Masters Thesis in Computer Science

at Freie Universität Berlin

Biorobotics Working Group

Development of a Multi-Agent AI Framework for Autonomous FOREX Trading

David Dormagen
david [dot] dormagen [at] fu-berlin.de

Supervisor: Prof. Dr. Tim Landgraf

Berlin, 17.10.2016

Abstract

In this thesis, a C++ framework for automatic trading on the foreign exchange market (FOREX) is developed. The framework allows an ensemble of prediction models to either run on a live market, handling the communication with the broker and the execution of trades; or to be evaluated on historical data through a virtual market simulation. The solution provides a ZeroMQ messaging interface to other languages such as Python; this allows for rapid prototyping of new prediction models through the utilization of the extensive machine learning ecosystem of Python.

In addition to the architecture, the implementation of some example agents that provide different feature transformations of the exchange rate is described. The possibility of dependencies among the agents together with a careful data handling allows for multiple iterations of training of the classifiers while making sure that no information is leaked in the process.

Contents

1 Introduction

This section will give an overview about the problem setting and the goals this thesis achieves.

In later chapters, this thesis will assume some prior knowledge of certain key concepts including, but not limited to, how FOREX trading works for the end-user (the trader), how prediction models generally approach regression or classification problems, and how certain key metrics work that are used to evaluate the performance of models.

In order to reduce the need to explain basic concepts during the later chapters, this section introduces some knowledge that will allow reading through the later chapters more fluently. Readers who are already familiar with the concepts presented here, might want to skip the section or skim through it.

This section will start by introducing the FOREX market for currency trading, giving some insights of how trading and predictions work in general. Then it will look at the problem from a machine-learning point of view and transform the problem into a classical machine learning problem. It will show how prediction or classification work in general and then introduce a few models that are used in later chapters.

Afterwards some common metrics to evaluate the performance of the predictions will be explained, which will be applied to this problem in a later chapter.

Concepts that are only used in certain areas of the thesis or that are limited in scope are introduced during the chapter where they are required for understanding.

1.1 Foreign Exchange Market

The foreign exchange market, in short *FOREX market*, is an instrument to trade currencies. Next to the stock market for goods, it is one of the biggest trading instruments. The Bank for International Settlements (2007) reported an average daily turnover of $3.2 trillion in 2007, having risen by 60% to 70% from 2004. In 2013, the turnover had risen to $5.3 trillion (Bank of International Settlements (2013)).

The FOREX market is split up into pairs of two currencies that can be traded against each other. The currencies that were traded with the highest volume in 2013 were the US dollar with being part of around 90% of all trades and the euro following with around 30% (Bank of International Settlements (2013)).

At any point a participant in the market, a *trader*, can decide to bet on one side of a currency pair. Depending on whether the selected currency is the first or the second of the pair, the trade is said to be a *buy* trade or a *sell* trade. A *buy* trade (or *long position*) bets on the first currency of a pair gaining value relative to the second currency. Vice-versa, a trader opening a *sell* trade (or *short position*) expects the first currency of the pair to decrease in value relative to the second.

After having opened the trade, a trader can close it again manually or by using automatic thresholds in either direction. The difference in the rate between the time points of opening and closing the trade will either be credited to or subtracted from the trader's account.

In reality this process involves depositing a certain amount of money in a trading account at a so-called *broker*, which will execute your trades for you. When a trader opens a trade, she lends a chosen amount of money in either the first currency or the second currency from the broker and when the trade is closed again, any difference between the originally lent amount and the new value will need to be either payed to the trader by the broker (in case the lent money gained value) or to the broker by the trader, and thus be subtracted from her trading account.

Thus, because a trader does not directly buy or exchange currencies and only bets on changing exchange rates, it is possible to trade higher amounts than the trader would usually be able to and to trade with substantial potential profit and high risk even with small amounts of money.

1.1.1 Trading strategies and market theory

A participant in the market, who is willing to execute trades, has to decide on the exact moment of when to open or close trades. There are a few different main categories of strategies to motivate a trader's action in the market. The most intuitive way to predicting future exchange rates is called fundamental analysis. Practitioners search for an explanation behind the current market behavior by looking into other aspects of the market, such as unemployment rates or political events (for a concise list of fundamental factors affecting exchange rates, see Patel et al. (2014)).

