast (one-step-ahead) 10 real world financial time series of currency exchange rate movements, spanning over a period of more than 15 years. Finally, the last section contains a conclusion, in which a summary of the results of the tested models is found, as

well as a brief discussion regarding the future of the application of data mining methods for financial time series forecasting.

1. Utilized data mining methods

Three data mining methods are used in this study, namely the *k*-Nearest Neighbour (*k*-NN) method, the Ridge Regression and the Multilayer Perceptron Feed-Forward Neural Network (FFNN). In addition to that, the Principal Component Analysis (PCA) is applied for dimensionality reduction for the variable set and a series of technical indicators are applied as feature extractors and predictors. In this section all of the methods are briefly introduced.

1.1 *k*-Nearest Neighbour method

A popular data mining tool that can be used for financial time series forecasting is the *k*-nearest-neighbour method, in the form of either the *k*-NN classifier or the *k*-NN regression (see …). The logic of this method is intuitive. In order to predict the behaviour of the analysed time series in the future, we look into the past to find situations in which the time series behaved similarly as it is behaving now. By utilizing an objective similarity measure (such as the Euclidian distance between a vector describing the current state of the time series and a vector describing its state at a given time-point in the past), we select *k* occasions, in which the behaviour of the time series was most similar to the current one. We then look at the subsequent behaviour of the time series following these *k* time points, we calculate the desired statistics based on them (the mean, the median, the standard deviation, a specific quantile or whatever other quantity we want to predict) and use them as a forecast of the given quantity at the current time point into the future.

The k-nearest-neighbour method differs from most of the other data mining algorithms used for time series prediction, such as neural networks and support vector machines, in the fact that it is a local model of time series, while most of the other methods are global models. Local models of time series do not try to capture their entire dynamics with a single equation or a set of equations, but instead model each instance of the time series separately (i.e. there is basically a separate model for every instance), by utilizing similar observations. This mechanism makes these methods especially well suited for the prediction of time series with time-varying dynamics as well as the chaotic time series, for which they won multitude competitions in the past (McNames 2000), and even nowadays, they provide state-of-the art results for many time series prediction tasks (Bernas and Plasczek 2016).

The *k*-Nearest Neighbour algorithm is an universal data mining tool, that can be used, with slight modifications, for many different classes of prediction tasks (the main distinction being between classification and regression, based on whether is the variable discrete or continuous). In what follows, a version of the algorithm used for regression over time series data will be presented.

The K-Nearest Neighbour algorithm as used for time-series regression can be mathematically defined as follows (the description works with asset return time series but price changes could be used alternatively without any further modifications of the algorithm).

Assume that we have a time series of logarithmic returns, r_{t+1}, defined as $r_{t+1} = p_{t+1} - p_t$, where p_{t+1} denotes the logarithm of the asset price at the end of the time period $t + 1$ (in our case the period represents a single trading day, with p_t being its closing price). We want to construct a forecast of the expected mean one-period-ahead return $E(r_{t+1}|\mathcal{F}_t)$, with $\mathcal{F}_t$ being the available information at the end of the time period t.

A *k*-NN algorithm enabling sequential prediction of all of the returns r_t in a the time series of length T would proceed as follows.

Let us assume we have a financial time series $S = r_1, r_2, \dots r_T$ of asset returns r_t of length T. We further use the time series to define a set of T pairs of observations:

$$\boldsymbol{\Phi} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \dots\ (\boldsymbol{x}_T, \boldsymbol{y}_T)\}$$

With $\boldsymbol{x}_t$ denoting the vector of available predictors at time t, and $\boldsymbol{y}_t$ denoting the vector of target quantities to be predicted at time t. We further assume that the vector of predictors $\boldsymbol{x}_t$ contains the n past returns of the analysed time series, $r_{t-n}, r_{t-n+1}, \dots, r_t$, as well as a set of m technical indicators, whose values at time t are given by some function of the past returns up until time t. The value of an indicator i at time t is given as $f_{i,t} = f(r_1, r_2, \dots, r_t)$. The value of the predictor vector $\boldsymbol{x}_t$ can then be expressed as $\boldsymbol{x}_t = \{r_{t-n}, r_{t-n+1}, \dots, r_t, f_{1,t}, f_{2,t}, \dots, f_{m,t}\}$.

The value of the target vector $\boldsymbol{y}_t$ at each time point t is equal to the subsequent q returns that we want to predict, with q representing the forecast horizon (alternatively we could predict the cumulative return up to time $t+q$, i.e. $r_{t+1:r+q}$). We can then write $\boldsymbol{y}_t = \{r_{t+1}, r_{t+2}, \dots, r_{t+q}\}$.

To implement the *k*-Nearest Neighbour algorithm it is further necessary to define the utilized distance function $d(\boldsymbol{p}, \boldsymbol{q})$ measuring the distance between its input vectors $\boldsymbol{p}$ and $\boldsymbol{q}$. There are many possible distance functions that can be used in the *k*-NN algorithm, with four of the most commonly used ones being further discussed later in this section.

In order to construct a prediction of the target vector $\widehat{\boldsymbol{y}}_t$, it is necessary to order all of the past observations, i.e. the $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ pairs up until time $t-1$, based on the distance of their predictor vectors $\boldsymbol{x}_i$ from the current predictor vector $\boldsymbol{x}_t$. We thus get an ordered set of $t-1$ observations $\boldsymbol{\Phi}^*$, with $(.)$ in the lower index denoting their order. I.e. we get a set:

$$\boldsymbol{\Phi}^* = \{(\boldsymbol{x}_{(1)}, \boldsymbol{y}_{(1)}), (\boldsymbol{x}_{(2)}, \boldsymbol{y}_{(2)}), \dots\ (\boldsymbol{x}_{(t-1)}, \boldsymbol{y}_{(t-1)})\}$$

For which it holds that:

$$d(\boldsymbol{x}_{(1)}, \boldsymbol{x}_t) \leq d(\boldsymbol{x}_{(2)}, \boldsymbol{x}_t) \leq \cdots d(\boldsymbol{x}_{(n-1)}, \boldsymbol{x}_t)$$

For the purposes of this study, we further focus on the one-period-ahead forecasts of the asset price returns (or alternatively the movements). The target vector is thus one-dimensional and equal to $\boldsymbol{y}_t = y_t = r_{t+1}$. To construct the forecast of the target, we use a method known as *k*-NN regression, in which the forecast $\hat{y}_t = \hat{r}_{t+1}$ is calculated as an arithmetic average of the returns chronologically following after the *k* nearest neighbour observations (with *k* being a meta-parameter of the model). The forecast of the one-period-ahead return $\hat{r}_{t+1}$ can be calculated as follows:

$$\hat{r}_{t+1} = \widehat{y_t} = \frac{1}{k} \sum_{u=1}^{k} y_{(u)}$$

Where $\hat{r}_{t+1} = E(r_{t+1}|\mathcal{F}_t)$ denotes the one-period-ahead return forecast based on information set $\mathcal{F}_t$ up until the time point t, $y_{(u)}$ denotes the returns following chronologically on the following day after the k nearest neighbour observations to $\boldsymbol{x}_t$, and k denotes the number of *k* nearest neighbours utilized to construct the forecast.

A crucial part of the *k*-NN algorithm is the distance function $d(\boldsymbol{p}, \boldsymbol{q})$ used for the calculation of the nearest neighbour. There are multiple options for the choice of the distance function to be used for this purpose (with more advanced studies utilizing learnable metrics such as Goldberger et al. 2005 or Weinberger et al. 2006).

In our study, the following 4 common distance functions will be used.

- **Euclidian distance –** The Euclidian distance between two points with Cartesian coordinates defined by vectors $\boldsymbol{p} = (p_1, p_2, \dots, p_n)$ and $\boldsymbol{q} = (q_1, q_2, \dots, q_n)$ in a *n*-dimensional Euclidian space is defined as the length of the line connecting the two points, which is, according to the Pythagorean formula, given as follows:

$$d_{euclid}(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

  Where $d_{euclid}(\boldsymbol{p}, \boldsymbol{q})$ is the Euclidian distance between the points $\boldsymbol{p}$ and $\boldsymbol{q}$, p_i and q_i are their coordinates and n is the dimension of the Euclidian space.
- **Manhattan distance –** Similarly as in the previous case, for two vectors in a *n*-dimensional space, denoted by $\boldsymbol{p} = (p_1, p_2, \dots, p_n)$ and $\boldsymbol{q} = (q_1, q_2, \dots, q_n)$, Manhattan Distance can be defined as the sum of the absolute distances between their coordinates:

$$d_{manhattan}(\boldsymbol{p}, \boldsymbol{q}) = \sum_{i=1}^{n} |p_i - q_i|$$

  Where $\boldsymbol{p}$ and $\boldsymbol{q}$ are two vectors, p_i and q_i are their coordinates and n is the dimension of the space (i.e. the vectors).
- **Mahalabonis distance** – To define Mahalabonis distance for the purposes of the *k*-NN classifier, we assume $\boldsymbol{p} = (p_1, p_2, \dots, p_n)$ and $\boldsymbol{q} = (q_1, q_2, \dots, q_n)$ to be *n*-dimensional random vectors, coming from the *n*-dimensional distribution D, with covariance matrix $\boldsymbol{S}$. Mahalabonis distance between the two observations, $\boldsymbol{p}$ and $\boldsymbol{q}$, can then be defined as follows:

$$d_{mahalabonis}(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{(\boldsymbol{p} - \boldsymbol{q})^T \boldsymbol{S}^{-1} (\boldsymbol{p} - \boldsymbol{q})}$$

- **Maximum distance** – Denoting $\boldsymbol{p} = (p_1, p_2, \dots, p_n)$ and $\boldsymbol{q} = (q_1, q_2, \dots, q_n)$ as two vectors in a *n*-dimensional space, the Maximum distance is defined as the maximum absolute distance between any two coordinates of these vectors. This can be expressed as follows:

$$d_{max}(\boldsymbol{p}, \boldsymbol{q}) = \max(|p_1 - q_1|, |p_2 - q_2|, \dots, |p_n - q_n|)$$

Additionally, before applying the distance function to the predictor vectors in the *k*-NN method, it is common to standardize the predictor variables, either by projecting their values between zero and one, or by subtracting their mean and dividing them by the standard deviation. For the purposes of this study the second approach is used, with the mean and the standard deviation used for the normalization computed from the training sample data.

1.2. Ridge Regression

Another popular data mining method, that can be used for financial time series prediction (either alone or as a readout of a non-linear model such as kernel ridge regression, Exterkate et al. 2012), is the Ridge Regression, representing a penalized version of the standard linear (or possibly non-linear) regression, with a penalty term added, penalizing the absolute size of the model parameters, in order to prevent over-fitting. Due to the high levels of noise in financial time series, over-fitting is of serious concern in their modelling and forecasting, and the Ridge Regression is able to alleviate this risk to a significant degree and thus achieve more robust parameter estimates and target variable forecasts than the standard, un-penalized, regression models.

The equation for linear Ridge regression is the same as for the standard OLS linear regression and can be expressed, by using a matrix notation, as follows:

$$\boldsymbol{Xb} = \boldsymbol{y}$$

Where $\boldsymbol{X}$ is the $(T \text{ x } n)$ matrix of predictors, with T being the number of observations and n the number of predictors including the constant term (i.e. a vector of ones), $\boldsymbol{y}$ is a $(T \text{ x } 1)$ vector of the target variable and $\boldsymbol{b}$ is a $(n \text{ x } 1)$ vector of the regression model parameters.

Ridge Regression differs from the standard OLS linear regression in the estimation process of the parameter vector $\boldsymbol{b}$, as it adds a penalization term for the L_2 norm of the parameter vector to the sum of the squared residuals, that needs to be minimized.

The term to be minimized to find the parameter vector $\boldsymbol{b}$ in a standard OLS regression is:

$$\|\boldsymbol{Xb} - \boldsymbol{y}\|$$

The term to be minimized to find the parameter vector $\boldsymbol{b}$ in the Ridge Regression is:

$$\|\boldsymbol{Xb} - \boldsymbol{y}\| - \|\boldsymbol{\Gamma b}\|$$

Where $\|.\|$ denotes the Euclidian norm and $\boldsymbol{\Gamma}$ is the so called Tikhonov matrix, which is commonly chosen to be equal to $\boldsymbol{\Gamma} = \boldsymbol{\alpha I}$, where $\boldsymbol{I}$ denotes the identity matrix and $\boldsymbol{\alpha}$ is the penalization term determining strength of the penalization of the large values of the parameter vector $\boldsymbol{b}$. The penalization of large values of $\boldsymbol{b}$ can be interpreted as putting a zero-centered multivariate normal Bayesian prior on the distribution of the model parameters, which can, in many applications, dramatically reduce the extend of overfitting of the regression model.

By calculating the derivatives of the term to be minimized with respect to the parameter values, and by putting them equal to zero (to identify an extremum), it is possible to derive an analytical equation for the estimation of the model parameters:

$$\widehat{\boldsymbol{b}} = (\boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{\Gamma}^T\boldsymbol{\Gamma})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Where $\widehat{\boldsymbol{b}}$ denotes the estimate of the parameter vector.

While a wide variety of extended, non-linear, ridge regression based data mining methods exist (kernel ridge regression, extreme learning machines, echo state neural networks, etc.), in our study the ridge regression is used only in its most basic form (as a benchmark model), specifically as the linear and quadratic ridge regression. For the quadratic regression we additionally decided to use only the squared terms in the model equation and not the mixed-product terms, capturing interrelations in the predictors, due to their high number which seem to lead to overfitting in spite of the performed regularization.

The equation of the linear ridge regression model in which the last $n + 1$ returns and m technical indicators are used as predictors, can be written as follows (an analogical model will also be used with price movements instead of returns):

$$r_{t+1} = \beta_0 + \sum_{i=1}^{n+1} \beta_i r_{t-i+1} + \sum_{j=1}^{m} \beta_{n+j+1} f_{j,t} + \varepsilon_{t+1}$$

Where r_t denotes the logarithmic return at time t, $f_{j,t} = f(r_1, r_2, \dots, r_t)$ is a technical indicator calculated from the past evolution of r_t as feature extractor, $n + 1$ denotes the number of last returns used in the model, m denotes the number of technical indicators used in the model, $\beta_0, \beta_1 \dots \beta_{n+m+1}$ are the estimated model parameters and ε_{t+1} is the $i.i.d$ Gaussian white noise term at time $t + 1$. The model has altogether $n + m + 1$ parameters to be estimated.

Similarly, the equation for the quadratic ridge regression would be:

$$r_{t+1} = \beta_0 + \sum_{i=1}^{n+1}\left(\beta_i r_{t-i+1} + \gamma_i r_{t-i+1}^2\right) + \sum_{j=1}^{m}\left(\beta_{n+j+1} f_{j,t} + \gamma_{n+j+1} f_{j,t}^2\right) + \varepsilon_{t+1}$$

With r_t^2 denoting the squared returns, $f_{j,t}^2$ the squared technical indicators, and $\gamma_0, \gamma_1 \dots \gamma_{n+m+1}$ the parameters corresponding to the quadratic terms of the regression equation. The model has altogether $2 * (n + m) + 1$ parameters to be estimated.

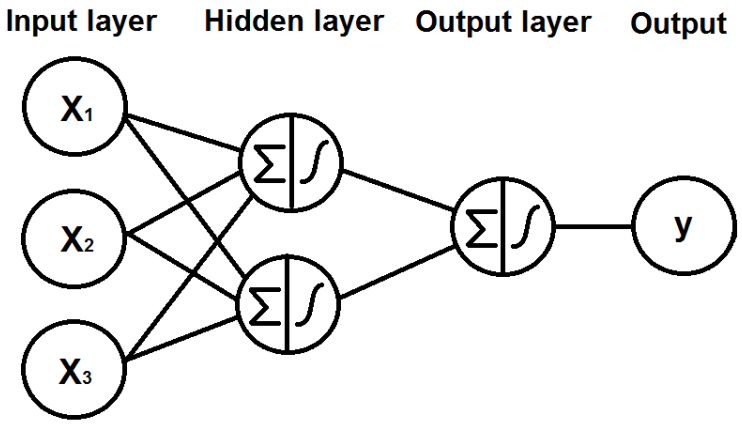
1.3. Multilayer Perceptron Feed-Forward Neural Networks

The third method used in our study is the Multilayer Perceptron Feed-Forward Neural Network. While Ridge Regression is a powerful technique, its drawback is that it captures only the linear relationships between the explanatory variables (i.e. past returns or technical indicators) and the target variable (future returns). In order to handle non-linear dependencies in the analyzed time series it is possible to use non-linear regression (such as the quadratic ridge regression), the regression would still, however, capture only certain kinds of relationships between the predictors and the targets as defined by the regression equation. In a situation where we do not know what relationships the analysed time series contains, it may be useful to utilize neural networks, which act as universal function approximators (Funahashi, 1989, Hornik, 1989), and are thus able to capture (with enough neurons and hidden layers) any possible relationship between the predictors and the target variable. The larger the network is, the more powerful it is with regards to what kind of non-linear relationships it can capture, its however also more prone to overfitting (i.e. fitting itself to the noise in the time series), which makes the use of large neural networks on the relatively short financial time series with high levels of noise problematic. In spite of this, neural networks are among the most popular data mining methods used for financial time series modelling (Lawrence 1997, Krollner, Vanstone and Finnie 2010, or Calvacante et. al. 2016).

For the purpose of this study, a Multilayer Perceptron Feed-Forward Neural Network (FFNN) is used, with a logistic activation function and alternatively 1, 2 or 3 hidden layers and optimized number of neurons (5, 10, 15 or 20), by using the validation period. The learning rate and the number of iterations used for network training (via a stochastic backpropagation of error algorithm) are optimized as well, to avoid overfitting.

Feed-Forward Neural Network is a network in which the connections proceed in a single direction, from the predictors, through the hidden layers up to the output layer. Schematically, the Multilayer Perceptron neural network with a single hidden layer can be depicted as shown on Figure 1:

Figure 1 – Multilayer Perceptron Feed-Forward Neural-Network with single hidden layer



The processing of information in the neural network can be described as follows:

1. The Input layer contains the inputs x_i for $i = 1, \dots, n$, which are, after normalization (in our case through subtracting of the mean and dividing with a standard deviation) propagated to the hidden layer
2. Each of the j neurons in the Hidden layer calculates a linear combination of its inputs by using the weights $w_{i,j}^{(1)}$, where i denotes the input, j the neuron and (1) the first hidden layer, and then applies a non-linear (in our case logistic) activation function to the value of the linear combination, the output of this operation is then propagated to the Output layer
3. The neuron in the Output layer calculates a linear combination of its inputs by using the output weights $w_j^{(2)}$, it may further apply its own activation function to the output, in our case, however, the output activation function will be the identity function (as we want to predict the returns, or price changes, which are both real valued)
4. The output of the Output layer is the forecast of the neural network

For the case of a multilayer perceptron with a single hidden layer with *k* neurons, hidden layer activation function $f_{Hidden}(.)$ and output layer activation function $f_{Output}(.)$, it is possible to calculate the output y from a set of *n* inputs x_i, where $i = 1, \dots, n$, as follows:

$$y = f_{Output}\left[\sum_{j=1}^{k} w_j^{(2)} f_{Hidden}\left(\sum_{i=1}^{n} w_{i,j}^{(1)} x_i\right)\right]$$

Where x_i denotes the inputs, $w_{i,j}^{(1)}$ the weights in the hidden layer, $w_j^{(2)}$ the weights in the output layer, y the output of the neural network, $f_{Hidden}(.)$ the hidden layer activation function and f_{Output} the output layer activation function. As already mentioned, in our case, the hidden layer activation function is the logistic function, $f_{Hidden}(z) = \frac{1}{1+e^{-z}}$, while the output layer activation function is identity function, $f_{Output}(z) = z$.

