Trading algorithms for high-frequency currency trading

Shahab Garoosi
Abstract

This thesis uses modern portfolio theory together with machine learning techniques to generate stable portfolio returns over eleven currency pairs with spreads included. The backtests show that support vector machine predicted future returns better than neural network and linear regression. Principal component analysis and data smoothing combined with the local outlier factor further improved the performance of the trading algorithm. However, the ensemble of the top performed predictor performed below the individual predictors. Also, the use of different error estimates showed the criticality of mean arctangent absolute percentage error over mean absolute error and over mean squared error for profitability. For obtaining sensible results in a transaction costless setting, adopting risk adjusted leverage proved necessary. Otherwise, the profit-maximizing leverage surpassed the risk adjusted in a spread setting.
Sammanfattning

Acknowledgement

I want to thank my supervisor Niklas Lundström. Without his help and support the thesis would not have been completed. I also want to thank my examiner Markus Ådahl for his feedback which added significantly to this thesis.
Contents

Abstract
Sammanfattning
Acknowledgement

1. Introduction .................................................................................................................. 7
   1.1. Background, problem description and methodology .............................................. 7
   1.2. Outline .................................................................................................................... 8
2. Theory............................................................................................................................. 9
   2.1. Modern portfolio theory (MPT) .............................................................................. 9
   2.2. Utility theory for portfolio selection ..................................................................... 11
   2.3. Error Estimates ..................................................................................................... 11
   2.4. Fixing broken correlation matrix ......................................................................... 12
3. Machine learning ......................................................................................................... 14
   3.1. Meta learning ........................................................................................................ 14
   3.2. Regression ............................................................................................................ 14
      Linear regression ....................................................................................................... 14
      Neural network ......................................................................................................... 14
      Support vector machine (SVM) ................................................................................ 16
   3.3. Dimensionality reduction ..................................................................................... 17
      Principal component analysis (PCA) ....................................................................... 18
   3.4. Data smoothing .................................................................................................... 18
      Laplacian smoothing ............................................................................................... 18
   3.5. Local outlier factor (LOF) ................................................................................... 19
   3.6. Ensemble Learning ............................................................................................... 19
4. Methods ......................................................................................................................... 20
   4.1. Data set ................................................................................................................. 20
   4.2. Regressors ............................................................................................................ 20
   4.3. Training and testing set ....................................................................................... 20
   4.4. Feature selection .................................................................................................. 21
   4.5. Data transformation ............................................................................................. 21
   4.6. Utility function ..................................................................................................... 22
   4.7. Performance measures ........................................................................................ 23
   4.8. Implementation ..................................................................................................... 24
5. Result .............................................................................................................................. 25
   5.1. Linear regression ................................................................................................. 28
1. Introduction

This chapter introduces the thesis. It starts with the background and problem description of the thesis, and ends with the methodology and outline.

1.1. Background, problem description and methodology

According to the efficient-market hypothesis, market efficiency effectively eliminates any profit opportunities. This makes it impossible for making secure profits in the market. However, it does not exclude the possibility for making profit under risk. The aim of this thesis is to build an as safe and profitable trading algorithm as possible. To achieve this goal, we base our algorithm on modern portfolio theory to diversify risk. To minimize the impact of risk further, we account for that risk varies over time. Thus, we may diversify risk over time by adjusting leverage accordingly. For example, in a high risk setting we may prefer modest leverage, while in a low risk setting we may prefer a more aggressive leverage. For this purpose, we introduce utility theory to determine the appropriate leverages.

To estimate the model parameters, historical prices are statistically analysed. Furthermore, the digitalisation has enabled the use of sophisticated data analysing tools. Here, machine learning is a collection of these tools. Their aim is to perform tasks by learning from data. In this thesis, we apply machine learning techniques in our trading algorithm for forecasting and data pre-processing.

Machine learning tools have been extensively applied to trading, where their success have been recognised, see for example (Hsu, Lessmann, Sung, Ma and Johnson, 2016). Among these support vector machine (SVM) and neural networks constitute, due to their nonlinear fitting ability, the leading forecasting tool for financial applications, (Hsu, Lessmann, Sung, Ma and Johnson, 2016). The merging of the two and of multiple predictors have similarly been explored with improved result on accuracy, (Ballings, Van den Poel, Hespels, and Gryp, 2015). In this thesis, we employ these methods, SVM, neural networks and the merging of the optimal performed predictors, and evaluate their performances to each other and to linear regression.

Another aspect of forecasting is overfitting. To treat this effect, machine learning techniques exist as well. For example, (Zhong and Enke, 2017) applied principal component analysis (PCA), a dimensionality reduction technique, to neural networks with improved result on accuracy. In this thesis, we employ PCA, local outlier factor and data smoothing to reduce overfitting.
To increase the profitability of our algorithm further, the algorithm is designed for high-frequency trading, since this enables a greater number of trading opportunities. To accommodate this setting, we introduce transaction costs in the form of spreads to our algorithm, which become notable in the smaller time frames.

To test the algorithm’s performance, it is backtested against historical data on a set of currency pairs. The reason for choosing currencies is because of their high liquidity which enables a higher trading frequency. An additional benefit of trading currencies over stocks is safety. Indeed, a country is less likely to go bankrupt than a company.

1.2. Outline
This thesis has six chapters. Chapters 2 and 3 outlines the theory underlining our algorithm respectively the machine learning tools used by our algorithm. These are followed by Chapter 4 which presents the details of the backtests and the algorithm designs. Chapters 5 and 6 ends the thesis with the result and conclusion part.
2. Theory

This chapter presents the theory used by our algorithm. It starts with Section 2.1 which reviews modern portfolio theory, the main theory in our algorithm, which then follows by Section 2.2 presenting the utility theory for portfolio selection. Section 2.3 covers different error estimates, and Section 2.4 deals with inconsistent estimations.

2.1. Modern portfolio theory (MPT)

MPT determines the optimal portfolio from a collection of one risk-free asset and \( n \) risky assets, (Aldridge, 2009). The risk-free asset is assumed to have a future return \( r_0 \) and the risky assets are assumed to have corresponding random variable returns \( r_i \) where \( i = 1, \ldots, n \) is an indexing of the risky assets. Here, the return is defined as the growth in value, that is

\[
    r_t = \frac{P_{t+1} - P_t}{P_t}
\]

where \( P_t \) is the asset price at time \( t \). Furthermore, each \( r_i \) is assumed normally distributed with expected value \( r_i^e \) and variance \( \sigma_i^2 \). In addition, each pair \( r_i \) and \( r_j \) is assumed correlated with covariance

\[
    \Sigma_{ij} = \rho_{ij} \sigma_i \sigma_j
\]

where \( \rho_{ij} \) is the correlation between asset \( i \) and \( j \) and satisfies \(-1 \leq \rho_{ij} \leq 1 \) and \( \rho_{ii} = 1 \). Under these assumptions, a portfolio \( p \) has a normally distributed random variable return \( r_p \) with expected return

\[
    r_p^e = x r^e
\]

and variance

\[
    \sigma_p^2 = x \Sigma x^T
\]

where \( x = [x_0 \ x_1 \ldots x_n] \) are the asset weights in the portfolio \( p \) and \( r^e = [r_0^e \ r_1^e \ldots r_n^e]^T \).