A different approach is taken by those who practice the so called *technical analysis*. Technical analysis focuses solely on the past exchange rate, assuming that past behavior of the market contains information that has not been included in the current exchange rate.

Classically, technical analysis involves finding visual patterns in a graph of the exchange rate from which the trader hopes to be able to extrapolate future price movement.

Allen and Taylor (1990) found that for short time horizons of below a day to one week, around 90% of the traders incorporate technical analysis into their decisions with 60% considering it more important than fundamental analysis. They found that technical analysis is employed less as the time horizon of the prediction increases. At a prediction horizon of more than a year fundamental analysis is considered to be more important than technical analysis by about 85% of the traders.

Lui and Mole (1998) could reproduce this trend a few years later on a different market.

Arguing against the sensibility of this trend, critics of technical analysis claim that the past movement of the exchange ratio of a currency pair does not contain any information about the future movement of this ratio. This goes along with the *efficient market hypothesis* introduced by Fama (1970) that claims that all available information that could drive the exchange rate has already been included in the market at any point in time by the different participants, each adding different pieces of information, making the (FOREX) market an efficient representation of the other fundamental information.

In such an efficient market, the past exchange rate would carry no information that would allow technical analysis to be profitable. However, there is evidence that historical prices might have an influence on future price movement - even if just through the irrationality of market participants. For example, Chang and Osler (1999) found that one of the common visual patterns in technical analysis, the head-and-shoulders pattern, is dominated by far simpler rules which implies that one of the methods employed by technical analysts can introduce irrationality and imperfection into the market.

As another point against the effectiveness of trading rules inferred from the past, Fildes and Makridakis (1995) notice that all evidence hints to the properties of economical time-series changing over time; and thus trying to predict future movements in such time-series with the knowledge of the changing, non-stationary behavior seems paradoxical. Only with the assumption of the existence of elements that are persistent over time, a prediction can be attempted.

This notion of humans' irrationality and changing characteristics goes along with the *adaptive market hypothesis*, which was introduced by Lo (2004). It states that there might be short-lived imperfections in the market, caused e.g. by fear and greed of the market participants, to which the market as a whole adapts over time. Those imperfections might however allow for unusual profit without unusually high risk while they persist.

1.2 Prediction models

In the context of this thesis, a *prediction model* or a *learner* is generally any function $\Phi(\vec{x})$ that maps any number of observations or *features* $\vec{x}$ to an output, a prediction, $\hat{y}$. Depending on the actual model, the input can either be only numerical (e.g. 1, 5.23, 100) or even categorical (e.g. "buy", "sell"). The output of Φ, $\hat{y}$, is the prediction of some unknown value and generally improves in quality (see subsubsection 1.2.1) the more data was available when deriving Φ. The process of deriving the function Φ is called the learning or *training* while the process of applying Φ to some values $\vec{x}$, that were possibly never seen during the learning phase, is called the prediction.

1.2.1 Measures of performance

Given two different prediction models, it is often necessary to compare their performance. Thus it is necessary to evaluate the performance of a model numerically. There are many typical error measures that have also been applied to financial time-series forecasting. Generally, the available error measures have to be sorted into two categories: measures for *regression* (i.e. the prediction of some continuous value) and measures for *classification* (i.e. the prediction of a category). As both types of prediction models are used throughout this thesis, a short overview over both will be given.

For regression, a common error measure is the *mean squared error* (or the *root mean squared error*), defined as:

$$MSE = \frac{\sum_{i=0}^{N}(y_i - \hat{y}_i)^2}{N},\ RMSE = \sqrt{MSE}$$

with $\hat{y}$ being the prediction of a model for some target value and y being the true value; the MSE is thus a measure of the arithmetical difference of the predictions and the true values. The MSE has known limitations for comparisons between different data sets (Clements and Hendry (1993), Armstrong and Fildes (1995)) as it is not invariant regarding the range of values: the actual meaning of the specific value of the MSE for the regression quality depends on the dataset and needs a comparison value to make sense. Thus, a normalized version of the MSE will be used when giving experimental results later, defined as:

$$MSE_{normalized} = \frac{MSE}{MSE_{baseline}}$$

with $MSE_{baseline}$being the mean squared error of a comparison prediction on the same dataset; it can for example be a simple linear regression or

even just the mean or mode value of the target values in the training set. A value of the normalized MSE below 1.0 indicates a prediction quality better than the baseline model, as we aim to reduce the error. Comparing with a simple baseline model can be especially valuable if it is not clear a-priori that a prediction better than random can be made for a certain problem, which would imply a constant prediction of the mean value of the training target as the optimal prediction. A similar normalization can be found in the evaluation of the Santa Fe time-series competition described in Gershenfeld et al. (1993).