The relationship between the output y and a set of inputs x_i, where $i = 1, \dots, n$, expressed as a vector $\boldsymbol{x}$, can alternatively be written by using a matrix notation as follows:

$$y = f_{Output}\left[\boldsymbol{w_2^T} f_{Hidden}\left(\boldsymbol{x^T W_1}\right)^T\right]$$

Where $\boldsymbol{x^T}$ is a vector of the n inputs x_i (for $= 1, \dots, n$), $\boldsymbol{W_1}$ is a $(n \text{ x } k)$ matrix of the weights in the hidden layer $w_{i,j}^{(1)}$ and $\boldsymbol{w_2^T}$ is a row vector of the k weights $w_j^{(2)}$ $(j = 1, \dots, k)$ in the output layer

For the training of the neural network, a Stochastic version of the Backpropagation of Error algorithm will be utilized, which is just a modified version of the Gradient Descend algorithm. The logic of the gradient descent method is to iteratively move the parameter values in the direction of the negative value of the gradient of the error function (starting from an initial set of parameter values), until an optimum (in this case minimum of the sum of squared residuals) is found. The error function is our case defined as follows:

$$Err = \frac{1}{2}(t - y)^2$$

In order to apply the algorithm, it necessary to compute the gradient, i.e. to calculate the derivatives of the error function with respect to the model weights (i.e. in the case of the single-hidden-layer neural network, these are the output layer weights and the hidden layer weights).

By using the chain rule, it is possible to express the derivative of the error function with respect to the output neuron weights $w_j^{(2)}$ as follows:

$$\frac{\partial Err}{\partial w_j^{(2)}} = \frac{\partial Err}{\partial y}\frac{\partial y}{\partial IN_2}\frac{\partial IN_2}{\partial w_j^{(2)}}$$

And it can easily be calculate to be (for our choice of the output activation function) equal to:

$$\frac{\partial Err}{\partial w_j^{(2)}} = (y - t)x_j^{(1)}$$

As $\frac{\partial Err}{\partial y} = (y - t)$, $\frac{\partial y}{\partial IN_2} = 1$ and $\frac{\partial IN_2}{\partial w_j^{(2)}} = x_j^{(1)}$

In line with the standard notation in the neural network literature (which is very useful in order to define the learning algorithms for more complex neural networks), the multiple of the first two derivatives is denoted as δ_2, so in our case the following holds:

$$\delta_2 = \frac{\partial Err}{\partial y}\frac{\partial y}{\partial IN_2} = (y - t)$$

In the next step it is necessary to derive the derivatives of the error function with respect to the hidden neuron weights $w_{i,j}^{(1)}$. By using the chain rule it is possible to express them as follows:

$$\frac{\partial Err}{\partial w_{i,j}^{(1)}} = \frac{\partial Err}{\partial y}\frac{\partial y}{\partial IN_2}\frac{\partial IN_2}{\partial x_j^{(1)}}\frac{\partial x_j^{(1)}}{\partial IN_{1,j}}\frac{\partial IN_{1,j}}{\partial w_{i,j}^{(1)}}$$

By using the derived value of δ_2 and computing the other derivatives, we can rewrite the expression:

$$\frac{\partial Err}{\partial w_{i,j}^{(1)}} = \delta_2 w_j^{(2)} x_j^{(1)}\left(1 - x_j^{(1)}\right)x_i$$

As $\frac{\partial IN_2}{\partial x_j^{(1)}} = w_j^{(2)}$, $\frac{\partial x_j^{(1)}}{\partial IN_{1,j}} = x_j^{(1)}\left(1 - x_j^{(1)}\right)$ and $\frac{\partial IN_{1,j}}{\partial w_{i,j}^{(1)}} = x_i$.

In line with the standard neural network notation, we define $\delta_{1,j}$ as:

$$\delta_{1,j} = \frac{\partial IN_2}{\partial x_j^{(1)}}\frac{\partial x_j^{(1)}}{\partial IN_{1,j}} = w_j^{(2)} x_j^{(1)}\left(1 - x_j^{(1)}\right)$$

After all of the derivatives and the deltas, δ_2 and $\delta_{1,j}$ are computed, we can proceed with the gradient descend algorithm by deriving the expressions for the weight updates.

The gradient descent weight adjustment for the output layer can then be expressed as:

$$\Delta w_j^{(2)} = -\eta\delta_2 x_j^{(1)}$$

And for the weights in the hidden layer as follows:

$$\Delta w_{i,j}^{(1)} = -\eta\delta_2\delta_{1,j}x_i$$

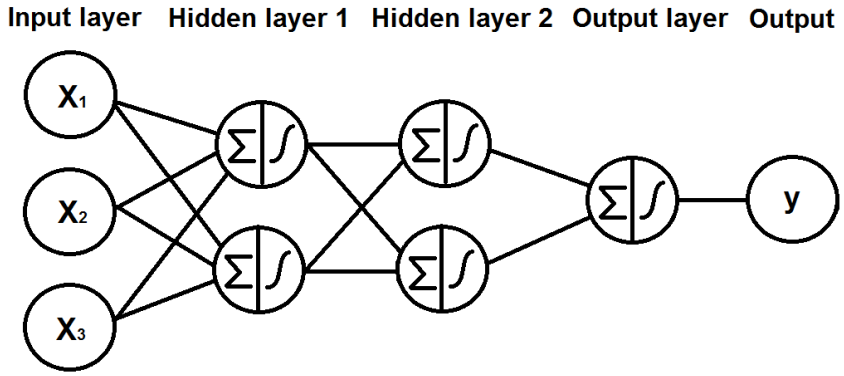
Where η denotes the learning rate, determining the speed of how quickly do the weights move in the direction of the negative of the gradient in a single iteration. The value of η is an important meta-parameter of the neural network, as it does, together with the number of iterations N, influence whether the network will under-fit or over-fit to the training data. It is thus useful to determine its value based on a validation sample, or a cross-validation.

In practice, there are multiple possibilities of how the gradient descent algorithm may proceed to train the neural network, distinguishing the *stochastic, batch* and *online* version of the algorithm. In this study, the stochastic version will be used, in which single observations are picked randomly from the training sample, the gradient is computed based on the single observation and the weights are changed accordingly. The stochastic gradient descent has the advantage of arriving at wholly different neural networks in each training run of the algorithm as it is every time trained on a different set of randomly chosen observations. The predictions of so trained networks can then be combined into an envelope which usually has better characteristics and is less prone to overfitting than if only a single network was trained via some other method. In our case envelopes of 100 networks will be trained and their predictions averaged (using an unweighted arithmetic average) for each of the tested models.

While the multilayer perceptron with a single hidden layer is a powerful universal function approximator, in practical settings it may have problems to approximate more complex dependencies between the inputs and the outputs. The performance of the neural network can, in such cases, often be dramatically increased by adding multiple hidden layers into the network (so called deep learning). As the relationships in the financial time series may be quite complex, but the series are short and contain high levels of noise, we will test feed forward neural networks with up to 3 hidden layers in the empirical section of this study.

Figure 2 shows schematically the multilayer perceptron with 2 hidden layers.

Figure 2 - Multilayer Perceptron Feed-Forward Neural-Network with two hidden layers



The processing of the inputs and their transformation into the output is analogical in the two-hidden-layer networks as in the network with only a single hidden layer. Mathematically the functioning of the network can be expressed as follows:

$$y = f_{Output}\left\{\sum_{u=1}^{m} w_u^{(3)} f_{Hidden}\left[\sum_{j=1}^{k} w_{j,u}^{(2)} f_{Hidden}\left(\sum_{i=1}^{n} w_{i,j}^{(1)} x_i\right)\right]\right\}$$

Where $w_u^{(3)}$ denotes the weights in the output layer, $w_{j,u}^{(2)}$ and $w_{i,j}^{(1)}$ the weights in the two hidden layers (with *m* and *k* neurons), x_i the inputs of the neural network and y the output.

Alternatively the same can be rewritten by using a matrix notation as follows:

$$y = f_{Output}\{f_{Hidden}[f_{Hidden}(\boldsymbol{x^T W_1})\boldsymbol{W_2}]\boldsymbol{w_3}\} \tag{21}$$

Where $\boldsymbol{x^T}$ is a row vector of the $(1 \text{ x } n)$ inputs, $\boldsymbol{W_1}$ is the $(n \text{ x } k)$ matrix of the weights in the first hidden layer $w_{i,j}^{(1)}$, $\boldsymbol{W_2}$ is the $(k \text{ x } m)$ matrix of the weights in the second hidden layer $w_{j,u}^{(2)}$, and $\boldsymbol{w_3}$ is a column vector $(m \text{ x } 1)$ of the weights in the output layer $w_u^{(3)}$.

The training of the neural network is again performed via the *backpropagation of error algorithm*, which can be, by using the delta notation, extended to an arbitrary number of hidden layers.

Assuming a total of H layers in the neural network (i.e. hidden + output layer), the output layer delta, $\delta^{(H)}$, can be expressed (for the case of the identity activation function in the output layer) as follows:

$$\delta^{(H)} = (y - t)$$

The $\delta_q^{(h)}$ corresponding to a neuron q, in a hidden layer h (for $h < H$) can then be expressed as:

$$\delta_q^{(h)} = \left(\sum_r \delta_r^{(h+1)} w_{q,r}^{(h+1)}\right) x_q^{(h)} \left(1 - x_q^{(h)}\right)$$

Where $\delta_q^{(h)}$ is the delta for the neuron q in the layer h, $\delta_r^{(h+1)}$ is the delta for the neuron r in the layer $h+1$, $w_{q,r}^{(h+1)}$ is a weight of the connection between the neuron q and neuron r, and $x_q^{(h)}$ is the output of the neuron $x_q^{(h)}$.

Thus, starting with the deltas in the output layer $\delta^{(H)}$ and moving through the structure of the neural network backwards, the value of the deltas $\delta_q^{(h)}$ for all of the neurons q can be calculated.

The weights adjustment for any weight $w_{p,q}^{(h)}$, in any layer h (including H), would then be as follows:

$$\Delta w_{p,q}^{(h)} = -\eta \delta_q^{(h)} x_p^{(h-1)}$$

Where $w_{p,q}^{(h)}$ is the connection between the neuron (or input) p and neuron q in layer h, $\delta_q^{(h)}$ is the delta of the neuron q and $x_p^{(h-1)}$ is the output of the neuron p, which is eventually, in the case of $h = 1$ (and thus $h - 1 = 0$) equal to the input to the neural network x_p.

As already mentioned, neural networks with alternatively 1, 2 and 3 hidden layers are used in the empirical section of this study, with optimized number of neurons, learning rate and number of iteration based on the results in the validation sample. The predictors and the target variable will be based either on past movements of the currency exchange rates or on their past returns, analogically as in the case of the *k*-NN model and the ridge regression model.

2.4. Principal Component Analysis

When large number of predictors, or highly correlated predictors, are used in data mining applications, the out-of-sample performance of many predictive models may significantly decrease, due to the problems known as the curse of dimensionality and multicollinearity.

Curse of dimensionality is a term used to describe a series of problems that emerge when a high number of predictors is used as input to a statistical model and a data mining algorithm. Although the problem affects to certain degree all of the methods used in this study, it is most pronounced in the case of the *k*-Nearest-Neighbour algorithm, whose performance is known to quickly deteriorate when the number of predictors used for the classification or regression becomes too high. The reason for this is that with large dimension of the vector, used to describe the state of the time series, and calculate its similarity to the past states, the distance between all of the observations becomes very similar, causing it to lose its predictive power.

The second mentioned problem – multicollinearity – does again affect most kinds of statistical models to a certain degree, it is, however, most problematic in the case of regression models, where causes problems ranging from biasing the diagnostic statistics and the standard errors, to dramatically increasing the overfitting during the estimation. While these problems are most pronounced in the case of OLS linear regression and the utilization of Ridge Regression does on itself represent a partial solution to the multicollinearity problem, if the correlation between the predictors is high enough, the problem of increase levels of overfitting may to a certain degree persist even in this case.

In order to cope with the curse of dimensionality and multicollinearity, it is necessary to perform dimensionality reduction and decorrelation of the predictor matrix, which can both be efficiently achieved with the method known as Principal Component Analysis (PCA).

Principal Component Analysis (PCA) is a statistical technique that uses an orthogonal transformation to transform a set of correlated variables into a set of linearly uncorrelated variables called principal components. The transformation proceeds so that the first principal component has the highest possible variance and each subsequent component has the highest possible variance under the condition that it is orthogonal to all of the preceding components. The output of the Principal Component Analysis, applied to a *n*-dimensional series of predictor variables, is thus a set of $p<n$ new variables that are uncorrelated and orthogonal to each other, with the first component having higher variance than the second, the second having higher variance than the third, etc.

When utilized in the *k*-Nearest Neighbour regression, or any other predictive model for that matter, the first $e<p$ components (with *e*, denoting the number of employed components, which is an additional meta-parameter of the model) are then used as the predictor variables instead of the original (high-dimensional and possibly correlated) set of original predictors. So, in the case of the *k*-NN algorithm, the principal components are used as the state vector, used for the calculation of the similarity between the current state of the time series and its historical states, in the ridge regression the principal components are used as the independent variables in the regression equation, etc.

2.5. Technical Indicators as Feature Extractors

In spite of the *k*-Nearest Neighbour method being able to model time-varying and chaotic time series, and Multilayer Perceptron Feed-Forward Neural Networks functioning as universal function approximators, in practical settings, the methods may still have problems to capture all of the hidden non-linear dependencies in the analysed time series, especially when working with time series of limited length and high levels of noise. In these cases, the predictive power of the methods may often be significantly increased, by extending or replacing the original dataset of the predictor variables (i.e. in the time series modelling tasks these are generally the past values of the time series), with variables derived from the original dataset, specifically designed to extract important features from it (i.e. features important for forecasting). The data mining techniques extending or replacing the original datasets with more informative set of predictors, often specifically designed for

the given data mining task, are called Feature Extraction techniques and process of doing so is called Feature Extraction.

One of the very popular used methods of feature extraction, universally applicable to a broad variety of data mining problems, is the Principal Component Analysis, discussed in the previous section as a method of dimensionality reduction and decorrelation of the predictor dataset. As already mentioned, we use this method in the empirical part of this study, as an alternative to the application of the models to raw data, in all of the performed modelling tasks.

Another method of feature extraction, which is specific only for the financial time series modelling problems, is the replacement of the returns or movements of the original time series with the values of technical indicators, developed by traders as a practical tool used for trading based on technical analysis of the market behaviour.

While technical indicators were not originally developed as tools for feature extraction for data mining methods, the do, indeed, perform the exactly same thing as feature extraction techniques. They apply a specific non-linear function to the past asset price behaviour, in order to extract important features from its evolution and make thus its behaviour more interpretable with regards to future movements prediction and trading signal detection. This makes technical indicators potentially very useful as feature extractors in data mining applications and the have indeed been used for this purpose in wide variety of past studies (…).

For the purposes of our study, we use 5 different technical indicators as feature extractors. All of the utilized indicators are from the sub-group of technical indicators called “oscillators”, whose main property is that they are oscillating around a given value, or in a given range. This is an important property for our purposes as from an econometric point of view, the oscillators are stationary (while other groups of technical indicators are not), and they can thus be directly used in the utilized models to forecast the also stationary target variable (future return or future movement of the foreign exchange rate).

The following oscillators will be used in our study: Momentum (MOM), Rate of Change (ROC), Relative Strength Index (RSI), Commodity Channel Index (CCI), Fast Stochastic Oscillator (FSO) and Internal Bar Strength Indicator (IBS). The utilized indicators were deliberately selected from the subset of oscillators, containing only a single optimizable parameter, corresponding to the period (i.e. number of past returns) from which it is calculated (some of the utilized indicators may actually have multiple optimizable parameters but we keep all of them, except for the one, fixed at commonly used values, while only the single parameter varies).

The choice of what technical indicators are to be added into the model will be decided based on a selection procedure that is described in the next section. Each indicator will then be tested with 10 different values of its free parameter, corresponding to the values of the Fibonacci series (although the Fibonacci series is sometimes used as a technical analysis tool on its own, is not based on any hypotheses about its predictive power on the financial markets, and any other, similar series could actually be used instead of it with similar results). The tested values of the parameters will be 2, 3, 5, 8, 13, 21, 34, 55, 89 and 144. An exception is then made in the case of the IBS indicator in whose case the parameter/period value of 1 will be tested as well as it is a common parameter option for this indicator, while for the other tested indicator the value of 1 cannot be used as meaningful option, as their construction requires at least two periods to be used.

Momentum Oscillator (MOM) is among the simplest technical indicators. It measures the change of the price over the last p periods, with p being the parameter of the model. It can be expressed as:

$$MOM_t(p) = P_t - P_{t-p}$$

Where $MOM_t(p)$ denotes Momentum at time t with period p, P_t is the asset price at time t and p Is the period (i.e. parameter) of the Momentum Oscillator.

Rate of Change (ROC) is an indicator similar to the Momenum, but based on relative changes instead (i.e. it corresponds to the return over the last p periods). ROC can be calculated as:

$$ROC_t(p) = \frac{(P_t - P_{t-p})}{P_{t-p}}$$

Where $ROC_t(p)$ denotes the Rate of Change indicator of period p at time t, P_t is the asset price at period t and p is the period (i.e. parameter) of the indicator.

Relative Strength Index (RSI) represents a more sophisticated (and very popular) technical indicator, measuring the proportion of the sum of all positive to all negative price movements over a period of p periods. The values of the indicator are additionally projected to the interval between 0 and 100 (which is a common practice in many technical indicators). RSI can be calculated as follows:

$$RSI_t(p) = 100 - \frac{100}{1 + \frac{SMA(U_t, p)}{SMA(D_t, p)}}$$

With U_t and D_t defined as $U_t = \max(P_t - P_{t-1}, 0)$ and $D_t = \max(P_{t-1} - P_t, 0)$, and $SMA(X_t, p)$ denoting a simple moving average of the input series X_t, with period p (which is the parameter of the indicator. $RSI_t(p)$ does then give us the value of the Relative Strength, at time t.