The optimal portfolio then becomes the portfolio which maximizes \( r_p^e \) given \( \sigma_p^2 \), or conversely the portfolio which minimizes \( \sigma_p^2 \) given \( r_p^e \). Thus, the optimal portfolio problem can be formulated as minimizing

\[
    \sigma_p^2 = x \Sigma x^T = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j \rho_{ij} \sigma_i \sigma_j
\]

subject to \( \sum_{i=0}^{n} x_i = 1 \) and \( r_p^e = \sum_{i=0}^{n} x_i r_i^e \). This procedure to minimize risk by considering the correlations between assets is termed risk diversification.

The optimal portfolio problem stated above can be subsequently solved by the Lagrangian...
\[
\mathcal{L} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j \rho_{ij} \sigma_i \sigma_j - \lambda \left[ \sum_{i=0}^{n} x_i r_i^e - r_p^e \right] - \mu \left[ \sum_{i=0}^{n} x_i - 1 \right],
\]

which has the solutions

\[
\frac{\partial \mathcal{L}}{\partial x_i} = \sum_{j=1}^{n} x_j \rho_{ij} \sigma_i \sigma_j - \lambda r_j^e - \mu = 0 \quad (1)
\]

for \( i = 0, 1, \ldots, n \) subject to \( \sum_{i=0}^{n} x_i = 1 \) and \( r_p^e = \sum_{i=0}^{n} x_i r_i^e \).

In MPT (when transaction costs are excluded), all optimal portfolios are unique up to leverage. Geometrically, this can be illustrated by the line which is tangent to the efficient frontier and which intercepts the y-axis at \( r_0 \) (see Figure 2.1.1). Here, the efficient frontier displays the risk-return combinations of the optimal portfolios with the risk-free asset excluded. Thus, the tangent displays the corresponding combinations of the optimal portfolios with the risk-free asset included.

\[\text{Figure 2.1.1. The optimal portfolios in MPT when transaction costs are excluded.}\]

In high-frequency trading, transaction costs from the bid-ask spread become notable. Accounting for these costs, the optimal portfolios in MPT will instead be

\[
\arg \max_x E[x r^T - TC] - \lambda V[x r^T - TC]
\]

where \( \lambda \) is the level of risk aversion; here, a higher \( \lambda \) corresponds to a greater risk aversity, a lower \( \lambda \) corresponds to lower risk aversity, and zero \( \lambda \) corresponds to risk neutrality. In this thesis, the transaction costs \( TC \) will be consisting of the spread which is the cost of
buying/selling a unit of an asset. Thus, $TC$ becomes proportional to the magnitude of the order size,

$$TC = |x|S^T$$

where $S_i$ in $S = [S_0 S_1 ... S_n]$ is the spread of asset $i$ and $|x| = [|x_0| |x_1| ... |x_n|]$.

2.2. **Utility theory for portfolio selection**

In the previous section, we observed that the optimal portfolios depended additionally on risk and reward preferences. These factors determine the leverage. For example, adopting a risk averse strategy, we raise our leverage when risk drops and conversely if it rises. To capture these adjustments, we introduce utility theory. Utility theory determines the optimal choice by assigning a cardinal preference ordering to choices, called utilities, (Lee, Finnerty, Lee, Lee and Wort, 2013). Thus, the optimal choice becomes the utility optimizing choice. Assuming risk aversion, we subtract a risk term $\sigma^b$, for a positive constant $b$, from our utility. Equivalently, adopting positive return preferences, we add $(r^e)^a$, for a positive constant $a$, to our utility. In result, our utility function becomes

$$U(r^e, \sigma) = (r^e)^a - c_\sigma^b$$

where $c$ is a positive constant. Furthermore, we may impose diminishing marginal utilities for risk and return, $a < 1$ and $b > 1$. Diminishing marginal utility means diminishing utility of each added return and risk unit. This is a reasonable assumption, since each added return unit can be assumed to be less important, while each added risk unit be more ruinous.

2.3. **Error Estimates**

There are different error estimates for assessing the accuracy of a forecast. In this thesis, we consider three estimates. These are the mean square error (MSE), the mean absolute error (MAE) and the mean absolute percentage error (MAPE), see (Aldridge, 2009). If $\hat{X}_i$ is the predicted value and $X_i$ is the corresponding observed value, then the error estimates are calculated as

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (X_i - \hat{X}_i)^2,$$

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |X_i - \hat{X}_i|$$

respectively.
$\text{MAPE} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{X_i - \hat{X}_i}{X_i} \right|$

for $n$ sample points. The MAPE has the drawback that it is undefined at $X_i = 0$. To overcome this problem, we convert it to the mean arctangent absolute percentage error (MAAPE),

$\text{MAAPE} = \frac{1}{n} \sum_{i=1}^{n} \arctan \left( \left| \frac{X_i - \hat{X}_i}{X_i} \right| \right)$

as proposed in (Kim and Kim, 2016). If the MAPE measures the slope ratio with the numerator $X_i - \hat{X}_i$ being the vertical change and the denominator $X_i$ the horizontal change, then the MAAPE can be seen as measuring the slope angle. Thus, when the MAPE approaches infinity, the MAAPE approaches its bounded maximum, $\pi/2$ radians.

2.4. Fixing broken correlation matrix

When statistically estimating the variances and correlations, inconsistencies may arise. This is because the estimated correlation matrix $p \in \mathbb{R}^{n \times n}$ may not be positive semi-definite,

$x^T px \geq 0$ for all $x \in \mathbb{R}^n$,

which results in a negative variance estimate. To fix this problem, we replace the correlation matrix $p$ with the closest positive semi-definite correlation matrix $X$, as proposed in (Higham, 2002), that is we minimize

$\|p,X\|$

subject to the positive semi-definiteness conditions $X = X^T$, $-1 \leq X_{ij} \leq 1$ and $X_{ii} = 1$ for all $1 \leq i \leq n$ and $1 \leq j \leq n$. Here, the distance $\|X,Y\|$ between the $n \times n$ matrices $X$ and $Y$ is defined in the different error metrics as

$\text{MSE} = \sqrt{\sum_{i=1}^{n} \sum_{i=1}^{n} (X_{ij} - Y_{ij})^2}$

$\text{MAE} = \sum_{i=1}^{n} \sum_{i=1}^{n} |X_{ij} - Y_{ij}|$

and
\[
\text{MAAPE} = \sum_{i=1}^{n} \sum_{i=1}^{n} \arctan \left( \frac{|X_{ij} - Y_{ij}|}{X_{ij}} \right).
\]

To in practice determine the corrected correlation matrix \( X \) local exhaustive search around the estimated correlation matrix \( p \) can be used. The search can further be made efficient by applying various heuristics.
3. Machine learning

This chapter presents the machine learning tools used in our algorithm. These include meta learning, different regression methods, dimensionality reduction techniques, data smoothing; outlier detection, and ensemble learning.

3.1. Meta learning

A central learning idea we use in our algorithm is meta learning. If machine learning is about learning an algorithm to perform a task, then meta learning is about automating the algorithm’s own learning, (Brazdil, Giraud-Carrier, Soares and Vilalta, 2008). In practice, this could for example involve technique, strategy and/or parameter optimization. In our algorithm, we use it specifically to optimize the number of variables for regression, where optimality is meant with respect to the chosen error metric.