Another typical measure in time-series prediction is the *ratio of correctly predicted signs* sometimes also called the *hit ratio*, defined as:

$$hit\,ratio = \frac{\#correctly\,predicted\,upward\,movements + \#correctly\,predicted\,downward\,movements}{\#all\,predictions}$$

This might be more desirable compared to the mean squared error, as the important factor when deciding profit or loss is the direction of the exchange rate movement, regardless of the magnitude of the change; this is due to the direction inducing a specific action of the trader (selling or buying). Thus, the problem of exchange rate prediction can be directly understood as a classification problem - in the above case with the two classes (i.e. "direction is positive", "direction is negative").

Generalizing this to multiple classes (e.g. an additional class for "direction will stay the same") leads us to the notion of the *accuracy* for classification problems, defined as:

$$accuracy = \frac{\#correct\,predictions}{\#all\,predictions}$$

Or, more formally, with $\#(i,j)$ being the number of times that a prediction of class i was made for a true class of j and C being the set of all classes:

$$accuracy = \frac{\sum_i^C \#(i,i)}{\sum_i^C \sum_j^C \#(i,j)}$$

The accuracy is 1 if all predictions are correct and 0 if all predictions are wrong; it can be understood as the ratio of correct predictions, regardless of the type of error. However, in the case of FOREX predictions, the type of error might play an important role: a possible "direction will stay the same" or "no action" prediction can never lead to loss or profit as it triggers no action from the trader; thus the consequences of an actual move predicted as "no action" might be negligible - though vice-versa not so much.

A metric used throughout this thesis will be a weighted version of the accuracy. Given weights $w(i,j)$ for a prediction of class i when the true class is j, and $\#(i,j)$ as the number of predictions of class i as class j, it is defined as:

$$weighted\,accuracy = \frac{\sum_i^C \sum_j^C \#(i,j) \times w(i,j)}{\sum_i^C \sum_j^C \#(i,j)}$$

From this, the typical definition of accuracy can be obtained by setting w to the Kronecker delta $w(i,j) = \delta_{ij} = \begin{cases} 1 & if\, i = j \\ 0 & otherwise \end{cases}$.

By setting w to the expected payoff of the action that the prediction would trigger, we get the *mean profit*. However, in the presence of an action that leads to no profit or loss (i.e. the class "no action"), the mean profit can easily be optimized by making only a few (e.g. just one) correct predictions and predicting the remaining samples to be in the category "no action". While such a safe prediction might be desirable, it is also desirable to make as many (correct) actions as possible for statistical significance; thus the metric used will be the *absolute profit* rather than the mean profit, which simply leaves out the normalization term in the denominator.

Similarly, a metric used will be another adjustment of the accuracy, denoted in the following as the *efficacy*. It is defined by counting only the samples that are predicted to be in a category different from the category "no action". Setting $C' = C \setminus \{"no\,action"\}$ and w to the Kronecker delta, it is defined as:

$$efficacy = \frac{\sum_i^{C'} \sum_j^C \#(i,j) \times w(i,j)}{\sum_i^{C'} \sum_j^C \#(i,j)}$$

Like the mean profit, the efficacy can be very susceptible to fluctuations caused by single predictions if again nearly all predicted classes fall into the category that is not counted (i.e. "no action"). Thus, when used for monitoring training, a lower confidence estimate of the efficacy is used instead of the raw efficacy itself. For that, the efficacy is treated as the probability of success in a binomial experiment with the numerator of the given efficacy formula being the number of successes and the denominator being the number of total trials. Note that this is not absolutely clean from a mathematical point of view as a repeated Bernoulli experiment assumes that the trials are independent, which might not be the case in a time-series due to correlations in samples that are close in time.

1.2.2 Data handling for iterative training

The data that was used for training the machine learning algorithms and analyzing the results comes from gaincapital[1] in the form of daily tick data consisting of a time-stamp in millisecond resolution and the bid and ask prices. When later chapters talk about the exchange rate, it is usually the average of the bid and ask prices.