Commodity Channel Index (CCI) is a popular technical indicator with very intuitive statistical interpretation. It measures the distance of the current price from its *p*-period moving average, expressed in a number of moving mean absolute deviations. The value of the indicator is, nevertheless, rescaled, so that the value of +-100 corresponds to 1.5 mean absolute deviation distance from the moving average of the price. Additionally, instead of the closing price, the indicator performs all of the calculations by using the so called “typical price”, defined as an average of the high, low and closing price for the given trading day. CCI can be calculated as follows:

$$CCI_t(p) = \frac{1}{0.015} \frac{TP_t - SMA(TP_t, p)}{\sigma(TP_t, p)}$$

With TP_t denoting the typical price, defined as $TP_t = (H_t + L_t + C_t)/3$, where H_t is the daily high, L_t is the daily low and C_t is the daily closing price. $SMA(TP_t, p)$ does then represent the simple moving average of the daily typical price with period p and $\sigma(TP_t, p)$ is the moving mean absolute deviation, defined as $\sigma(TP_t, p) = SMA(AbsDev_t, p)$, with $AbsDev_t = |TP_t - SMA(TP_t, p)|$. Finally, the $CCI_t(p)$ is the value of a p-period CCI indicator at time t, with 0.015 being a scaling factor, rescaling the values of the indicator so that +-1.5 mean absolute deviations correspond to +-100.

Fast Stochastic Oscillator (SO) is an oscillator measuring the position of the current closing price to the range of the price behaviour (i.e. the range defined by the highest high and lowest low) over the last *p* periods. Fast and Slow versions of the stochastic oscillator exits, differing in the level of their smoothing (Fast Stochastic uses a 3-period simple moving average for the smoothing of the current price position compared to the past range, while the Slow Stochastic uses a double 3-period simple moving average, i.e. it calculates a 3-period out of a firstly calculated 3-period moving average). The Stochastic indicator utilizes a specific notation of the calculated components of the indicator, denoting them as $\%K_t$, $\%D_{fast,t}$ and $\%D_{slow,t}$. They can be defined as follows:

$$\%K_t(p) = \frac{P_t - LL_t(p)}{HH_t(p) - LL_t(p)}$$

$$\%D_{fast,t} = SMA(\%K_t, 3)$$

$$\%D_{slow,t} = SMA(\%D_{fast,t}, 3)$$

With $HH_t(p)$ and $LL_t(p)$ denoting the highest high and lowest low (in the last p periods), defined as $HH_t(p) = \max(H_t, H_{t-1}, \dots, H_{t-p+1})$ and $LL_t(p) = \min(L_t, L_{t-1}, \dots, L_{t-p+1})$, with H_t being the high at period t, L_t being the low and P_t being the close, $SMA(X_t, 3)$ denotes 3-period simple moving average of its input X_t, and $\%K_t(p)$, $\%D_{fast,t}$ and $\%D_{slow,t}$ being the three components of the stochastic oscillator (the latter two ones called as fast stochastic and slow stochastic).

Internal Bar Strength (IBS) is an indicator that may appear similar to the Stochastic Oscillator, but it does actually measure a different quantity (as long as $p > 1$). It is calculated as a p-period moving average from the percentage location of the daily closing price to the daily high-low range. IBS is thus telling us whether the prices tend to close near the daily maximus or the daily minimums. The indicator can be expressed as follows:

$$IBS_t(p) = SMA\left(\frac{C_t - L_t}{H_t - L_t}, p\right)$$

With H_t denoting the daily highest price, L_t the daily lowest price, C_t the daily closing price (equal to P_t in the notation for some of the previous indicators) and $SMA(X_t, p)$ denotes p-period simple moving average of its input X_t. $IBS_t(p)$ is then the Internal Bar Strength indicator with period p at time t.

Simulation study

In the first section of our empirical study, the tested methods are applied to a set of simulated time series of foreign exchange rate returns. The goal of this simulation study is to determine if the tested methods are able to capture the possible non-linear, noise-plagued, time-varying relationships that we expect to exist in the real world foreign exchange rate time series. If the methods achieve good results in the simulation study, they can be viewed as potentially suitable for real financial time series prediction. If, on the other hand, they do not capture the non-linear relationships in the simulated time series, we cannot expect them to perform any better in the empirical time series.

In order to perform the simulation, a Stochastic-Volatility Jump-Diffusion (SVJD) model with self-exciting jumps is estimated on the empirical time series of the EURUSD exchange rate (from 1.11.1999 to 12.6.2015), in order to get a set of reasonable, empirically realistic parameter values. These parameter values (with certain slight modifications) will then be used for the simulation of the time series in the simulation study, to which additional, non-linear dependency effects in the conditional mean will be added.

A SVJD model is used for the underlying process as it captures most of the empirically observable characteristics of the financial time series, such as stochastic volatility (i.e. time-varying standard deviation of the returns), jumps (i.e. large unexpected changes in the price, often caused by economic announcements) and the jump clustering caused by jump self-excitation (i.e. when a jump occurs it temporarily increases the probability of additional jumps occurring).

As the estimated SVJD model does not in itself contain any conditional mean dependencies that could be used to achieve additional trading profits, the simulated time series are extended with 10 kinds of non-linear conditional mean dependency patterns, in order to evaluate the ability of the utilized data mining methods to identify these patterns and to profit from their occurrence.

The utilized SVJD model contains 3 equations, one governing the evolution of the conditional mean of the logarithmic returns of the currency exchange rate, a second equation governing the evolution of the conditional volatility and a third governing the intensity of the jump occurrence.

The mean equation, governing the evolution of the logarithmic returns of the analysed exchange rate can be expressed as follows:

$$r(t) = \mu + \sigma(t)\varepsilon(t) + J(t)Q(t)$$

where $r(t)$ is the daily logarithmic return defined as $r(t) = p(t) - p(t-1)$ with $p(t)$ being the logarithm of the closing price at day t. Parameter μ represents the conditional mean of the daily returns, $\sigma(t)$ is the daily stochastic volatility, $\varepsilon(t) \sim N(0,1)$ is an *i.i.d.* standard normal random variable, $J(t) \sim N(\mu_J, \sigma_J)$ is a normally distributed random variable determining the size of the jumps and $Q(t) \sim \mathrm{Bern}[\lambda(t)]$ is a variable following a Bernoulli process with intensity $\lambda(t)$.

The equation governing the daily stochastic volatility uses an AR(1) process to model the behaviour of the logarithm of the conditional variance. The process is given as follows:

$$h(t) = \alpha + \beta h(t-1) + \gamma \varepsilon_V(t)$$

where $h(t) = \ln[\sigma^2(t)]$ is the logarithm of the daily conditional variance, $\alpha = (1-\beta)\theta$ is the long-term volatility, β is the autoregressive coefficient, γ is the volatility of the volatility and $\varepsilon_V(t) \sim N(0,1)$ is a series of standard normal *i.i.d.* random variables uncorrelated with $\varepsilon(t)$.

Finally, the third equation, governing the jump intensity $\lambda(t)$ is based on a discretized version of the self-exciting Hawkes process, and is given as follows:

$$\lambda(t) = \alpha_J + \beta_J \lambda(t-1) + \gamma_J Q(t-1)$$

where $\lambda(t)$ is the jump intensity at day t, $\alpha_\mathrm{J} = \left(1 - \beta_\mathrm{J} - \gamma_J\right)\theta_J$ determines the long-term jump intensity, β_J is the rate of the exponential decay of the jump intensity, and γ_J is the increase of jump intensity in the day following a jump occurrence.

As the model contains latent state variable time series (conditional variances, jump occurrences and jump sizes) that are unobservable from the price evolution, it is not possible to estimate it with a simple Maximum Likelihood Estimation method as is the case for many simpler models. More complex computational methods have to be used instead, such as the Efficient Method of Moments (EMM), Markov Chain Monte Carlo (MCMC) or Particle Filters (PF). In our case, a Bayesian estimation method with an MCMC algorithm is used for the estimation of the model on the EURUSD time series, with the construction of the algorithm based on Ficura and Witzany (2016).

The algorithm uses 20 000 Markov chain iterations with the first 5 000 of them being discarded (as the algorithm might have not converged yet to the target posterior distribution) and the last 15 000 being used to estimate the model parameters based on their posterior means and the Bayesian standard error based on posterior standard deviations.

The estimated parameter values are shown in Table 1.

Table 1 – Parameters of the SVJD model estimated on the EURUSD time series fro 1.11.1999 to 12.6.2015

	mui	muiJ	sigmaJ	alpha	beta	gamma	lambdaLT	betaJ	gammaJ
Parameter	5.62E-05	0.0008	0.0083	-0.0506	0.9951	0.0668	0.0277	0.4403	0.0291
Std.error	6.74E-05	0.0023	0.0019	0.0189	0.0018	0.0064	0.0209	0.2652	0.0217

As can be seen from the parameter values in the Table 1, the parameters do, more or less, correspond to our expectations, especially with regards to the highly persistent stochastic variance (the slope coefficient beta in the log-SV model is equal to 0.9951), and the relatively large (0.83%), infrequent (2.77%) jumps occurring in the price process. In constrast to our expectations, the estimated process does not contain significant jump clustering (i.e. the betaJ parameter is not close to 1 and the gammaJ parameter is statistically insignificant).

As the jump clustering is a potentially important effect of the financial time series behaviour, we will add this effect to the simulated time series (by setting betaJ to 0.99 and gammaJ to 0.02) in order to evaluate the trading model performances in the presence of this additional noise effect. Additionally, we slightly increase the mean jump intensity (lambdLT) to 0.05 and the absolute jump size (sigmaJ) to 0.01, so that the jumps play a more important role in the analysed time series, adding discontinuos noise component to their behaviour and complicating thus the identification of the added non-linear patterns by the tested data mining methods.

Finally, we set the unconditional mean parameter (mui), and the mean jump size (muiJ), equal to zero, so that the simulated process is a martingale, as long as the conditional mean patterns are not added to its behaviour. We perform this modification of the parameters so that the tested data mining models focus solely on the identification of the non-linear conditional mean effect that we add to the simulated time series behaviour and not on any other possible effects in the time series.

The parameters used for the simulation are thus as shown in Table 2:

Table 2 – Parameters used in the SVJD proces simulation

	mui	muiJ	sigmaJ	stdLT	beta	gamma	lambdaLT	betaJ	gammaJ
Parameter	0	0	0.01	0.01	0.99	0.1	0.05	0.97	0.02

Finally, the processes to be simulated are extended with 10 non-linear patterns governing the evolution of the conditional mean. The patterns are as follows:

1. Deterministic cyclical trend (modelled as a time-varying drift rate) governed by a sinus function with a period of 20 days (i.e. approximately one month). The trend strength is at its peak equal to 0.5 of the unconditional standard deviation of the simulated process.
2. Randomly alternating phases of uptrend and downtrend (characterized by a drift rate of either 0.25 or -0.25 unconditional standard deviations) governed by a Poisson arrival random variable with probability of trend-change of 5% per day.
3. Random 5-day surges or falls of the market, governed by a Poisson arrival variable with probability of 3% per day. The surge starts with a 0.25 times unconditional standard deviation drift in the first day and continues with 0.5, 0.75, 0.5 and 0.25 standard deviation drifts in the following 2th to 5th day.
4. Continuation of the price movement after a jump occurs, equal to 0.5 of the jump size in the following day after the jump occurrence

5. Correction after weekly surges or falls. If the price happens to move more that 4 positive or negative standard deviations over the preceeding 5 day period, there is a correction (i.e. change of the conditional drift rate) in the following day, equal to 0.5 of the unconditional standard deviation of the returns
6. Whenever is the price above its 10-day simple moving average, a positive drift equal to 0.25 of the unconditional standard deviation is added to the return equation, similarly, when the price is below its 10-day simple moving average, a -0.25 of the unconditional standard deviation drift is added to the return equation
7. When the price is above its 10-day simple moving average, the drift is equal to 0.25 unconditional standard deviations, unless the price is higher than 2 standard deviations above the moving average in which case the drift reverses to -0.25 standard deviations. Analogical is the situation in the opposite case when the price is below its 10-day SMA.
8. There are 3 frequencies of cyclical trends in the price behaviour, governed by sinus curves with periods equal to 5 days, 22 days and 66 days. At their maximum each of the sinus curves adds +0.25 (or -0.25) of the unconditional standard deviation to the overall price drift.
9. Randomly switching phases of positive price autocorrelation and negative price autocorrelation occur in the time series. The switches are governed by a Poison arrival process with a probability of a switch equal to 5% per day. The autocorrelation alternates in this kind between the values of +0.5 and -0.5.
10. The price reacts to the predicted dynamics by the technical analysis based Candlestick Patterns (which were found to possess predictive power even for the real financial time series, based on Caginalp and Laurent, 1998). Specifically, the patterns Bullish Engulfing, Bearish Engulfing, Morning Star, Evening Star, Three White Soldiers and Three Black Crows are added to the price behaviour. The patterns re defined via a series of inequalities and they trigger, in the following two days after their occurrence, either a positive or a negative drift (depending on whether they are bullish or bearish) equal to 0.5 of the unconditional standard deviation per day.

For the purpose of the simulation study, 10 financial time series of length 4000 periods (days) were simulated, each of them containing one of the aforementioned non-linear dependency effects. The sample size was chosen to be 4 000 trading days in order to match the size of the empirical samples of currency exchange rates, that will be used in the next section.

In order to realistically evaluate the performance of the applied data-mining models on the simulated time series, we divided the series into a training sample (first 2 000 days), validation sample (next 1 000 days) and testing sample (last 1 000 days). The training sample is used as the "reservoir" for the similarity search in the case of the *k*-NN method and as a sample for the estimation of the model parameters in the case of the Ridge Regression and the Feed Forward Neural Networks. The validation sample is then used to select the best set of meta-parameters of the model. The optimized meta-parameters for the three tested methods are as follows:

- For the ***k*-Nearest Neighbour** the number of nearest neighbours is the parameter to be optimized. We will test alternatively the values of *k={10,20,30,40,50,75,100,150,200,250,300,400,500}*. Additionally, in the version of the *k*-NN algorithm utilizing the Principal Component Analysis for dimensionality reduction, the number of principal components employed is a second meta-parameter optimized in the validation sample. In this case the first *pc* principal components are used instead of the original explanatory variable set, with *pc={2,3,4,5,6}*. As for the other parameters, we do not optimize the type of distance measure used in the *k*-NN algorithm, but instead apply the

algorithm in 4 different variants corresponding to the 4 possible distance measures (Euclidian, Manhattan, Mahalanobis and Maximum distance), in order to evaluate the performance of each of them separately.

- For the **Ridge Regression model**, the size of the penalization term *r* will be optimized in the validation sample, with *r={1,2,3,5,10,15,20,30,50,75,100,150,200,300,400,500,750,1000,2000,3000,5000,7500,10000,20000,30000,50000,75000,100000}*. Additionally, in the case when the Ridge Regression is combined with PCA, the number of principal components employed will be optimized as well.
- Finally, in the case of the **Feed-Forward Neural Networks**, we will optimize the step size in the gradient descent learning algorithm *q* as well as the number of iterations *N*, as both of these parameters are crucial in order to avoid overfitting. Additionally, we optimize the number of neurons *n* in each layer of the neural network. The tested values are *q={0.02,0.03,0.05,0.1}*, *N={3000,5000,7000,10000}* and *n={5,10,15,20}*. We do not optimize the number of hidden layers in the neural network, but instead test alternatively a neural network with 1, 2 and 3 hidden layers and compare their results afterwards. In the case when the FFNN model is combined with PCA for dimensionality reduction, the number of principal components will be optimized as well.

Table 3 provides a summarization of the alternative values used for the model optimization in the validation sample period.

Table 3 – Alternative parameter values used for model optimization in the validation sample pleriod

	No PCA	**With PCA**
K-Nearest Neighbour	k={10,20,30,40,50,75,100,150,200,250, 300,400,500}	k={10,20,30,40,50,75,100,150,200,250, 300,400,500} pc={2,3,4,5,6}
Ridge Regression	r={1,2,3,5,10,15,20,30,50,75,100,150,200, 300,400,500,750,1000,2000,3000,5000, 7500,10000,20000,30000,50000,75000, 100000}	r={1,2,3,5,10,15,20,30,50,75,100,150,200, 300,400,500,750,1000,2000,3000,5000, 7500,10000,20000,30000,50000,75000, 100000} pc={2,3,4,5,6}
Feed-Forward Neural Networks	q={0.02,0.03,0.05,0.1} N={3000,5000,7000,10000} n={5,10,15,20}	q={0.02,0.03,0.05,0.1} N={3000,5000,7000,10000} n={5,10,15,20} pc={2,3,4,5,6}

After the optimal values of meta-parameters are selected based on the performance of the different model settings in the validation sample period, the models with optimal settings will be applied to the testing sample, in order to realistically evaluate their out-of-sample performance.

Overall 9 basic specifications of the 3 model types will be employed:

1. ***K*-Nearest Neighbour** classifier will be tested with alternatively the Euclidian, Manhattan, Mahalabonis and Maximum distance used as the distance measure.
2. **Ridge Regression** model will be applied in 2 variants, i.e. as a Linear Ridge Regression and a Quadratic Ridge Regression, with the quadratic model containing additionally all of the predictors squared, but not the cross-multiples of the predictors.

3. **Feed-Forward Neural Network** will be tested alternatively with 1, 2 and 3 hidden layers, all with the same number of neurons (optimized) and a logistic activation function in the hidden layer and an identity function in the output layer.

With all of the models tested alternatively with the chosen set of predictors as well as with PCA employed for dimensionality reduction and an optimized number of principal components used instead of the original predictor set. I.e. as all of the models will be tested with and without the PCA dimensionality reduction, there will be altogether 18 model specifications tested.

Regarding the choice of predictors, later in the text we will propose a possible selection method in order to select an optimal set of several predictors out of a larger set of potential ones. Nevertheless, for the first, illustrative part of this study, we keep the testing simple and use as predictors simply the price movements over the last 5 trading days (alternatively, returns over the last 5 days will be used instead of the price changes).

The target variable will for all of the models be the price change in the following 1 day, in the case when last 5 daily price movements are used as predictors, or the return in the following 1 day, in the case when the last 5 daily returns are used as predictors.

Finally, it is necessary to determine what measure will be used to evaluate the performance of the tested models. For this purpose, either statistical measures, measuring the accuracy of the forecasts, such as sum of squared residuals or the R-Squared, could be used, or alternatively, measures of the trading profitability of the models, such as the total cumulative profit, total cumulative return or the Sharpe ratio could be used.

In our study the focus would be placed on the trading performance and the critical measure to evaluate the models would be the Drawdown ratio, calculated as the ratio between yearly profit and maximum drawdown (i.e. maximum drop of the cumulative profit during the whole model testing period). The Drawdown ratio was chosen instead of the other measures, due to its connection to the expected annual return of the trading system in the case when leverage is utilized by the trader, which is a common practice among speculators trading with currencies. As many retail brokers allow the investor to invest in 100/1 or even 1000/1 leverage, the return of a trading system depends primarily on the amount of risk the investor is willing to accept. A common approach among traders is to calculate the capital necessary for the trading of a given system based on the maximum historical drawdown. A drawdown ratio is thus directly linked to the expected return of the trading system. Specifically, if the investor decides to trade the system with a capital equal to the maximum historical drawdown, the drawdown ratio would represent the expected annual return on his capital. If the investor is more risk averse and decides to trade the system with a capital equal to a double of the maximum historical drawdown, a one half of the drawdown ratio would represent his expected annual return, etc.