3.2. Regression

Regression is a central learning method in our algorithm. Regression estimates the statistical relation \( f(x) = E[y|x] \) between variables \( x \) and \( y \) from a sample data \{ \((x^1,y^1),..., (x^l,y^l)\)\}, where \( l \) is the sample size. In our case, we use regression to predict future returns \( y = r_{t+1} \) from past returns \( x = (r_{t-k+1},..., r_t) \).

**Linear regression**

Linear regression linearly fits the input data to the output, in other words

\[
f(x) = \sum_{i=1}^{k} w_i x_i + b
\]

where \( b \) and \( w \) are constants. The fit is determined by the \( b \) and \( w \) which minimizes the sum of errors squared.

**Neural network**

Neural networks are nonlinear regression models, (Gurney, 1997). They consist of an input layer, a hidden layer and an output layer. Each layer in turn consists of a set of units, called neurons. The neurons in the input and output layer correspond to the input respectively output variables. In the hidden layer, each neuron receives weighted inputs from the neurons in the input layer. Depending on if the inputs’ aggregate exceeds a certain threshold, the received neuron sends a weighted signal to the neuron in the output layer. The neuron in the output layer, lastly, aggregates the received signals from the hidden layer and produces an output depending on if the aggregation exceeds a certain threshold. Formally, the output of neuron \( j \) is computed
by the step function

\[ x_j = \begin{cases} 
1 & \text{if } \sum_i w_{ij} x_i \geq b_j \\
0 & \text{if } \sum_i w_{ij} x_i < b_j 
\end{cases}, \]

where \( w_{ij} \) is the weight which the neuron \( i \) impacts neuron \( j \) and \( b_j \) is the bias (or threshold). Graphically, the inputs and signals are illustrated by links between nodes, where the nodes illustrate the neurons. To account for continuous (output) quantities, as is the case with regression, the output function of each neuron \( j \) is smoothened to

\[ x_j = \frac{1}{1 + \exp(\sum_i w_{ij} x_i + b_j)}. \]

Here, the fit is determined by the \( b \) and \( w \) which minimizes the sum of the errors squared.

To in practice fit neural networks, the numerical method of Levenberg-Marquardt (L-M) is, to the author’s knowledge, the fastest method for moderately sized neural networks, (Hagan and Menhaj, 1994). The L-M algorithm is a second order iterative method which combines the quasi Newton method with the gradient descent. The idea is that the Newton is faster and more accurate near an error minimum than the gradient method. Thus, the gradient method can first be used to point the direction to the minimum from where the Newton method can be subsequently applied. Because of this the L-M method finds only the local minimum. In the method, the Hessian matrix \( H \) is approximated by the Jacobian matrix \( J \) of the network errors with respect to the weights and biases, such that

\[ H = J^T J, \]

whereas the gradient is calculated as

\[ g = J^T e, \]

where \( e \) is a vector of network errors. Here, the Jacobian \( J \) is estimated through backpropagation, which is a numerical method for calculating network errors by running samples of random values through the neural network with the network weights constantly updated. Thus, the solution is given by the iteration

\[ v := v - (J^T J + \mu I)^{-1} J^T e, \]

where \( v \) is the concatenating matrix of \( b \) and \( w \); and where \( \mu \) regulates the weight between the gradient and the Newton method, and decreases after each iteration to shift the weight from the former to the latter.
Support vector machine (SVM)

SVMs are nonlinear regression models, (Smola and Schölkopf, 2004). SVM fits such that the function fitted deviates at most a distance $\varepsilon$ from the sample values $y^i$ at the sample points $x^i$ while being as flat as possible. To put it in formal terms, it facilitates to begin with the linear case. In the linear case, the function is

$$f(x) = \langle w, x \rangle + b$$

where $\langle *, * \rangle$ is the inner product and $w$ is a constant. The flatness, thus, translates to minimizing the magnitude of $w$, in other words the norm $\|w\|^2 = \langle w, w \rangle$. Hence, the fit is determined by minimizing

$$\frac{1}{2} \|w\|^2$$

subject to

$$\begin{cases}
    y^i - \langle w, x^i \rangle - b \leq \varepsilon \\
    \langle w, x^i \rangle + b - y^i \leq \varepsilon'
\end{cases}$$

where the inequality constraints ensure the $\varepsilon$-precision. The demand for $\varepsilon$-precision may, however, be infeasible; for example, a point $x^i$ may have sample values $y^i$, which differ more than a distance $2\varepsilon$ from each other, making it impossible for the $\varepsilon$-condition to hold. To cope with this obstacle, the $\varepsilon$-constraint is relaxed by introducing slack variables $\xi_i$ and $\xi^*_i$ at each sample point $i$. Thus, the optimization problem modifies to minimizing

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^{l} (\xi_i + \xi^*_i)$$

subject to

$$\begin{cases}
    y^i - \langle w, x^i \rangle - b \leq \varepsilon + \xi_i \\
    \langle w, x^i \rangle + b - y^i \leq \varepsilon + \xi^*_i, \\
    \xi_i, \xi^*_i \geq 0
\end{cases}$$

for all $i$, where the constant $C > 0$ determines the amount of the flatness of $f$ traded for the amount of deviation beyond $\varepsilon$ allowed. To extend the linear SVM to the nonlinear case, the input space is transformed by a nonlinear map $\phi$. Since $\phi(*)$ occur only as $\langle \phi(*), \phi(*) \rangle$ in the equations, $\langle \phi(*), \phi(*) \rangle$ is abbreviated by the kernel $K(*,*)$. In our case, we use the radial basis kernel $K(x^i, x) = \exp(-\|x^i - x\|)$, which is a normal distributed surface around $x^i$. Finally, formulating the optimization problem above in dual form gives the fit as

$$f(x) = \sum_{i=1}^{l} (\alpha_i - \alpha^*_i) K(x^i, x) + b.$$
with the coefficients $a_n$, $a'_n$ and $b$ minimizing the Lagrangian

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} (\alpha_i - \alpha'_i) (a_j - a'_j) K(x^i, x^j) + \epsilon \sum_{i=1}^{l} (\alpha_i + \alpha'_i) - \sum_{i=1}^{l} y^i (\alpha_i + \alpha'_i)$$

subject to

$$\begin{align*}
\sum_{i=1}^{l} (\alpha_i - \alpha'_i) &= 0 \\
0 &\leq \alpha_i \leq C \\
0 &\leq \alpha'_i \leq C
\end{align*}$$

for all $i$. A SVM fit using the radial basis kernel, thus, becomes a linear combination of normally distributed surfaces. To additionally compute $b$, the following conditions

$$\begin{align*}
\alpha_i \left( \epsilon + \xi_i - y_i + f(x_i) \right) &= 0 \\
\alpha'_i \left( \epsilon + \xi'_i - y_i + f(x_i) \right) &= 0 \\
\xi_i (C - \alpha_i) &= 0 \\
\xi'_i (C - \alpha'_i) &= 0
\end{align*}$$

need to hold for all $i$. The conditions assert that the $K(x^i, x)$-coefficient of $f$ is zero if the sample point $i$ is within $\epsilon$-precision from $f$.