The data set that is fed to the machine learning algorithms is a transformed version of said original data. The transformation comes from multiple indicators used in technical analysis (such as moving averages or normalizations) that do not need fitting on a specific data set (and thus do not need an independent training set themselves). This data was generated by running the framework in virtual evaluation mode and logging the current state of the system every second of market time. To train the agents that themselves are dependent on other agents' output, this process was repeated in multiple iterations, each adding new transformations as features.

When evaluating the performance of machine learning algorithms, it is crucial to make sure that none of the data used for the evaluation was used during training. Thus, great care has been taken during the training to ensure that no information is leaked in any step of training in order to not repeat the mistakes that Hurwitz and Marwala (2012) pointed out are inherent in several other proposed trading systems, such as the lack of a holdout data set that had not previously been involved in the process of fitting the models but is only used to evaluate the final performance. This can lead to the models overfitting the data that is used for training and validation by selecting the model that fits the validation set best; reporting the results for data the model had already seen would be subject to a bias, artificially enhancing the reported results even if they would not generalize to yet unseen data.

Taking the data of the years 2013 to 2015 and two iterations of dependent machine learning algorithms as an example, the process was as follows: the data for all years was generated using all agents that only depend on the raw tick data and had not been trained using historical data (e.g. simple moving averages or simple technical indicators). Afterwards, the data for the year 2015 was put aside as the holdout set for the final evaluation. The years 2013 and 2014 were split into two sets based on the days, such that trading days were alternately put into one set or the other. This yielded two distinct training sets, labeled $\mathcal{A}$ and $\mathcal{B}1$ for this example.

The data points of $\mathcal{A}$ were used for training the first iteration of machine learning algorithms. To achieve that, $\mathcal{A}$ was again split into two distinct sets: a training set and a validation set. The ratio for this split was about 4:1 and

[1] http://ratedata.gaincapital.com/, accessed on 2016-10-06

done in a way to not ignore the nature of time series data: the validation set consisted of the trading days at the end of the period. This ensures that we will not optimize our machine learning algorithm to interpolate between points in time that it had already seen but to extrapolate into the future.

After having trained the first iteration of machine learning algorithms on $\mathcal{A}$, they were integrated into the framework and the virtual evaluation was run again to generate a new set of data including the features generated by the added algorithms. The resulting data was split in the very same way as before, creating a new training set $\mathcal{B}2$ that consisted *only* of the data of the *same* trading days that were previously included in $\mathcal{B}1$. Thus, $\mathcal{B}2$ only consists of days that were not included in the training of the first iteration of machine learning algorithms. This makes sure that further machine learning algorithms, that are subsequently trained on $\mathcal{B}2$, are not trained on a set of days that the previous iterations of machine learning algorithms possibly overfitted on. Instead the previous generation had to generalize their knowledge onto the days contained the new training set.

$\mathcal{B}2$ could then be used to train the second iteration of machine learning algorithms in a similar way as before, splitting up $\mathcal{B}2$ into training and validation set again.

The year 2015 which was held out of the process so far could then be used to verify the performance of the whole ensemble, making sure that the system had never seen any of the data before.

Figure 1 visualizes the data split and the intermediate steps using the same labels as in the example above.