As already mentioned, for the first round of testing, no variable selection will be performed but instead the models will be tested with the last 5 day price movements as predictors, or alternatively the last 5 day returns.

Figure 3, Figure 4, Table 4 and Table 5 show the results for the case when the last 5 day price movements are used as predictors to predict the price movement in the following day. Based on the sign of the prediction the systems will then enter either a long or a short position for the given day.

Figure 3 shows the drawdown ratios for the first 5 simulated time series (i.e. corresponding to the non-linear patterns 1 to 5 as described above).

Figure 3 – Drawdown ratios of the 18 tested model specifications applied to the simulated time series with non-linear patterns 1 to 5 (models applied to past price movements)

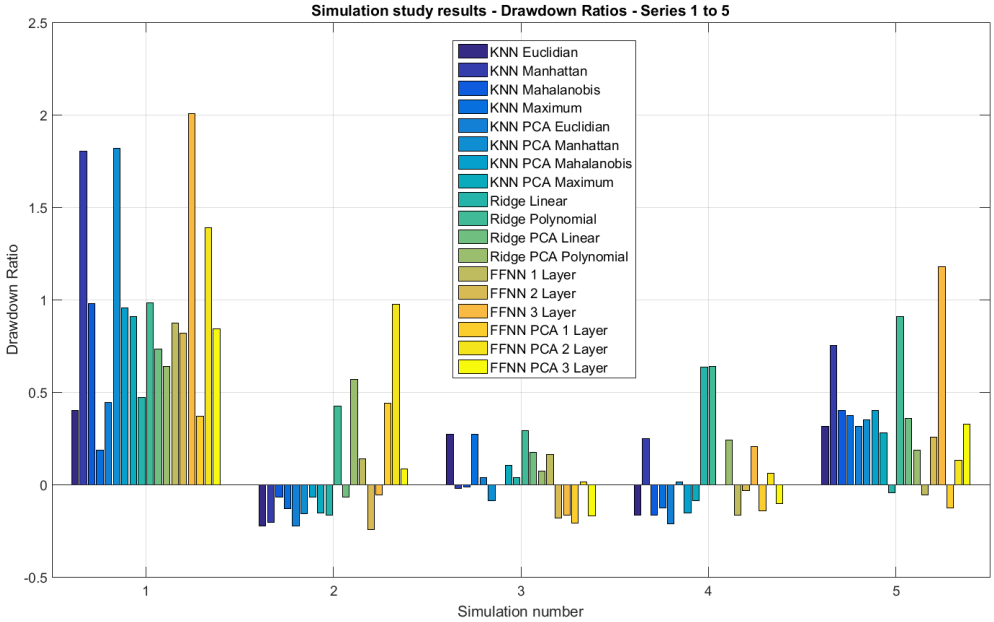
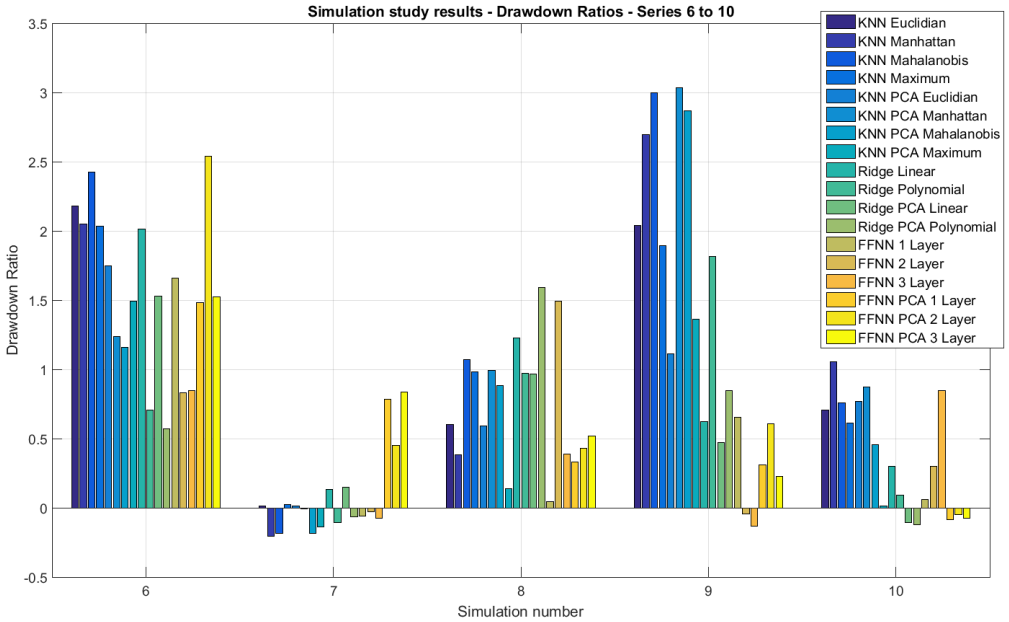


Figure 4 corresponds to the simulated time series 6 to 10.

Figure 4 – Drawdown ratios of the 18 tested model specifications applied to the simulated time series with non-linear patterns 6 to 10 (models applied to past price movements)



We can see from Figure 3 and Figure 4, that patterns 1 (cyclical sinusoid trend) and pattern 6 (moving average rule) were quite easily identified by all of the tested models, reaching high theoretical profits on them. Similar, although with more mixed results, is the situation for the patterns 5 (correction after weekly surges or drops) and 8 (triple sinusoid trend with different frequencies). Additionally, most models coped relatively well with pattern 9 (random switching

between positive and negative autocorrelation), although in the case of this pattern, the neural networks exhibited (surprisingly) significantly lower performance than the *k*-NN method and the ridge regression. Similar performance distribution was further found for the case of pattern 10 (candlestick patterns), which got relatively well identified by the *k*-NN method and the three hidden layer neural network (without PCA), but caused a loss for the ridge regression model and the PCA infused neural networks.

The remaining patterns proved to be problematic for most of the tested models. Pattern 3 (randomly arriving 5-day surges) got to a certain degree identified by some of the *k*-NN methods and by the ridge regression, but caused a loss for most of the neural networks with the exception of the most simple one, i.e. FFNN with one hidden layer without PCA. Opposite was then the situation for Pattern 2 (randomly switching uptrend and downtrend) which got identified by some of the neural networks (especially the PCA based ones) and by the quadratic ridge regression, but caused a significant loss to all of the *k*-NN methods.

The remaining two pattern were successfully identified only by very narrow set of methods. Surprisingly, one of these was pattern 4 (continuation of jump movements in the following days) which got successfully identified mainly by the ridge regression methods. Finally, the pattern 7 (moving average rule extended with overbought/oversold behaviour) got successfully identified only by the PCA extended neural networks, with all other methods achieving poor results for this pattern.

Important conclusion of the first results of the simulation study therefore is, that for different patterns in the underlying time series different data mining methods seem to be suitable, with no universal method that would reach consistently profitable results for all of the tested patterns.

As for the methods that failed at some specific patterns, it is worth noting that the power of these methods was most probably significantly affected by the choice of the predictor variables as the price movements over the last 5 days do possess only limited information about the long-term behaviour about the time series which might be crucial for an easy identification of some of the patterns. A case of this problem would definitely be the pattern 2 (randomly changing uptrend and downtrend) which apparently could not be identified by the *k*-NN method based on the given set of predictor variables but a different set might have made it easily identifiable. It is surprising, however, that a different method, namely the PCA extended Feed-Forward Neural Network (and to a certain degree even the quadratic ridge regression) succeeded to identify the trend changes relatively well even form the limited information in the available predictor variables (last 5 price movements).

Although no method proved to be universally applicable for the modelling of all of the tested patterns, some of them achieved significantly more stable performance than others. To evaluate which method is the most robust and universal one, the aggregated results (for all of the patterns) are shown in Table 4. The statistics in the table show the average, highest and lowest Drawdown ratio for each of the methods on all of the tested patterns, the average rank of the method compared to the other methods (to evaluate its average relative performance compared to others), the percent of patterns for which the method was profitable, the percent for which it was the best method out of the tested ones, and the percent of patterns for which the Drawdown ratio of the method was higher than the median Drawdown ratio of all of the methods. The red-green colour spectrum illustrates the performance of the given method compared to others.

The aggregated results of the simulation study are shown in Table 4.

Table 4 – Aggregated performance of the tested models in the simulation study (models applied to past price movements)

	Average DD_Ratio	Highest DD_Ratio	Lowest DD_Ratio	Average rank	Percent profitable	Percent of best	Percent >median
KNN_Euclidian	0.61	2.18	-0.22	9.65	0.80	0.00	0.50
KNN_Manhattan	0.86	2.70	-0.21	11.10	0.70	0.10	0.60
KNN_Mahalanobis	0.82	3.00	-0.18	11.45	0.60	0.00	0.70
KNN_Maximum	0.61	2.03	-0.13	11.10	0.80	0.00	0.70
KNN_PCA_Euclidian	0.46	1.75	-0.22	8.25	0.80	0.00	0.50
KNN_PCA_Manhattan	0.81	3.04	-0.16	11.60	0.70	0.10	0.70
KNN_PCA_Mahalanobis	0.63	2.87	-0.18	9.45	0.60	0.00	0.60
KNN_PCA_Maximum	0.39	1.49	-0.15	7.80	0.70	0.00	0.30
Ridge_Linear	0.52	2.01	-0.17	10.00	0.80	0.00	0.50
Ridge_Polynomial	0.67	1.82	-0.11	12.10	0.90	0.20	0.70
Ridge_PCA_Linear	0.42	1.53	-0.11	9.70	0.70	0.00	0.60
Ridge_PCA_Polynomial	0.45	1.59	-0.12	9.10	0.80	0.10	0.40
FFNN_1_Layer	0.33	1.66	-0.17	7.70	0.70	0.00	0.40
FFNN_2_Layer	0.32	1.49	-0.24	6.80	0.50	0.00	0.30
FFNN_3_Layer	0.50	2.00	-0.16	9.80	0.60	0.20	0.50
FFNN_PCA_1_Layer	0.31	1.48	-0.21	6.00	0.60	0.00	0.20
FFNN_PCA_2_Layer	0.65	2.54	-0.05	11.00	0.90	0.20	0.50
FFNN_PCA_3_Layer	0.40	1.52	-0.17	8.40	0.70	0.10	0.30

From Table 4 we can see that the *k*-NN methods provide relatively robust results, having one of the highest average Drawdown ratios and average ranks. Surprisingly, the most commonly used Euclidian distance *k*-NN, seems to be relatively worse than the versions with other distance functions, based on the relative performance measures, its absolute profitability (percent profitable) is however relatively high.

A further surprise might be that the polynomial (i.e. quadratic) ridge regression is apparently the most universal method, with the highest average rank of all of the methods, one of the two highest percentages of profitable patterns (90%, achieving a loss only on a single pattern), as well as one of the highest percentages of above median results.

The neural network models did, on the other hand, achieve relatively worse results than the other methods, with the possible exception of FFNN PCA with 2 hidden layers which has one of the two highest percentages of profitable patterns (90%).

As for the utilization of PCA dimensionality reduction, it seems to slightly decrease the performance of the *k*-NN algorithm (with the exception of the one with the Manhattan distance, for which it increased the performance) and to decrease the performance of the ridge regression, while, on the other hand, it seems to increase the performance of the neural network models with more than one hidden layer.

Another way of aggregating the results of the models, in order to evaluate which one is the most robust one, is to calculate their profitability in the case when they are applied for the simultaneous trading of all of the 10 simulated time series. The results (total profits, maximum drawdowns, drawdown ratios and the relative rank of the models based on their drawdown ratios) for this type of aggregation are shown in Table 5.

Table 5 – Aggregated profitability of the tested models in the simulation study for the case when each model is used for a simultaneous trading on all of the simulated time series (models applied to past price movements)

Model	Total profit	Max Drawdown	MDD Ratio	MDD Ratio p.a.	Relative Rank
KNN_Euclidian	0.904	-0.723	1.251	0.315	6.000
KNN_Manhattan	1.059	-0.712	1.488	0.375	9.000
KNN_Mahalanobis	1.308	-0.900	1.453	0.366	7.000
KNN_Maximum	0.995	-0.819	1.214	0.306	5.000
KNN_PCA_Euclidian	0.626	-0.798	0.785	0.198	3.000
KNN_PCA_Manhattan	1.117	-0.596	1.874	0.472	11.000
KNN_PCA_Mahalanobis	1.159	-0.784	1.479	0.373	8.000
KNN_PCA_Maximum	0.194	-0.979	0.198	0.050	2.000
Ridge_Linear	1.312	-0.774	1.694	0.427	10.000
Ridge_Polynomial	2.822	-0.569	4.958	1.249	17.000
Ridge_PCA_Linear	1.486	-0.645	2.304	0.581	13.000
Ridge_PCA_Polynomial	2.304	-0.507	4.547	1.146	16.000
FFNN_1_Layer	0.824	-0.692	1.192	0.300	4.000
FFNN_2_Layer	-0.052	-0.751	-0.069	-0.017	1.000
FFNN_3_Layer	1.513	-0.770	1.965	0.495	12.000
FFNN_PCA_1_Layer	1.627	-0.691	2.353	0.593	14.000
FFNN_PCA_2_Layer	3.504	-0.431	8.134	2.050	18.000
FFNN_PCA_3_Layer	2.456	-0.845	2.908	0.733	15.000

Surprisingly, the alternative method for the aggregation of the performance of the tested models gives slightly different results than the approach shown in Table 4. While the Polynomial Ridge Regression is still one of the best methods, even according to the results in Table 5 (it has the second highest relative rank based on the drawdown ratio in the simultaneous trading of all of the series), the *k*-NN based methods do in this case exhibit on average a worse performance than some of the Feed Forward Neural Network based models, specifically the ones utilizing the PCA for dimensionality reduction. Indeed, the FFNN with 2 hidden layers and PCA achieved in Table 5 the highest performance of all of the models, with the two other PCA utilizing feed forward neural networks, while scoring worse than the polynomial ridge regression (with or without PCA), still surpassed all of the *k*-NN based models.

Figure 5, Figure 6, Table 6 and Table 7 show the results for the case when the last 5 day returns are used as predictors to predict the return in the following day. Based on the sign of the prediction the systems will then enter either a long or a short position for the given day.

We can see from Figure 5 and Figure 6, that especially for the first 5 patterns, the models applied to the past simulated returns achieve worse performance than what was the case for the models applied to the past price movements. Nevertheless, the general tendencies remain similar to a certain degree. Specifically, for the patterns 1, 6, 8 and 9, most of the models achieved relatively high profitability, although it is apparent that especially for the pattern 1, some of the neural networks failed in the case when past returns were used as predictors, while in the case when movements were used as predictors, their performance was high. Performance at pattern 10 was worse for the return-based models than for the movements-based, while the performance at pattern 7 was better for most of the models. As for the patterns 2,3,4 and 5, most of the return based models achieved bad results in these cases, with few exceptions, such as the PCA extended two layer neural network for pattern 3 or the non-PCA *k*-NN methods for pattern 5.

Figure 5 – Drawdown ratios of the 18 tested model specifications applied to the simulated time series with non-linear patterns 1 to 5 (models applied to past returns)

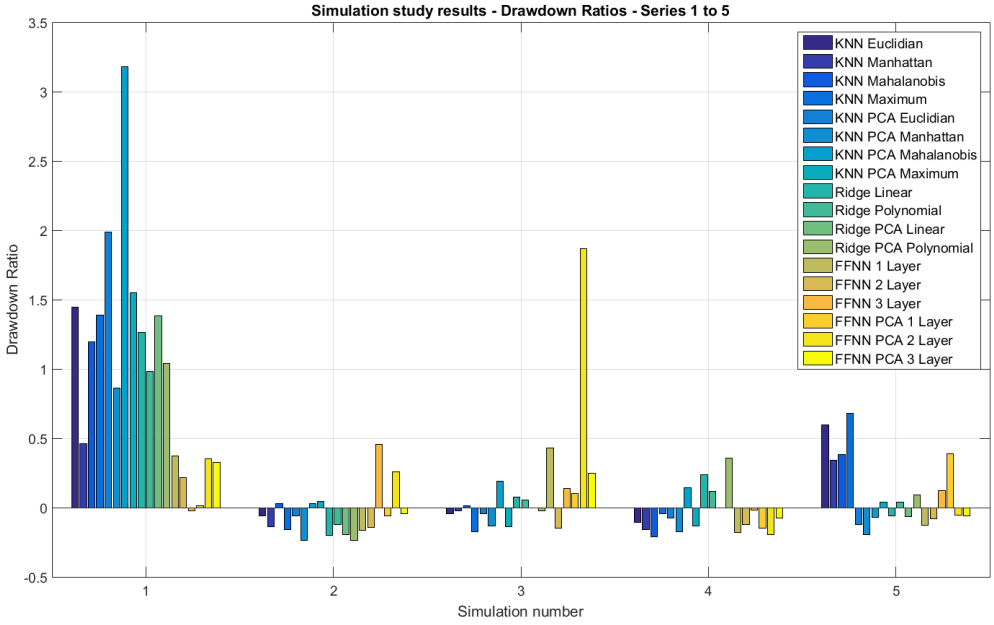


Figure 6 – Drawdown ratios of the 18 tested model specifications applied to the simulated time series with non-linear patterns 6 to 10 (models applied to past returns)

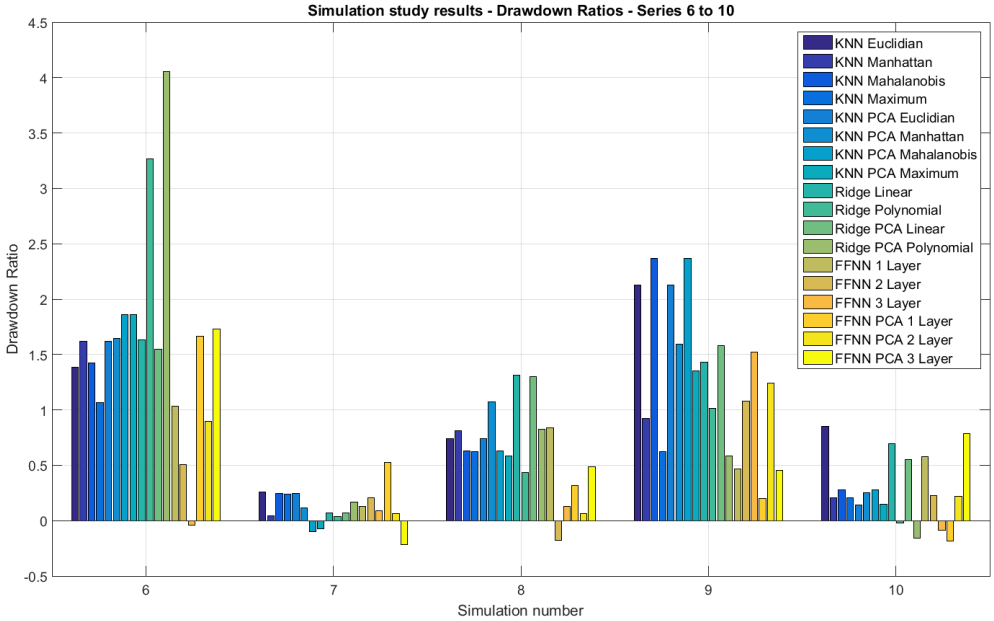


Table 6 shows the aggregated returns for the return-based models in the simulation study.