The most popular approach to fit the SVM in practice is, to the author’s knowledge, the sequential minimal optimization (SMO) method, (Platt, 1999). SMO optimizes and solves the Lagrangian above by subsequently pair optimizing its parameters. Because of the simplicity of the pair optimization problems, they allow to be analytically solved. This process is iterated until the solution converges. To obtain faster convergence, the initial points in the iteration are selected by some heuristic.

### 3.3. Dimensionality reduction

To avoid overfitting, it is crucial that the number of input variables $k$ do not exceed the sample size $l$. Furthermore, it is preferable to have $k \ll l$. (However, too few input variables can also lead to underfitting.) Therefore, it is desirable to reduce the number of input variables while retaining as much of necessary information as possible. This process of reducing variables is called dimensionality reduction.
**Principal component analysis (PCA)**

PCA is a dimensionality reduction technique, which linearly combines a set of variables into a set of new variables called principal components, which are linearly uncorrelated. In other words, PCA reduces the dimension of the input space by constraining it to the linear subspace with largest sample variation, (Jolliffe, 2002). PCA is applicable for linear regression and neural networks, where the input space stays invariant under linear transformation.

The principal components are ordered in descending component variance. Since PCA retains the largest sample variation, the first component’s loading is

$$ w(1) = \arg \max_{|w|=1} \{||Xw||\} $$

where $X$ is the sample set of the input space. The $k$th component’s loading then becomes the direction of the largest sample variation excluding the previous components, that is

$$ w(k) = \arg \max_{|w|=1} \{||\hat{X}_k w||\} $$

where $\hat{X}_k = X - \sum_{s=1}^{k-1} Xw(s)w_T(s)$.

### 3.4. Data smoothing

To avoid overfitting for general regressions, when the interrelations of input variables are unknown such as SVM, other approaches than the previous are more appropriate, (though there exist techniques for handling this as well). One approach is to smooth the sample data, (Härdle, 2012). This enables for smoothening out sample irregularities distorting the fit. However, excessive smoothing can equally well lead to underfitting.

**Laplacian smoothing**

To smooth, we use the Laplacian smoothing, which shift the positions of the data points towards the average position of their neighbouring points, that is

$$ \hat{x}_i = x_i + \lambda \frac{1}{|N_k(i)|} \sum_{j \in N_k(i)} w_{ij} (x_j - x_i) $$

where $x_i$ and $\hat{x}_i$ are the old respectively new position of point $i$, $N_k(i)$ is the set of $k$:th nearest neighbours of $i$, $0 \leq \lambda \leq 1$ determines the level of smoothing. Furthermore, we use the inverse distance weighting $w_{ij} = \frac{w_{ij}}{\sum_{j \in N_k(i)} w_{ij}}$, where $\bar{w}_{ij} = \frac{1}{|x_i - x_j|}$, such that a higher weight is assigned the closer the neighbour is to the point. Thus, the further a neighbour is from the point relative to the other neighbours, the less smoothening impact they have.
3.5. Local outlier factor (LOF)

To detect outliers, we use the LOF. The LOF measures the deviation of a point from its neighbours compared to the neighbours’ deviation from their respective neighbours, (Breunig, Kriegel, Ng and Sander, 2000). To define it, we first need to define the reachability distance and local reachability. The reachability distance of $A$ and $B$ is defined as

$$\text{reachability\_distance}_k (A, B) = \max \{k\_\text{distance}(B), d(A, B)\}$$

which is the distance of $B$ to $A$ but at least the distance to its $k$th neighbour. The local reachability of a point $A$ can be seen as a measure of the density around $A$ and is defined as

$$\text{lrd}(A) = 1/\left(\frac{\sum_{B\in N_k(A)} \text{reachability\_distance}_k (A, B)}{|N_k(A)|}\right),$$

where $N_k(A)$ is the set of $k$:th nearest neighbours. The LOF is then defined as

$$\text{LOF}_k (A) = \frac{\sum_{B\in N_k(A)} \text{lrd}(B)}{\sum_{B\in N_k(A)} \text{lrd}(A)} / |N_k(A)|,$$

and thus, measures the density around $A$ relative to the densities around its neighbours. In this way, LOF gives a local estimation of a points deviation.

Compared to other outlier detection methods, LOF has the advantage that it assigns a degree of being an outlier. By integrating the LOF in the smoothing process of Section 3.4, we can reduce outliers as well as noise. Particularly, by adjusting the smoothening degree $\lambda$ by the LOF, points can be smoothened with respect to their outlier degree.

3.6. Ensemble Learning

To diversify the risk in a learning algorithm, a set of learners can be combined into an improved learner. This is referred to as ensemble learning, (Zhang and Ma, 2012). More exactly, by considering the correlation of the different predictors, they can be linearly combined to diversify prediction errors, similarly to how assets are combined to diversify risk in MPT. The weights $w = (w_1, ..., w_m)$ distributed between the $m$ predictors in the strategy will then be

$$\arg \min_w wpw^T,$$

where $p$ is the correlation matrix of the prediction errors between the predictors.
4. Methods

The aim of this paper is to develop an as profitable and stable trading algorithm as possible. To achieve this goal, we evaluate different algorithms by backtesting them on historical closing prices. To find the optimal trading algorithms, the algorithms are optimized and tested for different strategies and algorithm parameters, such as number of input variables, regressors and utility functions. This chapter reviews the specification of these algorithms, the backtests and their evaluation procedures.

4.1. Data set

For backtest, we use minute tick data of closing prices, which are the final prices traded on the minute time frames, and spreads from 21:00 of 25 april 2017 to 21:00 of 27 april 2017 for AUD/USD, EUR/USD, GBD/USD, NZD/USD, USD/CAD, USD/CZK, USD/DKK, USD/JPY, USD/MXN, USD/NOK and USD/SEK collected from MetaTrader 5. Here, the currency pairs have been chosen based on that their potential returns exceed the spread costs. Besides these currency pairs to allocate between, we introduce a risk-free asset with a zero risk-free return which envisages unlimited capital available to our disposal.

4.2. Regressors

The regressions evaluated for forecasting the expected returns are linear regression, neural networks and SVM. To train the regressors, L-M is used for neural network and SMO for SVM. Additionally, PCA and data smoothing are applied for linear regression and neural networks, whereas for SVM only data smoothing is applied. For SVM the radial basis kernel is used. In addition to these regressors, the top performable regressors from each category are ensembled and evaluated.

4.3. Training and testing set

The sample data \{(x^1, y^1), ..., (x^{l'}, y^{l'})\} used for predicting the future return \(y^{i+1}\), where \(y^{i+1}\) is the return in the ensuing time step \(y^i\) for all \(i\), is split into two sets, \{(x^1, y^1), ..., (x^l, y^l)\} and \{(x^{l'+1}, y^{l'+1}), ..., (x^{l'}, y^{l'})\}. The first sample set is used for training the regression, that is fitting the regression to the set, see Section 3.2. The second sample set is used for out-of-sample testing the fitted regression, that is comparing the regression’s prediction values to the actual values. The use of this comparison will become apparent in the next section. But one use of it is for estimating the (co)variances used as model parameters in MPT. These are, moreover, estimated from the regressor’s performance on the testing data,
\[ \sigma^2 = \frac{1}{l^* - l} \sum_{i=l+1}^{l^*} (y^i - \hat{y}^i)^2 \]

where \( \hat{y}^i \) and \( y^i \) is the predicted respectively actual return at time \( i \). In our algorithm, we use a sample size of 100 points where the first 70 points are used for training our regressors and the remaining 30 for testing it, that is \( l^* = 100 \) and \( l = 70 \). Moreover, we use a time ordering indexing such that \( y^i \) is the return at time \( i \).