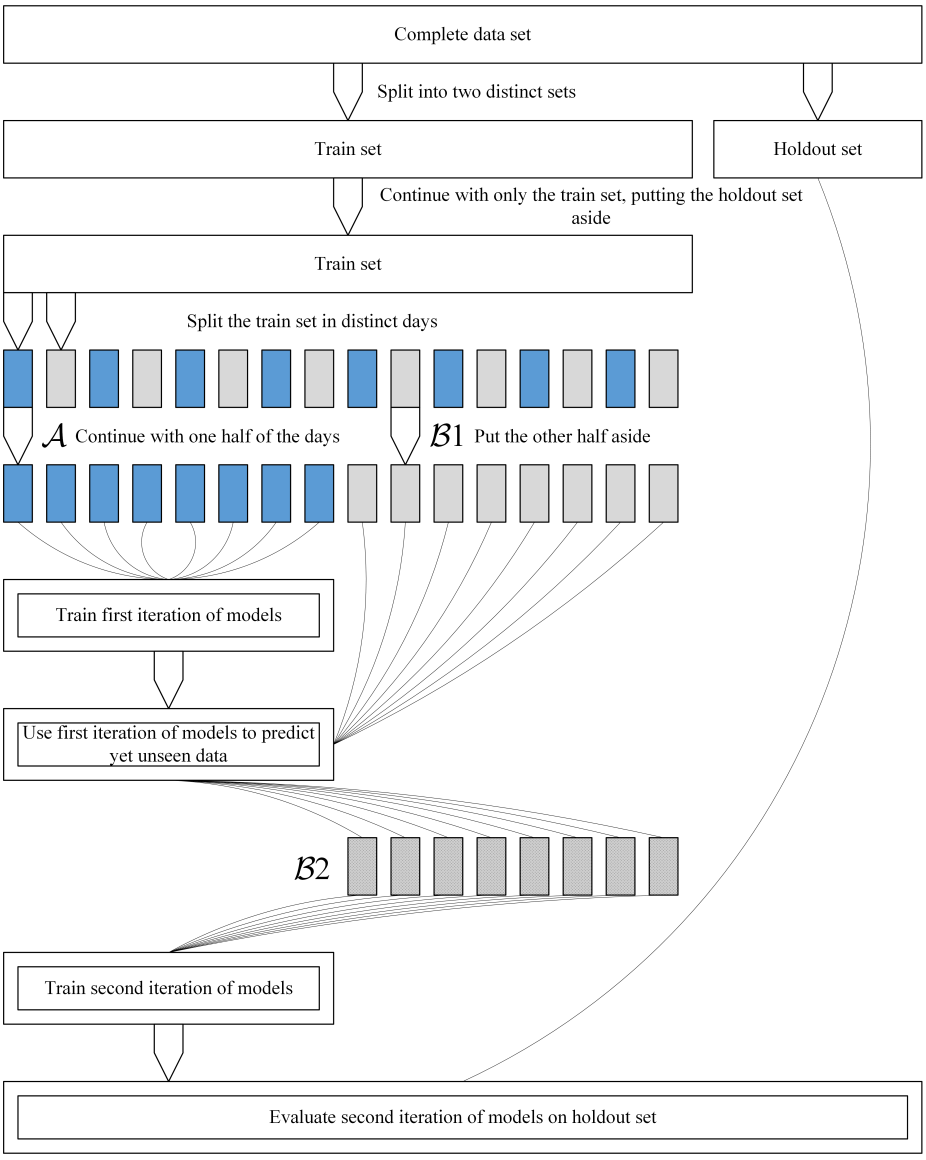

Figure 1: This diagram gives an overview over the handling of the data. The way the initial set of data is treated should prevent any data leaking and enable a clean evaluation of the final performance.

1.2.3 Finding a target function

Every supervised learning algorithm requires a so-called *target function* or simply *targets*. The targets are the outcome that the algorithm should learn to deduce from the data - be it categorical for classification problems or numerical for regression problems.
For autonomous FOREX trading, people have used many different *target functions* for their learners. The most naive function that at a first glance seems to be the most sensible would be to simply predict the exchange rate itself. That means, the target y_i for a certain time i would simply be the exchange rate at the next time step $i + 1$.

This has some severe drawbacks, however, as others have already pointed out (see e.g. Hurwitz and Marwala (2012)): most supervised learners use an error function to estimate the performance of the current prediction - most of the common error measure for time series prediction involve the difference between the current outputs $\hat{y}$ of the learner and the targets y, such as $\delta = \hat{y} - y$ (De Gooijer and Hyndman (2006)). This yields a measure of how close the learner's output is compared to the targets, however this is not what determines profit or loss in FOREX trading. In FOREX trading, the direction of the exchange rate is the important quantity - that is, whether the rate will go up or down in the future.

As we are interested in the direction of the price movement as opposed to the actual price itself, the most naive way again would be to use the derivation of the exchange rate as the target, which would yield $y_i = \frac{\mathrm{d}\rho(t)}{\mathrm{d}t}_{i+1}$. This improves the clarity of the intention of the target function - that is, to represent price movement. On the other hand, it does not solve the issue that the default error function of most learners fail to capture the important aspect of the target. As an example, imagine the situation that the target at a certain time y_i is $+1$. A prediction $\mathcal{A}$ of $\hat{y} = -1$ would only have an absolute difference of 2 to the desired output. A predicted value $\mathcal{B}$ of $+4$ would have an absolute different of 3. However, if one opened a trade based on the outcome of the predictor, the prediction $\mathcal{A}$ would lead to a loss as the predicted difference in price movement is incorrect. $\mathcal{B}$ would lead to a profit however, as the predicted direction of the price is correct, even though the predicted numerical value of $\mathcal{B}$ was farther away from the targets y than the value of $\mathcal{A}$.