Table 6 – Aggregated performance of the tested models in the simulation study (models applied to past returns)

	Average DD_Ratio	Highest DD_Ratio	Lowest DD_Ratio	Average rank	Percent profitable	Percent of best	Percent >median
KNN_Euclidian	0,72	2,12	-0,11	12,60	0,70	0,10	0,70
KNN_Manhattan	0,41	1,62	-0,16	8,30	0,70	0,00	0,20
KNN_Mahalanobis	0,63	2,37	-0,21	11,30	0,90	0,10	0,70
KNN_Maximum	0,44	1,39	-0,18	9,00	0,70	0,10	0,40
KNN_PCA_Euclidian	0,66	2,12	-0,12	10,50	0,60	0,00	0,70
KNN_PCA_Manhattan	0,48	1,64	-0,24	8,10	0,60	0,00	0,50
KNN_PCA_Mahalanobis	0,85	3,18	-0,10	12,60	0,80	0,20	0,80
KNN_PCA_Maximum	0,52	1,86	-0,14	9,30	0,70	0,00	0,50
Ridge_Linear	0,65	1,63	-0,20	11,55	0,80	0,10	0,70
Ridge_Polynomial	0,58	3,27	-0,12	9,20	0,80	0,00	0,40
Ridge_PCA_Linear	0,62	1,58	-0,20	10,55	0,60	0,00	0,50
Ridge_PCA_Polynomial	0,67	4,05	-0,24	9,90	0,70	0,20	0,60
FFNN_1_Layer	0,34	1,03	-0,18	8,10	0,70	0,00	0,40
FFNN_2_Layer	0,16	1,08	-0,18	5,80	0,50	0,00	0,20
FFNN_3_Layer	0,23	1,52	-0,09	8,70	0,60	0,10	0,50
FFNN_PCA_1_Layer	0,28	1,67	-0,19	8,40	0,70	0,10	0,50
FFNN_PCA_2_Layer	0,47	1,87	-0,19	8,00	0,80	0,10	0,20
FFNN_PCA_3_Layer	0,36	1,73	-0,22	9,10	0,60	0,00	0,50

We can see from Table 6 that the aggregated results for the models applied to the past returns differ to a certain degree from the aggregated results in the case when the models were applied to the past price movements (Table 4). Specifically, the *k*-NN method with the Mahalanobis distance seem to be the most robust method in this case, followed by the *k*-NN method with the Euclidian distance function. Ridge regression (linear as well as quadratic) achieved relatively robust results even in this case, with the difference that for the return-based models the linear ridge regression performed better than the quadratic ridge regression, while for the movement-based models the quadratic ridge regression performed better. The neural network based models exhibit again a relatively worse performance than the *k*-NN based models and the ridge regression based models. Similarly, the impact of the PCA on model performance is again rather inconclusive, increasing the profitability of some of the models, while decreasing it for others.

Table 7 shows the results for the second method of aggregation in which the results are reported for the case when each of the model is applied for the simultaneous trading on all of the simulated time series. We can see that while in the case of movements-based models, the two approaches to results aggregation achieved slightly conflicting results (with the second one speaking more for the neural network models), in the case of the return-based models the results in Table 6 and Table 7 present more or less similar picture, with the *k*-NN based models dominating the other approaches, i.e. the Ridge Regression and the Feed Forward Neural Networks. Some of the neural networks (3 layer FFNN without PCA and 2 layer FFNN with PCA) surpassed the Ridge Regression models, nevertheless, the results of some of the other neural network models were poor, while the performance of all of the Ridge Regression models is rather stable. The mixed performance of the tested neural network architectures in the performed studies may indicate that the analysed time

series may be too short to fully utilize the potential of these models, causing them to engage in overfitting. Ridge regression results are stable, on the other hand, due to its simplicity.

Table 7 – Aggregated profitability of the tested models in the simulation study for the case when each model is used for a simultaneous trading on all of the simulated time series (models applied to past returns)

Model	Total profit	Max Drawdown	MDD Ratio	MDD Ratio p.a.	Relative Rank
KNN_Euclidian	2,468	-0,524	4,713	1,188	18,000
KNN_Manhattan	1,056	-0,730	1,447	0,365	10,000
KNN_Mahalanobis	2,099	-0,715	2,934	0,739	16,000
KNN_Maximum	1,573	-0,428	3,678	0,927	17,000
KNN_PCA_Euclidian	1,119	-0,693	1,616	0,407	12,000
KNN_PCA_Manhattan	-1,268	-1,927	-0,658	-0,166	1,000
KNN_PCA_Mahalanobis	1,389	-0,533	2,607	0,657	15,000
KNN_PCA_Maximum	0,973	-0,792	1,229	0,310	9,000
Ridge_Linear	0,926	-1,044	0,887	0,224	7,000
Ridge_Polynomial	0,969	-1,075	0,902	0,227	8,000
Ridge_PCA_Linear	0,742	-0,974	0,762	0,192	6,000
Ridge_PCA_Polynomial	0,503	-0,900	0,559	0,141	5,000
FFNN_1_Layer	-0,293	-1,598	-0,183	-0,046	3,000
FFNN_2_Layer	-0,828	-2,133	-0,388	-0,098	2,000
FFNN_3_Layer	1,493	-0,772	1,934	0,487	14,000
FFNN_PCA_1_Layer	1,264	-0,785	1,610	0,406	11,000
FFNN_PCA_2_Layer	1,194	-0,678	1,760	0,444	13,000
FFNN_PCA_3_Layer	0,327	-0,931	0,351	0,089	4,000

Empirical application to foreign exchange rates

In the empirical study of the performance of the selected data mining models for the foreign exchange rate forecasting, the methods will be applied to the time series of 10 major foreign exchange rate time series in the period ranging from 01.11.1999 to 12.6.2015, containing altogether 4 062 daily observations (for each of the series). The analysed time series correspond to the currencies 1. AUDUSD, 2. EURGBP, 3. EURJPY, 4. EURUSD, 5. GBPJPY, 6. GBPUSD, 7. NZDUSD, 8. USDCAD, 9. USDCHF and 10. USDJPY, with the data being provided by ForexHistoryDatabase.com.

Similarly, as in the simulation study, the whole data sample will be divided into a development sample (first 2 000 days), validation sample (day 2 001 to 3000) and testing sample (day 3 001 to 4 062), with the development sample used for the model parameter estimation (and as a "reservoir" for the similarity search in the *k*-NN method), the validation sample will be used for meta-parameter optimization and the testing sample will be used for model performance evaluation.

The main criterion for the assessment of the model predictive power will be the Drawdown ratio defined as the ratio between the average annual return of the system and the maximum Drawdown (i.e. the maximum decrease of the invested equity, calculated, for the purposes of model comparison, over the testing sample). The Drawdown ratio can be used to calculate the expected return on the models for investors with different levels of risk aversion. A common rule, used by speculators to determine an acceptable level of leverage, is to demand that the invested capital covers at least the maximum drawdown of the system to be traded. For an investor, whose capital covers exactly the maximum Drawdown, does the Drawdown ratio represent the expected annual

return on his capital. For a more risk averse investor, with a lower leverage, calculated so that his capital covers the double of the maximum Drawdown, does the Drawdown ratio represent the double of his expected annual return (i.e. his return p.a. will be one half of the Drawdown ratio).

In order to simulate a real-life setting in the back-testing procedure, commissions and spreads need to be accounted for, in order to realistically asses the real-world profitability of the tested systems. To keep the issue simple, we further assume a 1 pip spread for all of the analysed currencies (i.e. equal to 10 USD for the EUR/USD currency exchange rate).

As in the simulation study, the proposed models will first be tested with a default set of predictors, without any selection procedure. The default predictors will for this purpose be again either the last 5-day price movements, or the last 5-day returns. In the case when the last 5-day price movements are used, the following 1-day price movement will be the target variable for the model training, while in the case when the last 5-day returns are used, the following 1-day return will be the target.

Figure 7, Figure 8, Table 8, Table 9 and **Chyba! Chybný odkaz na záložku.** shows the aggregated profitability for the case when all of the tested methods are applied for the simultaneous trading on each one of the analysed currency exchange rate time series. The purpose of the analysis is to evaluate, what currency is potentially most suitable for trading with the tested data mining tools. It is apparent from **Chyba! Chybný odkaz na záložku.** that EURUSD is the most suitable currency and at the same time the only currency on which the simultaneous trading of all of the tested systems at once would have ended with a net profit. As already mentioned this is a surprising result as the EURSUD is the most traded currency exchange rate pair and it should thus, according to the efficient market hypothesis, exhibit the highest levels of information efficiency and therefore the lowest profit potential. The opposite seems to be true, however, from the data that we studied.

Table 10 show the results of the models in the case when the last 5-day price movements are used as predictors and the following one-day price movement as the target variable.

We can see from Figure 7 and Figure 8 that the profitability of the tested data mining methods on the real-world foreign exchange rate time series is not very high, with most of the methods achieving a loss on most of the currencies. Among the currencies on which multiple methods achieved profits, indicating that their behaviour might be inefficient with regards to the efficient market hypothesis, is primarily the EURUSD exchange rate, with lesser profits being achieved by multiple methods also for EURJPY and GBPJPY. These results are surprising as the EURUSD exchange rate is the most traded currency exchange rate and it should thus, theoretically, be the one exhibiting the highest levels of information efficiency, i.e. being the rate that is least predictable. Opposite tendency can, however, be observed from our data, speaking for the alternative hypothesis that high levels of trading activity may paradoxically be creating or exacerbating the market inefficiency.

As for the profitability of the different model classes, the most robust results have been achieved by the Ridge Regression methods, which consistently profited at least on the EURJPY, EURUSD, GBPJPY and NZDUSD exchange rates. The *k*-NN models were mostly unprofitable, with the exception of the Manhattan distance using *k*-NN (without PCA), which achieved profits on AUDUSD, EURGBP, EURJPY, EURUSD, GBPUSD, USDCAD, USDCHF and USDJPY and appears thus to be the most robust individual method out of the tested ones. The Feed-Forward Neural Networks, similarly to the *k*-NN (apart from the Manhattan distance using one), achieved rather poor results, especially in their non-PCA using version. The best among the FFNN models were the two-layer and three-layer networks coupled with PCA, with the three-layer network (which seems to be the most profitable one) achieving relatively high profits on GBPJPY, USDCAD and USDJPY.

Figure 7 – Drawdown ratios of the 18 tested model specifications applied to the empirical time series of 5 currency exchange rates (1. AUDUSD, 2. EURGBP, 3. EURJPY, 4. EURUSD, 5. GBPJPY) (models applied to past price movements)

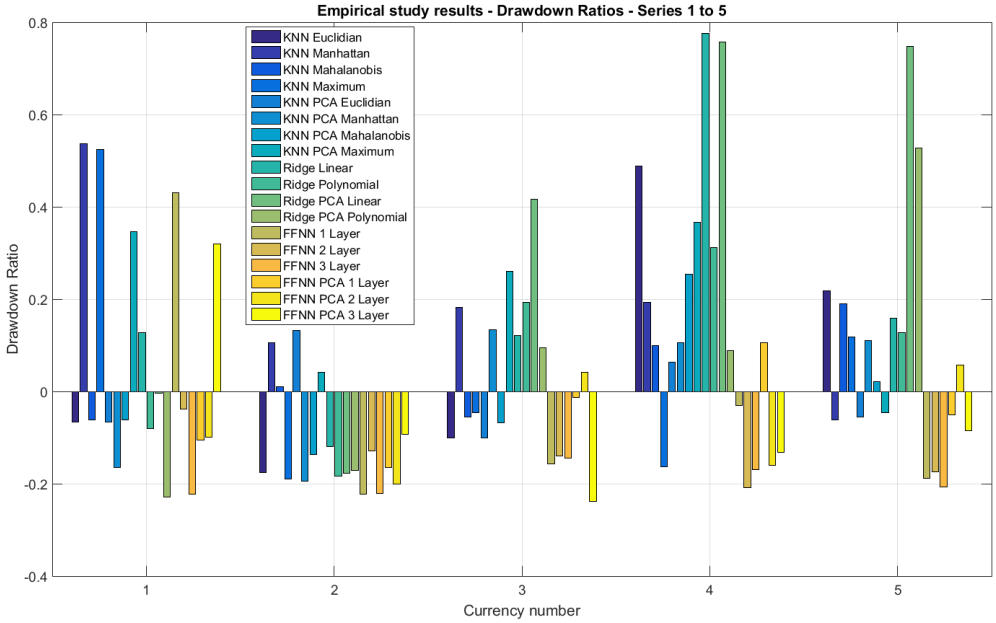


Figure 8 – Drawdown ratios of the 18 tested model specifications applied to the empirical time series of 5 currency exchange rates (6. GBPUSD, 7. NZDUSD, 8. USDCAD, 9. USDCHF, 10. USDJPY) (models applied to past price movements)

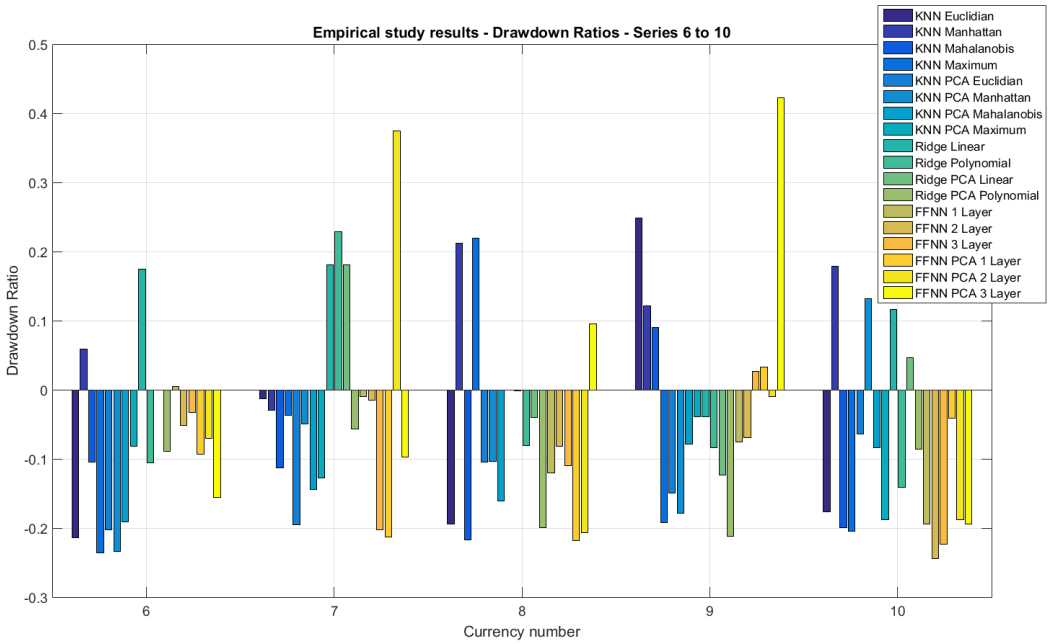


Table 8 shows the aggregated performance of the tested models on all of the currencies. The reported statistics are the average Drawdown Ratio, Highest Drawdown Ratio, Lowest Drawdown Ratio, Average Rank among the tested models, Percent of currencies on which the model was profitable, Percent of currencies for which it was the best model out of the tested ones, and the Percent of currencies in which its Drawdown ratio was above the median for all of the models.

The results in Table 8 confirm clearly that the *k*-NN method with Manhattan distance function, together with the Linear Ridge Regression (with or without PCA) are the most robust methods, with the Manhattan based *k*-NN achieving profit on 8 out of the 10 tested currencies and the non-PCA based Ridge regression on 7 of them. All of the other methods achieved positive profits in less than 50% of the tested currencies, indicating a low performance and robustness. The worst results were achieved by the FFNN models without PCA, which were unprofitable for almost all of the analyzed time series. The neural networks with PCA ended with high profits on some of the currencies, but negative profits on others, indicating low robustness of these method. The profitability of *k*-NN (with the exception of the Manhattan distance based one) was only slightly better than what was the case for the neural network based models.

The results of the analysis are rather surprising, as the theoretically simplest methods achieved the best results (i.e. the simple linear ridge regression and the *k*-NN with Manhattan distance). The more complex models, particularly the neural networks, failed, which could possibly be explained by the relatively low number of observations in the analysed time series, leading to overfitting.

Table 9 shows the aggregated results of the analysis in the case when each of the models is applied for the simultaneous trading on all of the currency exchange rate time series. The results in Table 9 confirm the conclusion of the *k*-NN algorithm being the most promising method for currency trading in the analysed period (measured based on the Drawdown ratio), followed by the Linear Ridge Regression with and without PCA. All of the other methods would have ended with a loss when applied to the simultaneous trading on all of the currencies during the studied period.