4.4. Feature selection

As stated in Section 3.2, historical closing returns \( r_{t-k+1}, \ldots, r_t \) are used as input variables for estimating future expected returns \( r_{t+1}^e \), that is \( x^t = (r_{t-k}, \ldots, r_{t-1}) \) and \( y^t = r_t \). In our algorithm, the performance optimizing input size \( k \) is selected. The input size \( k \) is determined at each time step as following:

- Given an input size \( k \), fit model to the sample training data.
- Calculate the error (MSE, MAE or MAAPE) of the model predictions to the sample testing data.
- Iterate the previous two steps for different input sizes \( k \).
- Select the model with the minimum performed testing error.

To avoid overfitting, the \( k \) values are bounded by the sample training size, see Section 3.3. Furthermore, the \( k \)'s are chosen such that higher \( k \) values appear less frequent, because regressors with larger input sizes differ less than small ones. Also, the set and set size of the \( k \)'s are restricted by the regressors computational training time. Specifically, for neural networks this set is chosen as \( \{10, 20, 30, 40\} \) with the elements in \( \{7, 13, 20, 27\} \) being the respective number of neurons in the hidden layer. For SVM regression, the set is instead chosen as \( \{1, 5, 10, 15, 20, 30\} \). For linear regression, the set is chosen as \( \{1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 40\} \). Also, the sample size applying PCA on is chosen to 100.

4.5. Data transformation

Since the regressions fit only with respect to MSE, see Section 4.8, it need to be addressed when intending to fit with the other error estimates. For MAE, we overlooked this, due to MAE already being similar to MSE. However, for MAAPE, we modify the sample set such that the impact of the sample points increases the smaller their absolute \( y \)-values are, as accounted by MAAPE. This is accomplished by resampling each \( y^i \), round \( \left( l \frac{\max(|y^1|, \ldots, |y^l|) - |y^i|}{\max(|y^1|, \ldots, |y^l|)} \right) \) number of times. This rule resamples a sample point by the distance of its absolute \( y \)-value to the
maximum absolute $y$-value in the sample set, with the distance scaled by $l/\max(|y^1|, ..., |y^l|)$ and rounded off to the closest integer. Thus, each sample point is resampled between zero and $l$ times, where a point with a zero $y$-value is resampled $l$ times whereas one with a maximum absolute $y$-value is resampled zero times. Here, $l$ is chosen as 70, the sample length for training, 100 minus 30. For SVM regression, we instead transform the sample data according to $y := \frac{y}{|y|} \sqrt{|y|}$, which after the fit is transformed back. Due to the transformation’s diminishing derivative, it decreases the differences between the $y$-values, the larger the absolute $y$-values are, making smaller absolute $y$-values more sensitive and larger absolute $y$-values less sensitive for the fit. This transformation is valid for SVM because SVM makes, compared to neural network and linear regression, unconstrained fits. Thus, the transformations do not intervene the end result anymore than intended (as long as the transformation and its inverse is one-to-one).

When data smoothing is applied on the sample set $\{(x^1, y^1), ..., (x^l, y^l)\}$, we use the smoothening degree $\lambda = \arctan(\text{LOF}_k - 1) / (\frac{\pi}{2})$. This smoothening rule is chosen because it enables no smoothening when the point is not considered an outlier, that is $\lambda = 0$ when $\text{LOF}_k \leq 1$; higher smoothening the higher the outlier degree, $\text{LOF}_k$, that is $\lambda'(\text{LOF}_k) \geq 0$; while restricting the smoothed point to between its original position and the positions of the neighbours, that is $0 \leq \lambda \leq 1$ for all $\text{LOF}_k$. In the smoothening, the number $k$ is chosen equal to the number of points spanning a simplex in the feature space, which is one plus the dimension of the input-output space.

### 4.6. Utility function

As motivated in Section 2.2, we adjust leverage by the utility function

$$U(r^e, \sigma) = (r^e)^a - c\sigma^b$$

where $a$, $b$ and $c$ are constants. Here, we conduct a common test for parameter values $a = 0.8$, $b = 1.2$ and $c = 7$, where the choice of the $a$ and $b$ values were motivated in Section 2.2 and $c$ is chosen such that it scales $\sigma$ to $r^e$ in $U(r^e, \sigma)$. Unfortunately, biases in the model variables can impair the utility to adjust leverage desirably. By adjusting the utility parameters, however, the impact of these biases may be reduced. Thus, in addition to conducting the common test, we, from the vantage point of the common test, experiment and evaluate the backtest performance for different fixed utility parameters.
4.7. Performance measures

Backtest performances are assessed by their profitability and stability, which in turn we measure by the profitability ratio $PR$ and the variance of this estimate. To correctly account the profitability, returns need to be evaluated relative to their expected, because of leverage scalability. In addition, since the utility scales the returns, the returns need no further normalizing for correct performance evaluation. The profitability can thus be seen as measuring the reliability of the long run performance, while the stability can be seen as measuring the fluctuations around the long run performance. Since the ordering of the returns matters for stability, the profitability is, further, assessed by the cumulative returns rather than the returns separately. Indeed, if the underperformed and overperformed returns were clustered than spread over time, performance would be less smooth and thus less stable, because of inflated performance swings. In conclusion, $PR$ is defined as the ratio which scales the cumulative returns $CR_t$’s to their expected while minimizing the variance between them, where $CR_t = \sum_{i=1}^{t} R_i$ and $R_i$ is the return at time $i$. Here, the reason for minimizing variance is to ensure that the obtained ratio is the one which most accurately scales the performed profits to the expected.

The resulting variance from this calculation gives in turn an estimate of the stability of the strategy. To ensure transparent variance estimate between different cumulative return scales, the cumulative returns are, further, normalized by the average cumulative return $N = \sum_{t=1}^{T} E[CR_t] / T$ where $T$ is the time length of the backtest. The profitability ratio $PR$, thus, computes as

$$PR = \arg \min_{\alpha} \left\{ \sum_{t=1}^{T} \left( \frac{\alpha CR_t - E[CR_t]}{N} \right)^2 \right\},$$

and the stability as

$$V = \sum_{t=1}^{T} \left( \frac{PR \cdot CR_t - E[CR_t]}{N} \right)^2.$$

Here, optimal profitability would correspond to $PR = 1$, which means that the average performance aligns with the expected. A higher $PR$ would, on the other hand, correspond to a lower profitability, and the higher the $PR$, the lower the profitability. As an additional measure of stability, we use the kurtosis

$$Kurt = \frac{\sum_{t=1}^{T} \left( \frac{PR \cdot CR_t - E[CR_t]}{N} \right)^4}{V^2},$$
which has the advantage of detecting large performance swings not properly captured by the variance estimate. Finally, for these measures to be valid, $PR$ is required to be estimated positive (that is the cumulative return needs mainly to stay positive).