Now it appears that predicting the price as a numerical value for FOREX trading with a classical target function that includes the difference of the prediction and the ground truth value might be conceptually flawed. The solution proposed and tested in this thesis is to include information about the resulting action into the loss function of the predictors. Intuitively, the loss function should punish mistakes that lead to different outcomes more

than mistakes that do not change the outcome. The punishment can be weighted according to how severe the outcome of the predicted action would be. For example, predicting a strongly rising price for a declining ground truth price leads to a loss and thus is clearly worse and should be punished more than predicting to execute no trades when the ground truth indicates a rising or falling price, which would neither lead to a profit or a loss.

If there is empirical information available about the expected outcome of any mistake or correct action, this knowledge can be included into the loss function to scale the punishment accordingly.

In addition to this solution to the aforementioned classical problems with regressing a price to execute trades, the target that is used in this thesis is slightly different from a simple derivation of the price. Instead of constructing the numerical target value y_t for a time t by taking the derivative of the price as $y_t = p(t+T) - p(t)$ where T is a previously specified fixed time interval, the target function is the maximum value of the price change over a fixed time interval T before the difference changes signs. This is given in pseudo-code in Algorithm 1.

Before using this as our target for training, it is important to verify that the actual profit from trades is well-represented by the target function at the time of opening the trade. This was achieved by running the trading framework on the real market data from 2015, opening trades randomly and closing them with a trailing stop loss which was initially set to start 5 pips below the opening price of the trade. The outcome of this experiment was a series of trades with their profit in pips, each trade annotated with the value of the target function at the time when it was opened. Figure 2 visualizes the dependency between the target function and the resulting profit or loss of a trade that was to be opened at that function value. It is apparent that below a certain threshold, trades tend to lead to a loss; and similarly above a certain threshold tend to be profitable. This can be further quantified by looking at the distributions of the outcome of trades for different ranges of the target function. Figure 3 shows the distribution of the outcome of (long) trades that were opened inside certain ranges of the target function. The distribution suggests that we can find a threshold value for the target function which, if we could reliably predict it, could indicate whether a trade would yield a profit or be a loss.

For the remainder of the thesis, this recurring threshold value will be 5. Of all simulated trades that were opened above this threshold for long trades or below the negative of this threshold for short trades, more than 97% turned out to yield a profit even with the very simple strategy of closing trades used during the simulation.

With this fixed threshold, we can now classify any time point into one of three ground-truth categories: *sell*, *buy*, and *no action*. Opening a short

Algorithm 1 This pseudo-code gives the calculation of the target function that is used for training the models. The input requirement is an exchange rate in minute intervals (here P). The constants in the code are the interval of the maximum lookahead (here 15 minutes) and the normalization of the returned value (here one Pip; i.e. 0.0001).

```
function calculate_target(P):
  max = 0, min = 0
  last_sign = 0
  minutes_lookahead = 0
  while minutes_lookahead < 15:
    minutes_lookahead += 1
    price_derivative  = P[now + minutes_lookahead] - P
        [now]

    s = signum(price_derivative)
    if last_sign != 0 and s != last_sign:
      break
    last_sign = sign

    if price_derivative > max:
      max = price_derivative
    if price_derivative < min:
      min = price_derivative
  if abs(min) > abs(max):
    return min / 0.0001
  return max / 0.0001
```