Table 8 – Aggregated performance of the tested models in the empirical study (models applied to past price movements)

	Average DD_Ratio	Highest DD_Ratio	Lowest DD_Ratio	Average rank	Percent profitable	Percent of best	Percent >median
KNN_Euclidian	0,00	0,49	-0,21	10,00	0,30	0,00	0,40
KNN_Manhattan	0,15	0,54	-0,06	14,60	0,80	0,20	0,90
KNN_Mahalanobis	-0,04	0,19	-0,22	9,15	0,40	0,00	0,30
KNN_Maximum	-0,02	0,52	-0,24	8,00	0,30	0,10	0,40
KNN_PCA_Euclidian	-0,07	0,13	-0,20	7,70	0,20	0,10	0,20
KNN_PCA_Manhattan	-0,04	0,13	-0,23	8,40	0,40	0,00	0,50
KNN_PCA_Mahalanobis	-0,06	0,25	-0,19	8,35	0,20	0,00	0,30
KNN_PCA_Maximum	0,05	0,37	-0,19	12,00	0,40	0,00	0,70
Ridge_Linear	0,15	0,78	-0,12	14,45	0,70	0,20	1,00
Ridge_Polynomial	0,02	0,31	-0,18	10,70	0,40	0,00	0,60
Ridge_PCA_Linear	0,18	0,76	-0,18	13,55	0,50	0,20	0,80
Ridge_PCA_Polynomial	-0,03	0,53	-0,23	8,10	0,30	0,00	0,40
FFNN_1_Layer	-0,06	0,43	-0,22	7,80	0,20	0,00	0,30
FFNN_2_Layer	-0,12	-0,02	-0,24	7,70	0,00	0,00	0,50
FFNN_3_Layer	-0,15	0,03	-0,22	4,90	0,10	0,00	0,20
FFNN_PCA_1_Layer	-0,08	0,10	-0,22	8,00	0,20	0,00	0,50
FFNN_PCA_2_Layer	-0,05	0,37	-0,21	8,60	0,30	0,10	0,50
FFNN_PCA_3_Layer	-0,02	0,42	-0,24	9,00	0,30	0,10	0,40

Table 9 – Aggregated profitability of the tested models in the empirical study for the case when each model is used for a simultaneous trading on all of the currency exchange rate time series (models applied to past price movements)

Model	Total profit	Max Drawdown	MDD Ratio	MDD Ratio p.a.	Relative Rank
KNN_Euclidian	-0,375	-0,710	-0,528	-0,133	12,000
KNN_Manhattan	0,916	-0,448	2,046	0,516	18,000
KNN_Mahalanobis	-0,333	-0,746	-0,446	-0,112	13,000
KNN_Maximum	-0,941	-1,434	-0,656	-0,165	9,000
KNN_PCA_Euclidian	-1,279	-1,373	-0,932	-0,235	1,000
KNN_PCA_Manhattan	-0,758	-1,066	-0,711	-0,179	8,000
KNN_PCA_Mahalanobis	-0,964	-1,265	-0,762	-0,192	6,000
KNN_PCA_Maximum	-0,107	-1,253	-0,085	-0,021	14,000
Ridge_Linear	1,060	-0,729	1,454	0,366	16,000
Ridge_Polynomial	-0,044	-0,891	-0,049	-0,012	15,000
Ridge_PCA_Linear	1,157	-0,772	1,499	0,378	17,000
Ridge_PCA_Polynomial	-0,606	-1,109	-0,546	-0,138	11,000
FFNN_1_Layer	-1,389	-1,790	-0,776	-0,196	4,000
FFNN_2_Layer	-1,899	-2,080	-0,913	-0,230	3,000
FFNN_3_Layer	-2,997	-3,281	-0,913	-0,230	2,000
FFNN_PCA_1_Layer	-1,059	-1,365	-0,776	-0,195	5,000
FFNN_PCA_2_Layer	-0,748	-1,029	-0,727	-0,183	7,000
FFNN_PCA_3_Layer	-1,378	-2,280	-0,604	-0,152	10,000

Chyba! Chybný odkaz na záložku. shows the aggregated profitability for the case when all of the tested methods are applied for the simultaneous trading on each one of the analysed currency exchange rate time series. The purpose of the analysis is to evaluate, what currency is potentially most suitable for trading with the tested data mining tools. It is apparent from **Chyba! Chybný odkaz na záložku.** that EURUSD is the most suitable currency and at the same time the only currency on which the simultaneous trading of all of the tested systems at once would have ended with a net profit. As already mentioned this is a surprising result as the EURSUD is the most traded currency exchange rate pair and it should thus, according to the efficient market hypothesis, exhibit the highest levels of information efficiency and therefore the lowest profit potential. The opposite seems to be true, however, from the data that we studied.

Table 10 – Aggregated profitability of the tested models in the case when all of them are traded simultaneously on each of the currencies (models applied to past price movements)

	Total profit	Max Drawdown	MDD Ratio	MDD Ratio p.a.	Relative Rank
AUDUSD	-0,319	-1,312	-0,243	-0,061	7,000
EURGBP	-1,975	-2,180	-0,906	-0,228	2,000
EURJPY	-0,685	-3,118	-0,220	-0,055	8,000
EURUSD	1,240	-1,422	0,872	0,220	10,000
GBPJPY	-0,162	-3,532	-0,046	-0,012	9,000
GBPUSD	-2,552	-2,800	-0,911	-0,230	1,000
NZDUSD	-1,161	-1,913	-0,607	-0,153	5,000
USDCAD	-1,834	-2,624	-0,699	-0,176	4,000
USDCHF	-1,527	-2,522	-0,606	-0,153	6,000
USDJPY	-2,765	-3,137	-0,881	-0,222	3,000

Figure 9, Figure 10, Table 11, Table 12 and Table 13 show the results of the models in the case when the last 5-day price returns are used as predictors and the following one-day return as the target variable.

We can see from Figure 9 that the tested models achieved certain profits for the AUDUSD, EURJPY and EURUSD currency exchange rate time series. For the other currencies, the performance of most of the models was rather bad.

As for the tested model classes, the *k*-NN based methods achieved more or less consistent profits on AUDUSD, EURJPY and EURUSD exchange rates, while some of the neural networks were profitable on AUDUSD and EURJPY, but generally, their results were inconsistent. Ridge Regression profited on EURJPY and EURUSD, but overall its results are less robust than in the case when it was applied to price movements instead of the price returns.

From Table 11 it is apparent that the results of the return-based models are generally consistent with the results of the movement-based models. The methods achieving profits in 50% or more of the tested currencies are again the *k*-NN method with Manhattan distance (with and without PCA) and the Linear Ridge Regression (with and without PCA). All of the other models achieved loses on the majority of the tested currency exchange rate time series.

Table 12 further confirms the previous conclusion as the four mentioned methods (*k*-NN method with Manhattan distance and Linear Ridge Regression, both either with or without PCA) are the only methods that would end with a profit when applied for the trading on all of the analysed currency exchange rates at once in the testing sample period.

Figure 9 – Drawdown ratios of the 18 tested model specifications applied to the empirical time series of 5 currency exchange rates (1. AUDUSD, 2. EURGBP, 3. EURJPY, 4. EURUSD, 5. GBPJPY) (models applied to past returns)

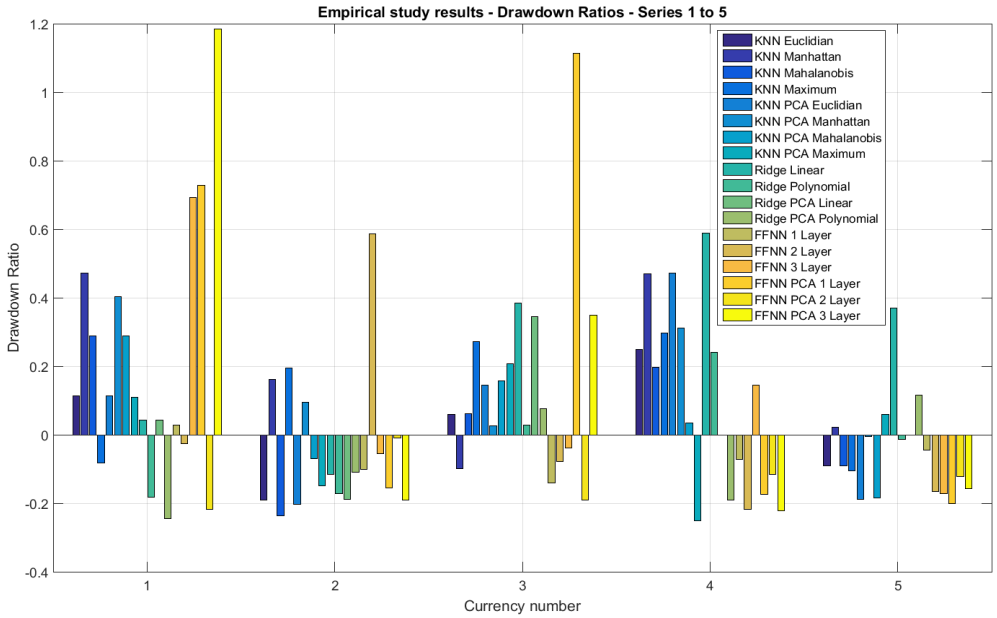


Figure 10 – Drawdown ratios of the 18 tested model specifications applied to the empirical time series of 5 currency exchange rates (6. GBPUSD, 7. NZDUSD, 8. USDCAD, 9. USDCHF, 10. USDJPY) (models applied to past returns)

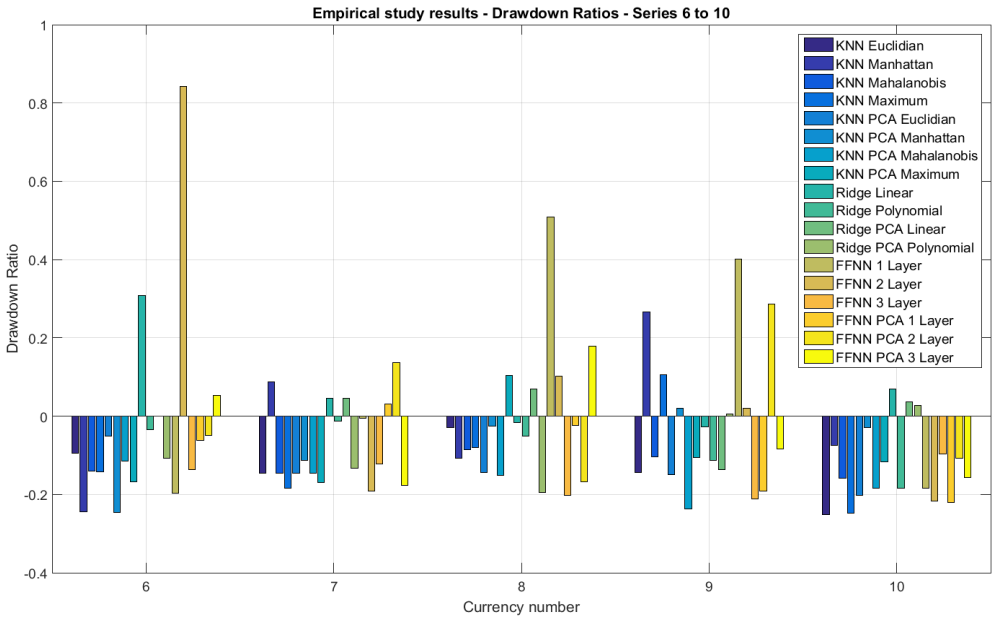


Table 11 – Aggregated performance of the tested models in the empirical study (models applied to past returns)

	Average DD_Ratio	Highest DD_Ratio	Lowest DD_Ratio	Average rank	Percent profitable	Percent of best	Percent >median
KNN_Euclidian	-0,05	0,25	-0,25	7,80	0,30	0,00	0,50
KNN_Manhattan	0,10	0,47	-0,24	12,00	0,60	0,00	0,70
KNN_Mahalanobis	-0,04	0,29	-0,24	7,90	0,30	0,00	0,20
KNN_Maximum	0,00	0,30	-0,25	8,90	0,40	0,00	0,40
KNN_PCA_Euclidian	-0,04	0,47	-0,20	7,60	0,30	0,00	0,40
KNN_PCA_Manhattan	0,04	0,40	-0,25	11,40	0,50	0,00	0,80
KNN_PCA_Mahalanobis	-0,06	0,29	-0,24	7,30	0,30	0,00	0,30
KNN_PCA_Maximum	-0,05	0,21	-0,25	9,00	0,40	0,00	0,40
Ridge_Linear	0,17	0,59	-0,12	14,45	0,70	0,30	0,80
Ridge_Polynomial	-0,05	0,24	-0,18	9,00	0,20	0,00	0,40
Ridge_PCA_Linear	0,02	0,34	-0,19	11,65	0,50	0,00	0,60
Ridge_PCA_Polynomial	-0,08	0,12	-0,24	9,00	0,40	0,00	0,50
FFNN_1_Layer	0,02	0,51	-0,20	9,60	0,30	0,20	0,50
FFNN_2_Layer	0,07	0,84	-0,22	8,70	0,40	0,20	0,40
FFNN_3_Layer	-0,02	0,69	-0,21	8,10	0,20	0,00	0,50
FFNN_PCA_1_Layer	0,08	1,11	-0,22	9,10	0,30	0,10	0,50
FFNN_PCA_2_Layer	-0,06	0,29	-0,22	9,30	0,20	0,10	0,50
FFNN_PCA_3_Layer	0,08	1,18	-0,22	10,20	0,40	0,10	0,60

Table 12 – Aggregated profitability of the tested models in the empirical study for the case when each model is used for a simultaneous trading on all of the currency exchange rate time series (models applied to past returns)

Model	Total profit	Max Drawdown	MDD Ratio	MDD Ratio p.a.	Relative Rank
KNN_Euclidian	-0,814	-1,289	-0,631	-0,159	8,000
KNN_Manhattan	0,242	-0,589	0,410	0,103	17,000
KNN_Mahalanobis	-0,727	-1,641	-0,443	-0,112	12,000
KNN_Maximum	-0,627	-1,307	-0,480	-0,121	10,000
KNN_PCA_Euclidian	-1,090	-1,697	-0,643	-0,162	7,000
KNN_PCA_Manhattan	0,030	-0,953	0,032	0,008	15,000
KNN_PCA_Mahalanobis	-1,380	-1,668	-0,827	-0,208	2,000
KNN_PCA_Maximum	-0,812	-1,078	-0,753	-0,190	6,000
Ridge_Linear	1,173	-0,769	1,525	0,384	18,000
Ridge_Polynomial	-0,645	-1,350	-0,478	-0,120	11,000
Ridge_PCA_Linear	0,063	-0,744	0,084	0,021	16,000
Ridge_PCA_Polynomial	-1,071	-1,376	-0,779	-0,196	3,000
FFNN_1_Layer	-0,565	-0,959	-0,589	-0,149	9,000
FFNN_2_Layer	-0,532	-1,428	-0,372	-0,094	14,000
FFNN_3_Layer	-1,144	-1,506	-0,760	-0,191	5,000
FFNN_PCA_1_Layer	-1,127	-1,466	-0,769	-0,194	4,000
FFNN_PCA_2_Layer	-0,999	-1,080	-0,925	-0,233	1,000
FFNN_PCA_3_Layer	-0,567	-1,338	-0,424	-0,107	13,000

Table 13 shows the profitability of the tested methods in the case when they are simultaneously traded on each one of the analysed currency exchange rate time series. We can see that while in **Chyba! Chybný odkaz na záložku.** shows the aggregated profitability for the case when all of the tested methods are applied for the simultaneous trading on each one of the analysed currency exchange rate time series. The purpose of the analysis is to evaluate, what currency is potentially most suitable for trading with the tested data mining tools. It is apparent from **Chyba! Chybný odkaz na záložku.** that EURUSD is the most suitable currency and at the same time the only currency on which the simultaneous trading of all of the tested systems at once would have ended with a net profit. As already mentioned this is a surprising result as the EURSUD is the most traded currency exchange rate pair and it should thus, according to the efficient market hypothesis, exhibit the highest levels of information efficiency and therefore the lowest profit potential. The opposite seems to be true, however, from the data that we studied.

Table 10, showing the results for the models applied to the past price movements, only the EURUSD exchange rate appeared to be a profitable currency in trading, in the case when the models are applied to past returns, 3 currencies achieve non-negative profits, namely the AUDUSD, EURJPY and EURUSD.

Table 13 – Aggregated profitability of the tested models in the case when all of them are traded simultaneously on each of the currencies (models applied to past returns)

	Total profit	Max Drawdown	MDD Ratio	MDD Ratio p.a.	Relative Rank
AUDUSD	1,493	-0,892	1,673	0,422	10,000
EURGBP	-0,965	-1,253	-0,770	-0,194	2,000
EURJPY	1,891	-2,499	0,757	0,191	9,000
EURUSD	0,172	-1,826	0,094	0,024	8,000

GBPJPY	-3,119	-5,589	-0,558	-0,141	6,000
GBPUSD	-1,607	-2,159	-0,744	-0,188	3,000
NZDUSD	-1,797	-3,055	-0,588	-0,148	5,000
USDCAD	-0,928	-2,322	-0,400	-0,101	7,000
USDCHF	-1,896	-2,803	-0,677	-0,171	4,000
USDJPY	-3,836	-4,049	-0,947	-0,239	1,000

Variable selection

All of the results in the previous sections utilize either the last 5-day price movements or the last 5-day returns as predictors to forecast the future currency exchange rate movements. This is problematic due to two reasons. Firstly, it is probable that even the price movements (returns) before the arbitrarily chosen 5-day period do possibly influence the future price behaviour. The models, as applied in the previous sections, would, however, not be able to utilize the information older than 5 days as they see only 5 days in the past. Secondly, although the applied data mining methods were created to approximate all kinds of non-linear relationships between the predictor variables and the target variable, with limited amount of training data and limited complexity of the models (i.e. limited number of hidden layers and neurons in the neural network models, etc.) they may not be able to approximate all of the relationships that might exist between the past price evolution and the future price behaviour solely from the raw data about the past price movements.

Both of the aforementioned issues can be improved by including technical indicators into the predictor set, in addition or instead of the original predictors. These indicators could easily be set up to utilize information from a significantly longer time-period than the last 5 days, providing to the models the valuable information about the price behaviour in the past, in a compressed manner, leading to a lower risk of overfitting than what would be the case if all of the returns in the relevant period were used.

Additionally, the technical indicators, designed by traders for the purpose of trading signal identification and price prediction, work naturally as feature extractors, extracting form the past price evolution, the features that might be important in order to forecast the future behaviour. Utilization of these indicators in data mining applications for financial time series prediction may thus improve the ability of the applied methods to capture non-linear relationships in the price behaviour and increase their predictive accuracy.

As there is a wide variety of technical indicators that could be used to extend the predictor set, with each of them being parametrizable in many possible ways, it is not straightforward to decide what set of indicators should be used in order to achieve a maximum out-sample predictive performance of the models.

To decide, what set of indicators should be used, in order to extract meaningful features from the past price behaviour that would help the employed data mining method to forecast future price behaviour as accurately as possible, it is necessary to select the right set of indicators with a suitable variable selection procedure.

Many different variable selection procedures can be used for this purpose, ranging from a brute force search, through the well-known forward, backward and stepwise selection procedures to heuristic methods such as genetic algorithms and swarm intelligence methods. In our case, we use a simple forward selection procedure, extended with an early stopping criterion, to avoid overfitting to

the validation sample, which is an issue of serious concern, especially when the sample is short and large number of predictors enter the selection.

In order to apply the forward selection procedure while avoiding any forward looking bias, we divide the full data sample into a Training sample period (day 1 to 2000), Validation sample A (day 2001 to 2500), Validation sample B (2501 to 3000) and a Testing sample (3001 to 4000). We can see that in contrast to the analyses in the previous chapters, the validation sample will now be divided into two validation sub-samples, with one of them being used for the variable selection (i.e. forward selection of the predictors with the goal of maximizing the validation sample Drawdown ratio) and the second one for early stopping (the selection procedure will stop after the Drawdown ratio in the second validation sample stops increasing).

The criterion used for variable selection will be the Drawdown ratio calculated for the case when the given system is traded on all of the analysed time series at once (i.e. all of the 10 currency exchange rates). The final set of predictors will thus be the same for all of the currencies. The motivation for this approach is to mitigate the risk of overfitting to the validation sample via the selection procedure, as selecting a separate variable set for each of the time series could lead to overfitting to the validation samples of the individual time series easily.

Finally, to speed up the variable selection process, the selection procedure will utilize only the linear ridge regression model, with the selected variables being subsequently used in all of the models. This is necessary in our setting, as the calculations of some of the models are too time consuming in order to be used for a largescale variable selection. We thus have to rely on the assumption that indicators extracting useful features for the ridge regression model would also provide useful features to the more complex models as well.

The indicators entering the selection procedure are then the following ones: Momentum (MOM), Rate of Change (ROC), Relative Strength Index (RSI), Commodity Channel Index (CCI), Fast Stochastic Oscillator (FSO) and Internal Bar Strength Indicator (IBS), each one of them alternatively with the values of its period parameter equal to $p=\{2, 3, 5, 8, 13, 21, 34, 55, 89, 144\}$, with the exception of IBS for which the option of $p=1$ is tested too as it is popular among traders. This gives us a total of 81 technical indicators that enter the selection procedure and could potentially be added into the model.