### 4.8. Implementation

The algorithms are implemented in and run on MATLAB. There, MATLAB’s built-in functions fitrsvm, fitnet and regress are used for SVM, neural network respectively linear regression. For optimization purposes, such as finding optimal portfolio in a transaction cost setting, fixing broken correlation matrices and finding the profitability ratio minimizing variance, we use the MATLAB built-in function patternsearch. For optimizing over input size and finding optimal portfolio with transaction cost excluded, ordinary grid search is applied respectively the linear matrix Equation (1) in Section 2.1 is solved analytically. Because transaction costs disincentivize portfolio adjustments, we also upper and lower bound the allowed asset weights in the portfolios to avoid confining future portfolio choices to local optimums, see Equation (2) in Section 2.1. For the MATLAB functions, we use default settings unless specified.
5. Result

This chapter presents the result of our backtest on the historical data. The results, first overviewed in table form, are presented in the sections following.

The following tables collect the performances of the different learners. Here, the estimated values in Table 1 have their utilities adjusted for better performance.

<table>
<thead>
<tr>
<th>(PR; V; Kurt)</th>
<th>MSE</th>
<th>MAE</th>
<th>MAAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear regression -</td>
<td>(23.31; 0.47; 2.17)</td>
<td>(26.74; 0.80; 1.48)</td>
<td>-</td>
</tr>
<tr>
<td>Smoothing</td>
<td>(17.51; 0.36; 2.25)</td>
<td>(40.58; 0.70; 2.22)</td>
<td>-</td>
</tr>
<tr>
<td>PCA</td>
<td>(12.24; 0.37; 2.99)</td>
<td>(4.11; 0.65; 1.87)</td>
<td>-</td>
</tr>
<tr>
<td>Smoothing &amp; PCA</td>
<td>(29.37; 0.83; 2.15)</td>
<td>(7.19; 0.37; 1.36)</td>
<td>-</td>
</tr>
<tr>
<td>Neural network -</td>
<td>(0.10; 0.97; 3.39)</td>
<td>(74.01; 0.35; 2.79)</td>
<td>(282.41; 1.01; 1.99)</td>
</tr>
<tr>
<td>Smoothing</td>
<td>(0.12; 0.84; 2.51)</td>
<td>(0.07; 0.34; 1.99)</td>
<td>(136.99; 0.44; 3.88)</td>
</tr>
<tr>
<td>PCA</td>
<td>(147.40; 0.83; 2.05)</td>
<td>(120.55; 0.72; 3.39)</td>
<td>(57.98; 0.44; 1.99)</td>
</tr>
<tr>
<td>Smoothing &amp; PCA</td>
<td>(93.05; 0.69; 2.51)</td>
<td>(60.97; 0.22; 2.32)</td>
<td>(108.44; 0.57; 2.45)</td>
</tr>
<tr>
<td>SVM -</td>
<td>(70.38; 0.98; 2.48)</td>
<td>(20.78; 0.64; 2.28)</td>
<td>(6.59; 0.16; 1.82)</td>
</tr>
<tr>
<td>Smoothing</td>
<td>(26.39; 0.69; 3.33)</td>
<td>(9.81; 0.30; 2.46)</td>
<td>(12.11; 0.26; 3.37)</td>
</tr>
<tr>
<td>Ensemble (with and without smoothing) -</td>
<td>-</td>
<td>-</td>
<td>(13.15; 0.34; 3.76)</td>
</tr>
</tbody>
</table>

Table 1. Estimated performance of the different learners when excluding transaction costs.
Table 2. Estimated performance of the different learners when including transaction costs.

The following figures display the aggregate plots of all the trading performances (with and without transaction costs) for the utility parameters $a = 0.8$, $b = 1.2$, $A = 7$ and $B = 1$. A better overview and in-detail analysis of the plots in Figure 5.1 are presented in the Section 5.1, 5.2 and 5.3. The axis units in the figures and in all the forthcoming ones are percent on the $y$-axis and minutes on the $x$-axis.
Figure 5.1. Trading performance without transaction costs.
5.1. Linear regression

This section displays the in-detail results of the trading performances of the linear regression when transaction costs were excluded.

Figure 5.1.1 displays the aggregate plot of the trading performances of linear regressions with utility parameters $a = 0.8$ and $b = 1.2$. It shows none of the learners to be profitable. Disregarding this fact, PCA irrespective to smoothing or the used error estimate gave best performance. The breakdown of the learners around the time 1500 in Figure 5.1.1, hint to the shortcome of using a linear regression model for return prediction.
Figure 5.1.1. Trading performance for linear regression without transaction costs.
The following figure displays the cumulative return when linear regression with MSE was used. Best performance was with $a = 0.8$ and $b = 2$, giving $PR = 23.31$, $V = 0.47$ and $Kurt = 2.17$. This result indicates that risk adjusted leverage brings stability.

**Figure 5.1.2. Trading performance for linear regression with MSE.**

Applying data smoothing brought further stability. Moreover, risk adjusted parameters $a = 0.5$ and $b = 3$ gave best performance with $PR = 17.51$, $V = 0.36$, and $Kurt = 2.25$. 
Figure 5.1.3. Trading performance for linear regression with data smoothing and MSE.

For PCA with MSE, $a = 2$ and $b = 1$ gave best performance, with $PR = 7.70$, $V = 0.35$ and $Kurt = 2.60$. This favours instead a return maximizing leverage.

![Figure 5.1.3.](image)

Figure 5.1.4. Trading performance for linear regression with PCA and MSE.

On the other hand, the combination data smoothing and PCA proved unsuccessful. In that case, only constant expected returns were able to perform, which gave $PR = 29.37$, $V = 0.83$ and $Kurt = 2.15$, a declined performance from previous ones.

![Figure 5.1.4.](image)
Switching MSE for MAE produced different result, with best result with the risk adjusted leverage $a = 0.5$ and $b = 3$ which gave $PR = 26.74$, $V = 0.80$ and $Kurt = 1.48$.

Similarly, adding smoothing improved stability with $a = 0.5$ and $b = 3$ giving $PR = 40.58$, $V = 0.70$ and $Kurt = 2.22$. 

Figure 5.1.5. Trading performance for linear regression with data smoothing, PCA and MSE.

Figure 5.1.6. Trading performance for linear regression with MAE.
Applying PCA, however, gave less success. Return preferring parameters \( a = 1.5 \) and \( b = 1 \) gave best performance with \( PR = 4.11, V = 0.65 \) and \( Kurt = 1.87 \).

Adding smoothing, on the other hand, were able to remedy these failings. Return preferring parameters \( a = 1.5 \) and \( b = 1 \) gave \( PR = 7.19, V = 0.37 \) and \( Kurt = 1.36 \).
Figure 5.1.9. Trading performance for linear regression with data smoothing, PCA and MAE.

Compared to MSE and MAE, MAAPE failed to perform. Therefore, performance presentation for this has been skipped.

5.2. Neural Network

This section displays the in-detail results of the trading performances of the neural network when transaction costs were excluded.

The figure below displays the aggregate plot of the trading performances for neural network with utility parameters $a = 0.8$ and $b = 1.2$. It indicates highly fluctuating performances for the regressors.
Figure S2.1. Trading performance for neural network without transaction costs.
MSE with risk aversive parameter choices $a = 1$ and $b = 2$ gave $a = 0.10$, $a = 0.97$ and $a = 3.39$.