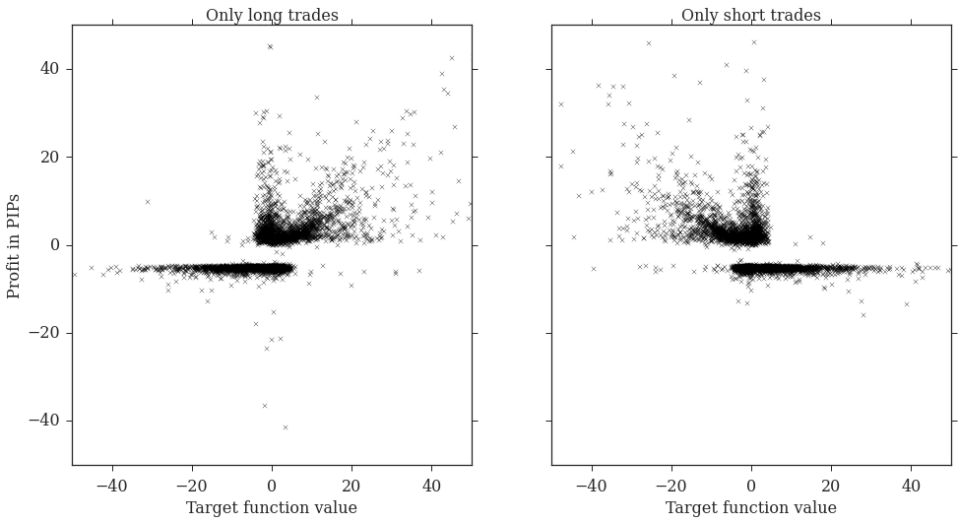

Figure 2: This figure shows scatter plots of the profits of simulated trades versus the value of the target function at the time the trade was opened. The left plot shows only long trades (where the leading currency was bought) while the right plot shows short trades (where the leading currency was sold). Taking the long trades as an example, it is visible that there is a dependency between the target function at a time t and the profit of a trade that was to be opened at that time. Long trades that are opened while the target function is negative tend to trigger the default stop loss value leading to a loss of money; long trades that were opened when the target function was positive tend to yield a profit. This relationship is inverted for the short trades.

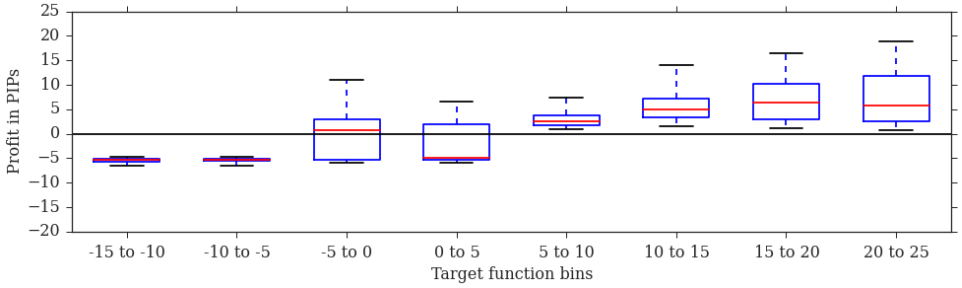

Figure 3: This boxplot shows the range of the profits made by long trades (where the leading currency was bought) for different ranges of the target function at the time the trade was opened. The horizontal line at 0 indicates a net loss or profit of 0. Values above the line are the outcome of profitable trades while values below the lines come from lost trades. The whiskers indicate the 5% and the 95% percentile, respectively. Long trades that were opened with the target function below -5 tend to trigger the default stop loss. The vast majority of trades above the target function value of +5 turns into a profit.

trade (i.e. selling the leading currency) is said to be the correct choice if the value of the target function is below -5. If it is above +5, opening a long trade (i.e. buying the leading currency) is said to be correct. With the target function in the range $[-5, +5]$ the target classification is *no action*. Using these categories, we can give confidence intervals for the resulting profit or loss still based on the simulated trades. Table 1 shows the bounds of the 90% interval for the outcome of the two possible actions (opening a long trade or a short trade). The outcome of not opening any trade is of course always 0; thus it is not shown in this table. It can be seen that executing the right action will generally lead to a profit while the wrong action might lead to a loss. It is apparent that executing the wrong action can have different outcomes depending on what the ground-truth category was. E.g. while the ground-truth category was *buy*, a short trade will generally lead to a loss; the same trade when the recommendation was *no action* can however still be profitable.

The knowledge about the severity of different miss-classifications will later be used to scale the punishment of the machine learning models accordingly. It has to be stressed that these bounds come from a simulation with randomly opened trades. However, these can be understood as a lower bound for the true values obtained by a more sophisticated trading strategy, assuming that this strategy would perform better than random.