While the selection procedures are often run by starting with an empty predictor set, adding different predictors one by one. In our case, we apply it in order to extend the original predictor set comprising of the last 5-day price movements or the last 5-day price returns. The goal of the selection procedure is thus to rather extend the original variable set with additional potentially useful variables, than to replace the original variable set with technical indicators altogether. The reason for this is that the initial tests showed that by starting with an empty predictor set the procedure often ends with highly overfitted set of selected predictors, with poor results in the out-sample period.

The selection procedure will proceed as follows:

1. The procedure starts with the predictor variable matrix, containing for each time point in the time series either the last 5-day price movements or the last 5-day returns (i.e. two alternative options are tested).
2. In every step of the selection, every one of the 81 technical indicators are added (one by one) into the variable matrix, a linear ridge regression model is estimated, predictions for all of the 10 currency exchange rate time series are computed and Drawdown ratios for the two

validation sample periods are computed, for a system trading all of the 10 currency exchange rates at once.

3. At the end of every iteration in the main cycle, the variable which caused the model to achieve maximum possible Drawdown ratio in Validation sample A, is permanently added into the model
4. The selection procedure ends either when no additional variable is able to increase the Validation sample A Drawdown ratio, or, alternatively, if the incorporation of the best variable (according to the Validation sample A) into the model leads to a decrease of the drawdown ratio in the Validation sample B (i.e. the early stopping criterion)

Figure 11 shows the evolution of the Drawdown ratios in the different iterations of the forward selection procedure algorithm starting with the last 5-day price movements as the initial predictor set, and using the following one-day price return as the target variable. Specifically, Figure 11 shows the evolution of the Drawdown ratios in the Validation sample A, Validation sample B and the Testing sample for the 3 iterations of the selection procedure. In the third iteration, the procedure was terminated by breaching the early stopping criterion (after adding the third proposed variable into the variable set, the Validation sample B Drawdown ratio decreased).

Altogether the procedure selected 2 additional variables into the model, CCI(5) in the first iteration and RSI(3) in the second iteration. In the third iteration, ROC(8) was added into the model, but as it decreased the Drawdown ratio in Validation sample B, it will not be a part of the final variable set used for the model results evaluation. The final variable set does thus contain altogether 7 variables (5 past price movements, CCI with period 5 and RSI with period 3).

We can see from Figure 11 that the combination of 2 selected variables does actually achieve a worse Drawdown ratio value in the Testing sample than either 1 selected variable or 3 variables. Nevertheless, as the Testing sample represents an out-sample period, simulating the real application of the models for trading purposes, we cannot base the variable selection on the results of the models in this sample, as this would lead to a forward looking bias influencing the results of the study. Therefore, as only the Validation sample A and B can be use for variable selection, we will work with the two selected variables from the first two iterations of the procedure.

Figure 12, Figure 13, Table 14, Table 15 and Table 16 show the results of all of the tested data mining models, applied to a predictor set selected with a Ridge Regression based selection procedure starting with an initial predictor set comprising of the last 5-day price movement for each day. As already mentioned, the selection procedure ended during the third iteration of the selection, adding altogether 2 additional predictors into the predictor set: CCI(5) and RSI(3).

Figure 11 – Drawdown ratios in the Validation sample A, Validation sample B and the Testing sample for the 3 iterations of the forward selection algorithm starting with the 5 past price movements as the initial predictor variable set

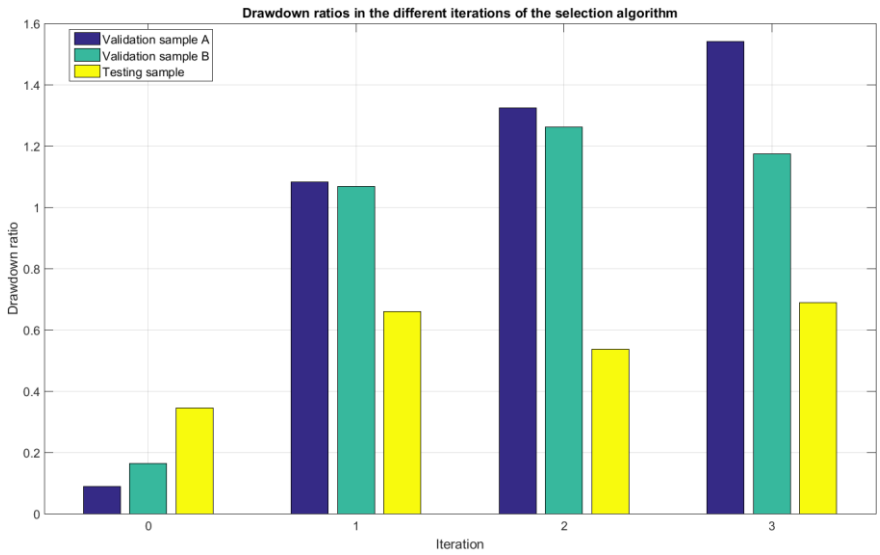


Figure 12 and Figure 13 show the results of all of the tested data mining models with the predictor set determined by the selection procedure starting with the past price movements. We can see that the profitability of the models is unfortunately not much better than in the case when only the basic set of the last 5 price movements was used as an input to the models. The three currencies that exhibited certain profitability in the previous case, i.e. the EURJPY, EURUSD and GBPJPY exchange rates seem to be more or less profitable in this case as well. Again, the ridge regression methods are among the most robust ones, achieving generally positive profits for ERUSD, GBPJPY, GBPUSD, USDCAD and USDJPY. The results of the other methods are, on the other hand, inconsistent and mostly unprofitable.

Table 14 shows the aggregated results of the tested methods, in order to evaluate in more detail, which method is the most robust one. The results clearly show the Linear Ridge regression without PCA to be the most consistently profitable method out of the ones tested, followed by the Quadratic Ridge Regression without PCA and the Linear Ridge regression with PCA. The Manhattan distance based *k*-NN, which performed well in the previous tests, achieves an above average performance in the current test as well, nevertheless, in spite of this, it is significantly behind the mentioned ridge regression models.

The clear domination of the Linear Ridge Regression model in the performed testing can potentially be explained by the fact that this was the method used for the variable selection, causing the selected variable set to be most useful primarily for this single method. Our hypothesis that variable selected based on the criterion that they increase the predictive power of the ridge regression, would have a positive impact on the predictive power of the other models as well, was not confirmed as the results of the other models do not seem to be significantly better than in the testes without any variable selection and in the case of the Manhattan based *k*- NN, the performance of the method even significantly decreased after the new variables were added.

In Table 15 we can see the same tendencies as in Table 14, i.e. while the results of the ridge regression based methods seem, on average, to be better than in the previous tests, the variable selection (based on the ridge regression) does not seem to improve the performance of the other methods at all, while significantly decreasing it for the Manhattan distance based *k*-NN.

From Table 16, we see that the most profitable currency in the test utilizing the variable selection is the GBPJPY exchange rate, followed by EURJPY and EURUSD.

Figure 12 – Drawdown ratios of the 18 tested model specifications applied to the empirical time series of 5 currency exchange rates (1. AUDUSD, 2. EURGBP, 3. EURJPY, 4. EURUSD, 5. GBPJPY) (selection starting with past price movements)

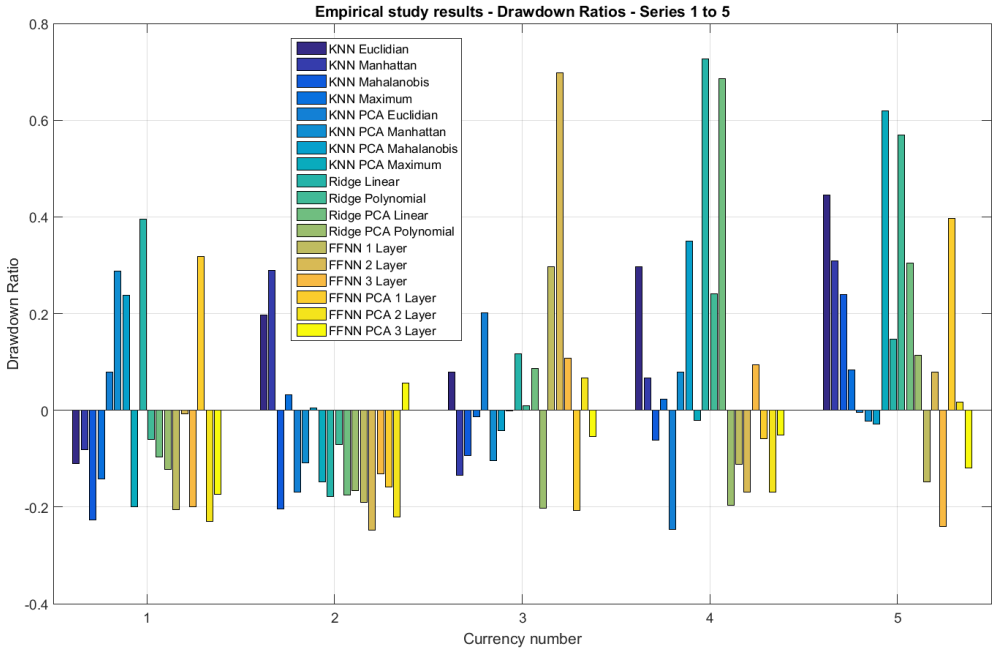


Figure 13 – Drawdown ratios of the 18 tested model specifications applied to the empirical time series of 5 currency exchange rates (6. GBPUSD, 7. NZDUSD, 8. USDCAD, 9. USDCHF, 10. USDJPY) (selection starting with past price movements)

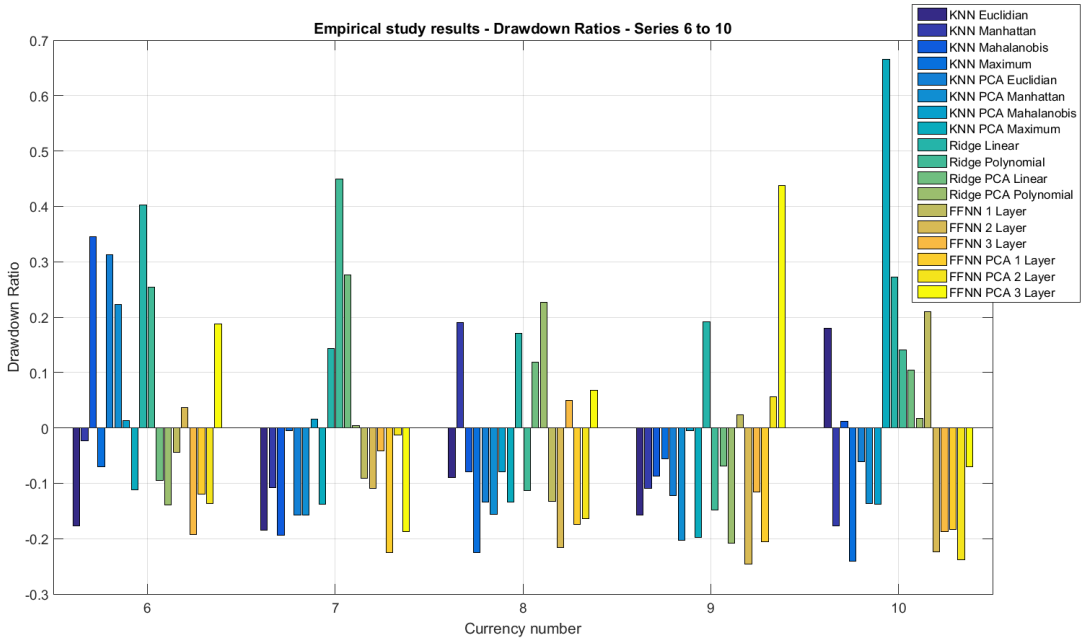


Table 14 – Aggregated performance of the tested models in the empirical study (selection based on past price movements)

	Average DD_Ratio	Highest DD_Ratio	Lowest DD_Ratio	Average rank	Percent profitable	Percent of best	Percent >median
KNN_Euclidian	0,05	0,45	-0,19	10,60	0,50	0,00	0,60
KNN_Manhattan	0,02	0,31	-0,18	10,90	0,40	0,10	0,70
KNN_Mahalanobis	-0,04	0,35	-0,23	8,00	0,30	0,00	0,50
KNN_Maximum	-0,06	0,08	-0,24	8,50	0,30	0,00	0,40
KNN_PCA_Euclidian	-0,03	0,31	-0,25	8,90	0,30	0,00	0,40
KNN_PCA_Manhattan	-0,03	0,29	-0,20	8,60	0,30	0,00	0,40
KNN_PCA_Mahalanobis	0,03	0,35	-0,14	11,50	0,50	0,00	0,70
KNN_PCA_Maximum	0,03	0,67	-0,20	9,30	0,20	0,20	0,30
Ridge_Linear	0,24	0,73	-0,18	15,10	0,90	0,30	0,90
Ridge_Polynomial	0,13	0,57	-0,15	12,90	0,60	0,10	0,80
Ridge_PCA_Linear	0,11	0,69	-0,18	12,30	0,60	0,00	0,80
Ridge_PCA_Polynomial	-0,07	0,23	-0,21	7,90	0,40	0,10	0,40
FFNN_1_Layer	-0,04	0,30	-0,21	8,90	0,30	0,00	0,40
FFNN_2_Layer	-0,04	0,70	-0,25	6,90	0,30	0,10	0,30
FFNN_3_Layer	-0,09	0,11	-0,24	8,20	0,30	0,00	0,50
FFNN_PCA_1_Layer	-0,06	0,40	-0,23	6,60	0,20	0,00	0,20
FFNN_PCA_2_Layer	-0,10	0,07	-0,24	6,30	0,30	0,00	0,30
FFNN_PCA_3_Layer	0,01	0,44	-0,19	9,60	0,40	0,10	0,40

Table 15 – Aggregated profitability of the tested models in the empirical study for the case when each model is used for a simultaneous trading on all of the currency exchange rate time series (selection starting with past price movements)

Model	Total profit	Max Drawdown	MDD Ratio	MDD Ratio p.a.	Relative Rank
KNN_Euclidian	0,074	-0,752	0,098	0,025	14,000
KNN_Manhattan	-0,261	-1,325	-0,197	-0,050	13,000
KNN_Mahalanobis	-0,696	-1,102	-0,632	-0,159	9,000
KNN_Maximum	-0,684	-1,240	-0,551	-0,139	10,000
KNN_PCA_Euclidian	-0,682	-1,061	-0,643	-0,162	8,000
KNN_PCA_Manhattan	-0,582	-0,853	-0,682	-0,172	7,000
KNN_PCA_Mahalanobis	0,172	-0,912	0,189	0,048	15,000
KNN_PCA_Maximum	-0,326	-1,155	-0,282	-0,071	12,000
Ridge_Linear	1,688	-0,747	2,260	0,569	18,000
Ridge_Polynomial	0,763	-0,731	1,044	0,263	16,000
Ridge_PCA_Linear	0,836	-0,633	1,322	0,333	17,000
Ridge_PCA_Polynomial	-0,949	-1,076	-0,881	-0,222	3,000
FFNN_1_Layer	-1,088	-1,480	-0,736	-0,185	5,000
FFNN_2_Layer	-1,317	-1,465	-0,899	-0,227	2,000
FFNN_3_Layer	-2,176	-2,678	-0,813	-0,205	4,000
FFNN_PCA_1_Layer	-0,934	-0,971	-0,962	-0,242	1,000
FFNN_PCA_2_Layer	-1,518	-2,116	-0,718	-0,181	6,000
FFNN_PCA_3_Layer	-0,410	-1,118	-0,367	-0,092	11,000

Table 16 – Aggregated profitability of the tested models in the case when all of them are traded simultaneously on each of the currencies (selection starting with past price movements)

	Total profit	Max Drawdown	MDD Ratio	MDD Ratio p.a.	Relative Rank
AUDUSD	-2,174	-2,542	-0,855	-0,216	2,000
EURGBP	-1,337	-1,476	-0,906	-0,228	1,000
EURJPY	0,227	-2,311	0,098	0,025	9,000
EURUSD	0,185	-2,080	0,089	0,022	8,000
GBPJPY	2,020	-3,738	0,540	0,136	10,000
GBPUSD	-0,191	-1,519	-0,126	-0,032	7,000
NZDUSD	-1,595	-2,444	-0,653	-0,164	6,000
USDCAD	-1,632	-2,455	-0,665	-0,168	5,000
USDCHF	-2,075	-2,733	-0,759	-0,191	4,000
USDJPY	-1,518	-1,965	-0,772	-0,195	3,000

In the second test of the proposed forward selection procedure, we start with the initial variable set comprising of the last 5 day asset price returns and use the following one day return as the target quantity for all of the models. Otherwise, the procedure stays the same.

Figure 14 shows that the Drawdown ratios in the different iterations of the return based selection procedure. We can see that the procedure selected only a single variable into the model, specifically the CCI(5), which was the first selected variable also in the previous, movements-based selection procedure. As the procedure proceeded to select a second variable, the Validation sample B Drawdown ratio strongly decreased, causing an early stopping of the selection procedure. We can further see from Figure 14 that in this case, the decrease of the Drawdown ration in the Validation sample B corresponds to the decrease of the Drawdown ratio also in the Testing sample.

Figure 14 – Drawdown ratios in the Validation sample A, Validation sample B and the Testing sample for the 3 iterations of the forward selection algorithm starting with the 5 past returns as the initial predictor variable set

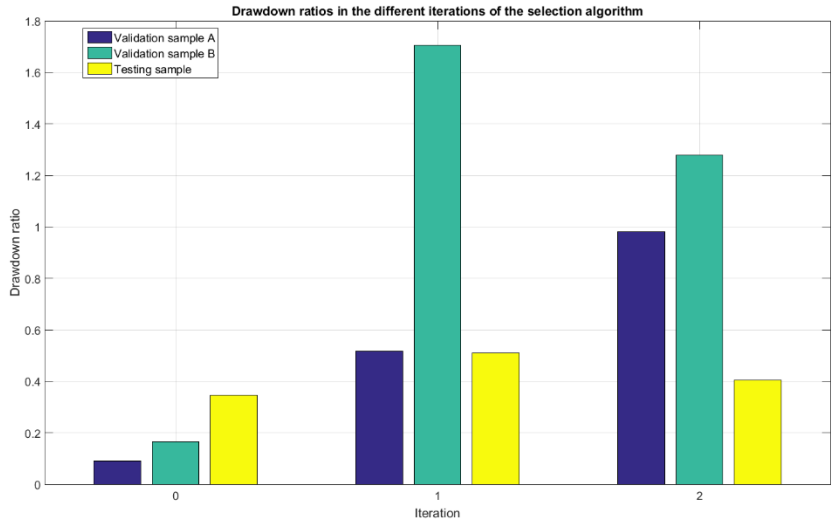


Figure 15, Figure 16, Table 17, Table 18 and Table 19 contain the results for all of the models in the case when the predictor set is determined by the ridge regression based forward selection procedure starting with the last 5 day returns as initial predictors. As we mentioned previously, the procedure did in this case add only a single additional variable into the predictor set, specifically the CCI oscillator with period 5 days.