![Figure 5.2.2. Trading performance for neural network with MSE.](image1)

Applying data smoothing, with risk aversive parameters $a = 1$ and $b = 2$ gave best performance with $PR = 0.12$, $V = 0.84$ and $Kurt = 2.51$.

![Figure 5.2.3. Trading performance for neural network with data smoothing and MSE.](image2)
For PCA with MSE, $a = 1$ and $b = 2$ gave best performance, with $PR = 147.40$, $V = 0.83$ and $Kurt = 2.05$.

![Figure 5.2.4. Trading performance for neural network with PCA and MSE.](image)

The combination PCA and data smoothing with MSE gave, with risk aversive parameters $a = 1$ and $b = 2$, $PR = 93.05$, $V = 0.69$ and $Kurt = 2.51$.

![Figure 5.2.5. Trading performance for neural network with data smoothing, PCA and MSE.](image)
For MAE, risk aversive parameters \( a = 0.8 \) and \( b = 1.2 \) gave best performance, giving \( PR = 74.01 \), \( V = 0.35 \) and \( Kurt = 2.79 \).

![Trading performance for neural network with MAE.](image)

**Figure 5.2.6.** Trading performance for neural network with MAE.

For MAE with data smoothing, \( a = 1 \) and \( b = 1.5 \) gave \( PR = 0.07 \), \( V = 0.34 \) and \( Kurt = 1.99 \).
For MAE with PCA, risk aversive parameter choices $a = 0.5$ and $b = 5$ gave $PR = 120.55$, $V = 0.72$ and $Kurt = 3.39$.

For MAE with PCA and data smoothing, risk aversive parameter choices $a = 1$ and $a = 2$ gave $PR = 60.97$, $V = 0.22$ and $Kurt = 2.32$. 
Figure 5.2.9. Trading performance for neural network with data smoothing, PCA and MAE.

For MAAPE, risk aversive parameter choices $a = 0.5$ and $b = 5$ gave $PR = 282.41$, $V = 1.01$ and $Kurt = 1.99$.

Figure 5.2.10. Trading performance for neural network with MAAPE.

For MAAPE with data smoothing, risk aversive parameter choices $a = 1$ and $a = 1.5$ gave $PR = 136.99$, $V = 0.44$ and $Kurt = 3.88$. 
Figure 5.2.11. Trading performance for neural network with data smoothing and MAAPE.

For MAAPE with PCA, risk aversive parameter choices $a = 0.5$ and $b = 1.5$ gave $PR = 57.98$, $V = 0.44$ and $Kurt = 1.99$.

Figure 5.2.12. Trading performance for neural network with PCA and MAAPE.

For MAAPE with data smoothing and PCA, risk aversive parameter choices $a = 0.8$ and $b = 1.2$ gave $PR = 108.44$, $V = 0.57$ and $Kurt = 2.45$. 
5.3. SVM

This section displays the in-detail results of the trading performances of the SVM when transaction costs were excluded.

The figure below displays the aggregate plot of the trading performance for SVM with utility parameters $a = 0.8$ and $b = 1.2$. Here, the MAAPE and MAAPE Smooth and the ensemble of them seem to produce the best performances.
Figure 5.3.1. Trading performance for SVM without transaction costs.
For MSE with risk aversive parameter choices $a = 0.5$ and $b = 5$ gave $PR = 70.38$, $V = 0.98$ and $Kurt = 2.48$.

Figure 5.3.2. Trading performance for SVM with MSE.

Applying data smoothing with risk aversive parameters $a = 1$ and $b = 2$ gave $PR = 26.39$, $V = 0.69$ and $Kurt = 3.33$.

Figure 5.3.3. Trading performance for SVM with data smoothing and MSE.
Replacing MSE with MAE, with risk aversive parameters $a = 1$ and $b = 2$ gave $PR = 20.78$, $V = 0.64$ and $Kurt = 2.28$.

Figure 5.3.4. Trading performance for SVM with MAE.

Applying smoothing, with risk aversive parameters $a = 1$ and $b = 1.5$ gave $PR = 9.81$, $V = 0.30$ and $Kurt = 2.46$.

Figure 5.3.5. Trading performance for SVM with data smoothing and MAE.
MAAPE with risk aversive parameters $a = 1$ and $b = 1.5$ gave $PR = 6.59$, $V = 0.16$ and $Kurt = 1.82$.

Figure 5.3.6. Trading performance for SVM with MAAPE.

Employing data smoothing with risk aversive parameters $a = 0.5$ and $b = 5$ gave $a = 12.11$, $a = 0.26$ and $a = 3.37$, a deterioration from previous.

Figure 5.3.7. Trading performance for SVM with data smoothing and MAAPE.
The ensemble of SVM MAAPE with and without smoothing with risk aversive parameters $a = 0.1$ and $b = 10$ gave $PR = 13.15$, $V = 0.34$ and $Kurt = 3.76$, a slight underperformance from the individual regressors.

![Figure 5.3.8. Trading performance for ensemble of SVM with MAAPE and with and without data smoothing.](image)

**5.4. With transaction costs**

This section displays the in-detail results of the trading performances in a transaction costs setting.

When transaction costs were included, only SVM with MAAPE performed. An example of the other learner’s performance is displayed in the figure below; in this case, it displays the performance for linear regression with MSE.
For SVM MAAPE, the table and figures show that the profit maximizing strategy outperforms the risk averse strategy.

<table>
<thead>
<tr>
<th>Regression</th>
<th>Utility parameters</th>
<th>(PR; V; Kurt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM MAAPE</td>
<td>$\alpha = 1, b = 1, A = 10$ and $B = 1$</td>
<td>(2.19; 0.39; 1.91)</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 1, b = 0, A = 1$ and $B = 0$</td>
<td>(1.68; 0.40; 2.43)</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 1, b = 2, A = 500$ and $B = 1$</td>
<td>(2.05; 0.57; 2.22)</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 0.8, b = 1.2, A = 7$ and $B = 1$</td>
<td>(3.77; 0.84; 3.11)</td>
</tr>
</tbody>
</table>

Table 5.4.1. Estimated performance for SVM MAAPE with different utility parameters when including transaction costs.
Figure 5.4.2. Trading performance for SVM with MAAPE.

Figure 5.4.3. Trading performance for SVM with MAAPE.
Including smoothing with the same parameter combinations gave some further improvement. The table and figures show, similarly as the previous predictor performance, that the profit maximizing strategy surpass the risk averse strategy.

<p>| Regression | Utility parameters | ((PR; V; Kurt)) |</p>
<table>
<thead>
<tr>
<th>SVM MAAPE Smooth</th>
<th>$\alpha = 1, \beta = 1, A = 10$ and $B = 1$</th>
<th>(2.71; 0.48; 1.89)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha = 1, \beta = 0, A = 1$ and $B = 0$</td>
<td>(1.33; 0.30; 1.46)</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 1, \beta = 2, A = 500$ and $B = 1$</td>
<td>(1.74; 0.35; 2.17)</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 0.8, \beta = 1.2, A = 7$ and $B = 1$</td>
<td>(1.90; 0.70; 2.71)</td>
</tr>
</tbody>
</table>

Table 5.4.2. Estimated performance for SVM MAAPE Smooth with different utility parameters when including transaction costs.

![Figure 5.4.6. Trading performance for SVM with data smoothing and MAAPE.](image-url)
Figure 5.4.7. Trading performance for SVM with data smoothing and MAAPE.

Figure 5.4.8. Trading performance for SVM with data smoothing and MAAPE.
Figure 5.4.9. Trading performance for SVM with data smoothing and MAAPE.

Ensemble of SVM MAAPE and SVM MAAPE Smooth were, however, unprofitable for the standard test $a = 0.8$, $b = 1.2$, $A = 7.558$ and $B = 1$. Compared to previous predictor performances, no difference between strategies were here displayed.

<table>
<thead>
<tr>
<th>Regression</th>
<th>Utility parameters</th>
<th>$(PR; V; Kurt)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensemble of SVM MAAPE and SVM MAAPE Smooth</td>
<td>$a = 1$, $b = 1$, $A = 10$ and $B = 1$</td>
<td>(1.40; 0.66; 3.49)</td>
</tr>
<tr>
<td></td>
<td>$a = 1$, $b = 0$, $A = 1$ and $B = 0$</td>
<td>(1.47; 0.48; 2.21)</td>
</tr>
<tr>
<td></td>
<td>$a = 1$, $b = 2$, $A = 500$ and $B = 1$</td>
<td>(1.41; 0.47; 2.19)</td>
</tr>
<tr>
<td></td>
<td>$a = 0.8$, $b = 1.2$, $A = 7$ and $B = 1$</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.4.3. Estimated performance for ensemble of SVM MAAPE and SVM MAAPE Smooth with different utility parameters when including transaction costs.
Figure 5.4.10. Trading performance for ensemble.

Figure 5.4.11. Trading performance for ensemble.
Figure 5.4.12. Trading performance for ensemble.

Figure 5.4.13. Trading performance for ensemble.
6. Discussions and conclusions

This chapter analyses and discusses the result of the backtest from the previous chapter.

From the backtest results, when disregarding transaction costs, parameter values of the utility required adjusting between the learners for the learners to be profitable. The explanation for this may be due to model biases between the learners. For example, the learners may have biases in their variances due to different forecasting accuracies. Thus, adjusting the parameters can partly smooth out these effects. Specifically, a lower exponent value of a variable in Equation 2 can lessen the impact of its bias. Conversely, a higher exponent value can instead disperse dampened effects. In any case, adopting a time-varying leverage improved performance and profitability. On another note, these adjustments may be unjustified since they may equally well contribute to the curse of dimensionality and/or in-sample error. To validate these adjustments, extended out of sample testing need to be accompanied. However, these results will not significantly change the conclusions otherwise. Therefore, the results are taken as additional support in the coming analysis.

Considering adjusted utility parameters with transaction costs excluded, most learners were profitable, as can be seen in Table 5.1 and the figures in Sections 5.1, 5.2 and 5.3. However, the same could not be said when the utility parameters were left unadjusted. The setting without transaction costs serves a good background for testing the different methods and strategies. Including transaction costs, on the other hand, rendered most learners unprofitable. This indicates the requirement for higher forecasting precision in the setting of spreads.

The results in Table 5.1 and Figures 5.2, 5.1.1 and 5.3.1 indicate that smoothing overall improved stability and profitability. The effect on the kurtosis were, however, ambiguous. Similarly, PCA improved stability and profitability, while leaving the effect on the kurtosis ambiguous, according to Table 5.1 and Figure 5.1.1. The combination of smoothing and PCA gave, however, ambiguous effect though with more inclination towards improvement. On the other hand, the combination was successful to reduce the kurtosis.

According to Tables 5.1 and 5.2 and Figures 5.2 and 5.1.1, the MAAPE for SVM (with and without smoothing) outperformed the other error estimates, both in terms of stability and profitability. That MAAPE outperforms demonstrates the importance of scaled error estimates for performance. This is reasonable since errors get scaled by leverage and/or the asset weights in the end, making the ratios only relevant. Additionally, it indicates that a trading strategy is more sensitive to fluctuations around small returns than large, since then the likelihood for
losses are higher. In contrast to SVM, this estimate failed for linear regression. This is probably due to the constrain in the linear fit making the higher weights on the smaller values, as is the case with MAAPE, impair the global fit. This backdrop is, however, not shared by SVM, due to its higher degree of freedom, which is why MAAPE works for it. Otherwise, considering the other error estimates, MSE and MAE, no significant differences between their performances were detected.

With transaction costs included, only the combination MAAPE and SVM (with and without smoothing) were profitable, see Table 5.2 and Figure 5.2. Furthermore, smoothing improved its performance further. The reason for the successes of SVM is likely due to its ability to fit nonlinearly, resulting in a higher fitting accuracy and better performance.

Compared to linear regression and SVM, Neural network demonstrated inconsistent and unstable performances. This is indicated by its fluctuating performances which does not align with the performances of the other regressors and theory. Specifically, its profitability ratio varies from well below one to well above those of the other regressors. A profitability ratio below one would indicate a high instability since an equal possibility exists for a comparable move in opposite direction. It is hard to pinpoint what causes these instabilities. But one explanation may be due to the randomness inherit in the L-M method. To fix this problem, one could extend the random sample training sets. Though this would bring stability, it would come at the cost of speed. On another note, using the L-M method may not do justice to neural network since L-M is only able to find the local optima. However, since L-M is the fastest and most efficient method and since speed is critical for trading at high frequency, there leaves little choice. The drawdown in speed applies as well to the previous suggestion. In conclusion, the cost of any of the proposed remedies may outweigh their benefits. The computational demand of neural network may further restrict the possibilities for improving upon the learner, making neural networks an unwise regression choice for high-frequency trading in our case.

Another unexpected finding was that the ensemble of the two top performed regressors performed below than the regressors individually, see Table 5.2 and Figure 5.2. A possible explanation for this may be due to inefficient diversification of the regressors, which in turn could have been due to modest sample size used. A more likely explanation, however, may be due to model bias differences which when compounded cause erratic model estimations.

Moving to the strategy aspect of our algorithm, the setting without transaction costs demonstrated the success for risk diversification to generate stability. However, the same
proved detrimental when spreads were included. There, instead, the profit maximizing strategy proved most effective. This suggests the need for high prediction accuracy to succeed in the higher trading frequencies, due to the impact of the spread. It also suggests that big gains at the cost of big losses are more preferred than their respective small ones. This is reasonable, since then the spread to return ratio diminishes, everything else equal, lessening the effect of the spread on the profit. However, that the profit-maximizing strategy surpasses the risk diversifying strategy may not be final. It could be that the spreads of most assets have rendered risk diversification ineffective, within the tested forecasting precision. Thus, the finding may just express a required minimum in forecasting precision.

In summary, we found the combination of SVM and MAAPE critical for profitability (in high-frequency setting). Furthermore, smoothing with LOF had varied effect, but on the overall it seemed to improve performance (as did PCA). Ensembling, though performed, demonstrated less success than the individual regressors; this could be due to the drawdown of increased model complexity.
7. References


Hsu, Ming-Wei; Lessmann, Stefan; Sung, Ming-Chien; Ma, Tiejun; Johnson, Johnnie E.V.. 2016. *Bridging the divide in financial market forecasting: machine learners vs. financial economists*. Expert Systems with Applications 61: 215-234.