Figure 15 – Drawdown ratios of the 18 tested model specifications applied to the empirical time series of 5 currency exchange rates (1. AUDUSD, 2. EURGBP, 3. EURJPY, 4. EURUSD, 5. GBPJPY) (selection starting with the past returns)

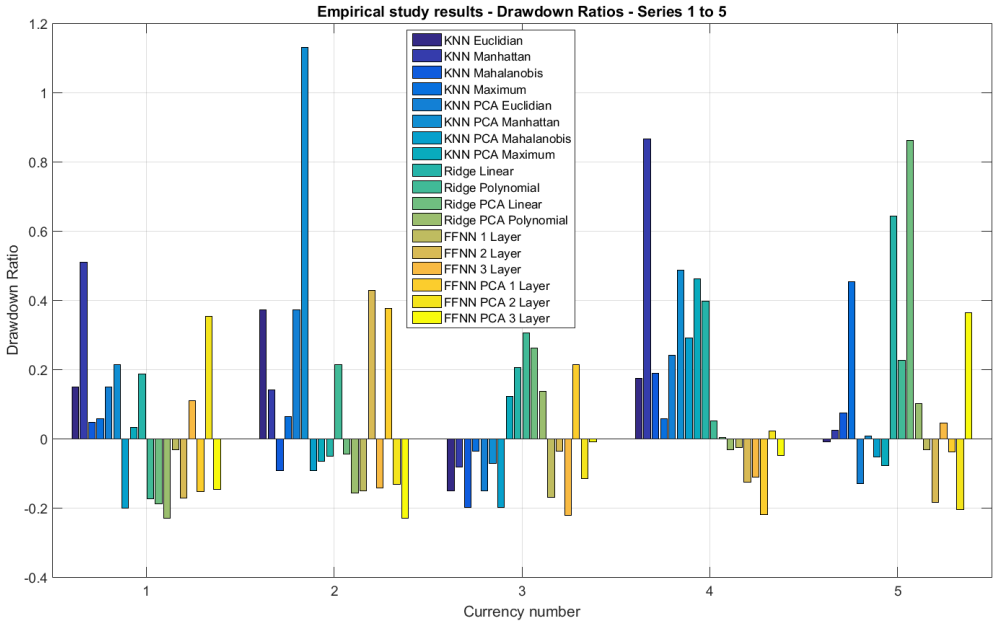


Figure 16 – Drawdown ratios of the 18 tested model specifications applied to the empirical time series of 5 currency exchange rates (6. GBPUSD, 7. NZDUSD, 8. USDCAD, 9. USDCHF, 10. USDJPY) (selection starting with the past returns)

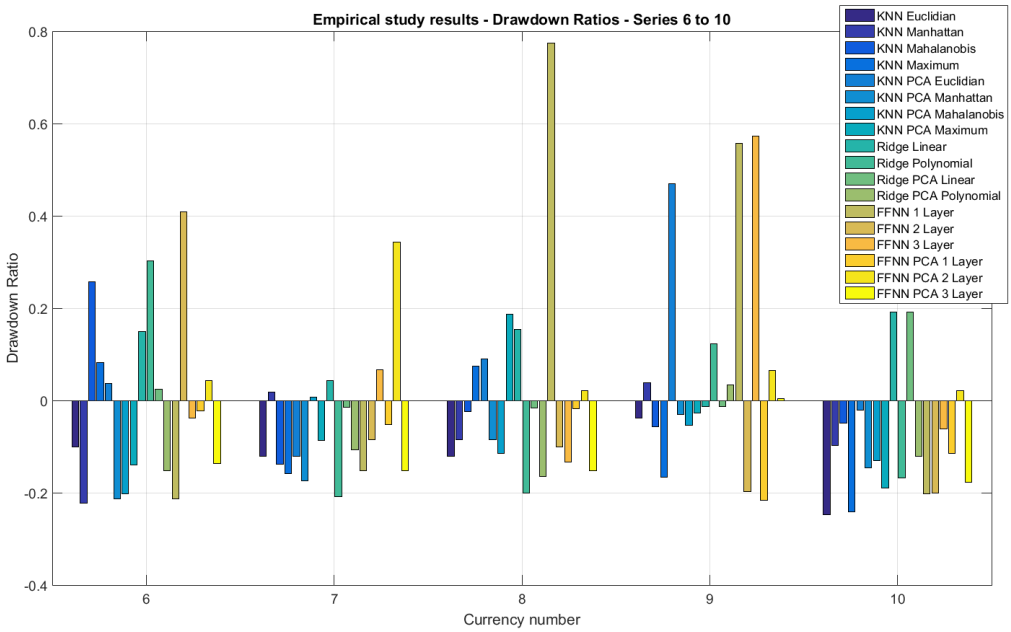


Table 17 – Aggregated performance of the tested models in the empirical study (selection starting with the past returns)

	Average DD_Ratio	Highest DD_Ratio	Lowest DD_Ratio	Average rank	Percent profitable	Percent of best	Percent >median
KNN_Euclidian	-0,01	0,37	-0,25	8,00	0,30	0,00	0,30
KNN_Manhattan	0,11	0,87	-0,22	11,60	0,60	0,20	0,70
KNN_Mahalanobis	0,00	0,26	-0,20	9,30	0,40	0,00	0,60
KNN_Maximum	0,02	0,45	-0,24	9,40	0,60	0,00	0,70
KNN_PCA_Euclidian	0,09	0,47	-0,15	11,50	0,60	0,00	0,70
KNN_PCA_Manhattan	0,11	1,13	-0,21	9,60	0,40	0,10	0,30
KNN_PCA_Mahalanobis	-0,07	0,29	-0,20	6,80	0,20	0,00	0,20
KNN_PCA_Maximum	0,02	0,46	-0,19	9,60	0,40	0,00	0,40
Ridge_Linear	0,19	0,64	-0,05	14,50	0,80	0,10	0,80
Ridge_Polynomial	0,05	0,31	-0,21	9,90	0,60	0,10	0,50
Ridge_PCA_Linear	0,11	0,86	-0,19	11,80	0,50	0,20	0,70
Ridge_PCA_Polynomial	-0,07	0,14	-0,23	7,30	0,30	0,00	0,40
FFNN_1_Layer	0,04	0,77	-0,21	7,30	0,20	0,10	0,20
FFNN_2_Layer	-0,03	0,43	-0,20	7,90	0,20	0,10	0,40
FFNN_3_Layer	0,01	0,57	-0,22	9,20	0,40	0,10	0,50
FFNN_PCA_1_Layer	-0,02	0,38	-0,22	9,00	0,20	0,00	0,60
FFNN_PCA_2_Layer	0,04	0,35	-0,21	11,20	0,70	0,10	0,60
FFNN_PCA_3_Layer	-0,07	0,36	-0,23	7,10	0,20	0,00	0,30

Table 18 – Aggregated profitability of the tested models in the empirical study for the case when each model is used for a simultaneous trading on all of the currency exchange rate time series (selection starting with the past returns)

Model	Total profit	Max Drawdown	MDD Ratio	MDD Ratio p.a.	Relative Rank
KNN_Euclidian	-0,644	-1,049	-0,613	-0,155	5,000
KNN_Manhattan	0,247	-0,634	0,389	0,098	16,000
KNN_Mahalanobis	-0,320	-1,060	-0,302	-0,076	9,000
KNN_Maximum	-0,102	-0,823	-0,124	-0,031	12,000
KNN_PCA_Euclidian	0,058	-1,326	0,044	0,011	14,000
KNN_PCA_Manhattan	-0,069	-0,746	-0,093	-0,023	13,000
KNN_PCA_Mahalanobis	-1,333	-1,824	-0,731	-0,184	3,000
KNN_PCA_Maximum	-0,100	-0,741	-0,135	-0,034	11,000
Ridge_Linear	1,448	-0,675	2,145	0,540	18,000
Ridge_Polynomial	0,118	-0,590	0,200	0,050	15,000
Ridge_PCA_Linear	0,613	-0,687	0,892	0,225	17,000
Ridge_PCA_Polynomial	-0,697	-0,915	-0,761	-0,192	2,000
FFNN_1_Layer	-0,553	-1,388	-0,399	-0,101	8,000
FFNN_2_Layer	-1,143	-1,608	-0,711	-0,179	4,000
FFNN_3_Layer	-0,448	-1,091	-0,411	-0,104	7,000
FFNN_PCA_1_Layer	-0,929	-1,087	-0,854	-0,215	1,000
FFNN_PCA_2_Layer	-0,164	-0,794	-0,206	-0,052	10,000
FFNN_PCA_3_Layer	-0,744	-1,255	-0,593	-0,149	6,000

From the results in Figure 15, Figure 16, Table 17, Table 18 we can see that the employed selection procedure slightly improved the performance for some of the *k*-NN based methods, the results are, however, rather inconsistent, and most of the tested methods do still lose on most of the analysed time series. The most robust model is in this case again the Linear Ridge Regression without PCA.

Table 19 shows the profitability of the tested methods when applied simultaneously for the trading on the different currencies. The only profitable currencies are in this test the EURUSD and the GBPJPY which is consistent for the previous results in which these currencies exhibited the best observed results as well.

Table 19 – Aggregated profitability of the tested models in the case when all of them are traded simultaneously on each of the currencies (selection starting with the past returns)

	Total profit	Max Drawdown	MDD Ratio	MDD Ratio p.a.	Relative Rank
AUDUSD	-0,096	-2,072	-0,046	-0,012	8,000
EURGBP	-0,043	-0,726	-0,059	-0,015	7,000
EURJPY	-1,776	-3,828	-0,464	-0,117	4,000
EURUSD	1,339	-1,462	0,916	0,231	10,000
GBPJPY	0,660	-2,993	0,221	0,056	9,000
GBPUSD	-0,972	-1,660	-0,585	-0,147	3,000
NZDUSD	-1,192	-1,959	-0,609	-0,153	2,000
USDCAD	-0,537	-1,471	-0,365	-0,092	5,000
USDCHF	-0,214	-1,808	-0,118	-0,030	6,000
USDJPY	-1,934	-2,137	-0,905	-0,228	1,000

Table 20 shows a comparison of the four tested variable sets, i.e. a set comprising of only the last 5 day price movements, the last 5 price returns, the last 5 day price movements + 5-period CCI and 3-period RSI and the last 5 day price returns with the 5-period CCI.

We can see that an improvement of results do to the additional explanatory variables can be observed mostly for the Linear Ridge Regression model and possible some other Ridge Regression models but not so much for the other methods. This indicates that the technical indicators (feature extractors) selected in order to improve the predictive power of the ridge regression model may not be the suitable ones for the other methods and that it would be better to run a separate selection procedure for each of the tested models (which would, however, be computationally very demanding). The employment of the selection procedures does also seem to only slightly improve the average Drawdown ratio for all of the models.

Table 20 – Comparison of the models applied with different sets of predictors (5 last price movements, 5 last price returns, 5 last price movements + CCI(5) + RSI(3) and 5 last price returns + CCI(5)

Drawdown Ratio p.a.	Price movements	Returns	Movements + selection	Returns + selection
KNN_Euclidian	-0,133	-0,159	0,025	-0,155
KNN_Manhattan	0,516	0,103	-0,050	0,098
KNN_Mahalanobis	-0,112	-0,112	-0,159	-0,076
KNN_Maximum	-0,165	-0,121	-0,139	-0,031
KNN_PCA_Euclidian	-0,235	-0,162	-0,162	0,011
KNN_PCA_Manhattan	-0,179	0,008	-0,172	-0,023
KNN_PCA_Mahalanobis	-0,192	-0,208	0,048	-0,184
KNN_PCA_Maximum	-0,021	-0,190	-0,071	-0,034
Ridge_Linear	0,366	0,384	0,569	0,540
Ridge_Polynomial	-0,012	-0,120	0,263	0,050
Ridge_PCA_Linear	0,378	0,021	0,333	0,225
Ridge_PCA_Polynomial	-0,138	-0,196	-0,222	-0,192
FFNN_1_Layer	-0,196	-0,149	-0,185	-0,101
FFNN_2_Layer	-0,230	-0,094	-0,227	-0,179
FFNN_3_Layer	-0,230	-0,191	-0,205	-0,104
FFNN_PCA_1_Layer	-0,195	-0,194	-0,242	-0,215
FFNN_PCA_2_Layer	-0,183	-0,233	-0,181	-0,052
FFNN_PCA_3_Layer	-0,152	-0,107	-0,092	-0,149
Average	-0,062	-0,096	-0,048	-0,032

Conclusion

In the study 3 different classes of data mining methods (*k*-Nearest Neighbour, Ridge Regression and Multilayer Perceptron Feed-Forward Neural Networks) are applied for the purpose of quantitative trading on 10 simulated time series, as well as real world time series of 10 currency exchange rates ranging from 1.11.1999 to 12.6.2015. Each of the employed methods is tested in multiple variants. The *k*-NN algorithm was applied alternatively with the Euclidian, Manhattan, Mahalanobis and Maximum distance function. The Ridge Regression was applied as Linear Ridge Regression as well as Quadratic Ridge Regression, and the Feed-Forward Neural Networks were applied alternatively with either 1, 2 or 3 hidden layers. In addition to the standard variant in which each of the methods was applied to the raw predictor set, all of the models were tested also in a version when the Principal Component Analysis (PCA) was used to perform dimensionality reduction of the variable set. The meta-parameters of the methods (number of *k* nearest neighbours, penalization term for the ridge regression, number of iterations, learning rate and neurons for the neural networks and the number of principal components in the PCA) were optimized on the validation sample.

In the simulation study a Stochastic-Volatility Jump-Diffusion model with self-exciting jumps was extended with 10 different non-linear conditional mean patterns in order to simulate 10 theoretical asset price time series and subsequently test whether the employed data mining methods are able to exploit the conditional mean dependencies to achieve trading profits. The results from the simulation study showed that no single method was able to exploit all of the non-linear patterns in the simulated time series, with each method profiting on some of them while losing on others. Two rounds of testing were performed with either the past 5 price movements or the past 5 asset returns as the predictor variables. In the first round, the quadratic ridge regression achieved the most robust results, followed by some of the *k*-NN methods, while in the second test, using the past returns, the *k*-NN methods were the most consistently profitable, followed by the linear ridge regression and the quadratic ridge regression. Although some of the Feed-Forward Neural Networks profited strongly on some of the time series, their results were inconsistent which we attribute to the short length of the analysed time series leading to overfitting. Additionally, we have not found the usage of PCA to either improve, nor worsen the performance of the tested methods on the analysed data sample.

In the second part of the study, the tested data mining models were applied to the empirical currency exchange rate time series. Overall, the profitability of most of the models was rather low, most of them ending with losses on the majority of the currency exchange rate time series. A surprising exception was the ridge regression method (linear regression with and without PCA), achieving relatively consistently profitable results on most of the currencies in both of the tests (i.e. either with price movements or returns as predictors). Additionally, the Manhattan distance based *k*-NN achieved consistently profitable results as well, while the other *k*-NN methods and the Neural Network based models achieved inconsistent results and loses on most of the analysed time series.

Finally, a forward selection procedure based on the linear ridge regression was applied, with the purpose of extending the original predictor set with a series of technical indicators. The results showed a limited power of the selection procedure to further improve the results for the linear ridge regression model, but when the chosen set of predictors was utilized by the other models, no improvement compared to the original predictor set was observed.

Additionally, we found that the EURUSD exchange rate is surprisingly the most promising for trading purposed with the data mining methods that we tested, followed possibly by EURJPY, GBPJPY and EURGBP. The high performance of the models on EURUSD is surprising as it is the most liquid currency which should theoretically exhibit the highest levels of information efficiency.

Literature

BERNAS, Marcin and PLACZEK, Bartlomiej, (2016). “Period-aware local modelling and data selection for time series prediction”, *Expert Systems With Applications*, Vol. 59 (2016), pp. 60–77

CAGINALP, G., LAURENT, H. (1998), “The predictive power of price patterns”, *Applied Mathematical Finance*, 1998, Vol. 5, pp. 181–205

CAVALCANTE, R.C., BRASILEIRO R.C., SOUZA, V.L.F., NOBREGA, J.P., OLIVEIRA, A.L.I., (2016). “Computational Intelligence and Financial Markets: A Survey and Future Directions”, *Expert Systems with Applications,* Vol. 55 (2016), pp. 194–211

EXTERKATE, P., GROENEN, P.J.F., HEIJ, CH., VAN DIJK, D. (2016). “Nonlinear forecasting with many predictors using kernel ridge regression”, International Journal of Forecasting, 2016, Vol. 32(3), pp. 736–753

FAMA, Eugene (1970). "Efficient Capital Markets: A Review of Theory and Empirical Work", *Journal of Finance*, 1970, 25 (2), pp. 383–417

FUNAHASHI, Ken-Ichi (1989). “On the Approximate Realization of Continuous Mappings by Neural Networks”, *Neural Networks*, 1989, Vol. 2, pp. 183-192

GOLDBERGER, J., HINTON, G., ROWEIS, S., SALAKHUTDINOV, R. (2005). “Neighbourhood Components Analysis”, *Advances in Neural Information Processing Systems*, 2005, Vol. 17, pp. 513-520.

HORNIK, Kurt (1989). “Multilayer Feedforward Networks are Universal Approximators”, *Neural Networks*, 1989, Vol. 2, pp. 359-366

CHRISTOFFERSEN, P. F., DIEBOLD, F. X., (2010). “Financial Asset Returns, Direction-of-Change Forecasting, and Volatility Dynamics”, Wharton Financial Institutions Center, Working Paper, 04-2010, pp. 1-41.

KROLLNER, B., VANSTONE, B., FINNIE, G., (2010). “Financial time series forecasting with machine learning techniques: A survey”, European Symposium on Artificial Neural Networks: Computational and Machine Learning, Bruges, Belgium, April 2010.

LAWRENCE, Ramon (1997). “Using Neural Networks to Forecast Stock Market Prices”, University of Manitoba, Department of Computer Science, December 1997, pp. 1-21

LO, Andrew (2004). "The Adaptive Market Hypothesis: Market Efficiency from an Evolutionary Perspective", *Journal of Portfolio Management,* 5-30, pp. 15–29.

MARTENS, H.A., DARDENNE, P. (1998). “Validation and verification of regression in small data sets”, *Chemometrics and Intelligent Laboratory Systems*, 1998, Vol. 44, pp. 99–121

MCNAMES, James, (2000). “Local Modeling Optimization for Time Series Prediction”, ESANN 2000, 8th European Symposium on Artificial Neural Networks Proceedings, Bruges, Belgium, April 2000.

WEINBERGER, K. Q., BLITZER, J. C., SAUL, L. K., (2006). "Distance Metric Learning for Large Margin Nearest Neighbor Classification*", Advances in Neural Information Processing Systems,* 18, 1473–1480

BAILEY, D.H., BORWEIN, J.M., DE PRADO, M.L., ZHUX, Q.J., (2015). “The Probability of Backtest Overfitting”, 2015

TEKEUCHI, Lawrence, LEE, Yu-Ying, (2013). “Applying Deep Learning to Enhance Momentum Trading Strategies in Stocks”, 2013