While there is now a target that can serve as a helpful tool for training it has to be noted that, as Hurwitz and Marwala (2012) have pointed out, the final evaluation necessarily needs to be based on the actual profit using a live or

	predicted buy		predicted sell	
	5%	95%	5%	95%
ground truth: no action	-6	8.6	-6.1	9.3
ground truth: buy	1.1	18	-6.7	-4.9
ground truth: sell	-6.7	-4.8	0.7	16.4

Table 1: This table shows the bounds of the 90% confidence interval for the expected profit or loss of long and short trades. The trades were split up into three ground-truth categories by using the previously found threshold for the target function.

simulated trading system which not only includes the initial prediction of when to open trades but also the closing and loss minimization strategies. This is the only way to ensure that the output of the trading system is sufficient for profitable trading.

1.3 Merging the output of multiple models

This thesis's final prediction which eventually triggers an action on the market will be a merged prediction made up from many different feature transformations and predictors. The idea of merging different predictions to improve the reliability and robustness of the prediction is not a new one - especially in time-series forecasting. De Gooijer and Hyndman (2006) give an overview over the available literature on combining time-series forecasts, focusing on homogeneous methods that each provide a prediction for a future value in a time-series. The evaluation of the M-Competition as well as the M2 and M3-Competitions, time-series forecasting competitions, found that the combination of different methods outperforms the single methods in most cases (Makridakis et al. (1982), Makridakis et al. (1993), Makridakis and Hibon (2000)). Stock and Watson (2004) analyze different combination forecasts for an economic growth data set. They combine homogeneous forecasts (forecasts that each predict the value of the same time-series for a given point in the future) using different methods and find that the combined forecasts usually provide better results and are more stable than the original predictions, which tend to have different characteristics during different time periods. They find that simple combinations, for example the arithmetic mean, perform better than more sophisticated approaches that try to incorporate the prediction performance of the individual predictors into the final forecast, though. The approach in this thesis differs from their purely homogeneous view on time-series forecasting, as it allows incorporating arbitrary single predictions (such as categorical predictions or regressions on different targets) as opposed to requiring each single prediction to estimate a fixed point in the future of one time-series.

See Clemen (1989) for a review and an annotated bibliography about existing work on combining forecasts.

The solution in this thesis will be able to combine the different predictions in a non-linear way, selecting the combination weights robustly. The predictions will be combined by training another machine learning classifier on top of them. This technique of training a machine learning model on the output of other models is generally termed *model stacking*, introduced by Wolpert (1992) in the context of neural networks, extended by Breiman (1996) for regression and generalized further by LeBlanc and Tibshirani (1996).

By using an appropriate model, this technique allows to integrate different types of predictions (such as numerical and categorical predictions) as well as additional features which are not predictions themselves. For example, the time of the day or an estimation of the volatility of the market could be included as a feature, allowing the merging model to assign different weights to the original predictions based on such additional features.

2 Agents

The heart of the prediction framework is a collection of different agents, that each give an estimate about both the current action to be taken (buy or sell) and a confidence estimate about this action. Each agent can use different data for its prediction and each agent can also have a unique frequency at which it performs predictions. That implies that certain agents' prediction might be updated slower or more frequently than others'.

The result of each agent's analysis is a value for its predicted action to be executed, hereafter called the *mood*, in the range $[-1, +1]$ where -1 stands for the desire to open a short position or a *sell* and analogically $+1$ stands for wanting to open a long position or a *buy*. In addition to the *mood*, also a confidence for that prediction is given, ranging in $[0, 1]$. This confidence can be understood as a probability of the mood to correctly reflect the current market. Thus, a confidence of 1 means that the agent is certain that the mood is correct. This does not incorporate the general reliability of the agent, which will be taken into account in the step that merges the predictions.

The remainder of this section will present some of the classical indicators that have been integrated into the framework to give the reader an idea of the intuition behind technical analysis. Further, the ability to add other types of features to the final prediction model, different from a direct prediction of the direction of the exchange rate, is shown by describing a time-aware clustering of the exchange rate that has been implemented.

2.1 Classical Technical Indicators

As introduced in subsubsection 1.1.1, technical analysis focuses on the value of the exchange rate itself and tries to derive future behavior based on patterns that were observed in the past.

One important aspect of technical indicators is to transform the current exchange rate into a fixed value range of which a certain consistent behavior is expected even if the actual value of the exchange rate changes over time. The output of the technical indicator can be understood as a normalization of the actual price. This normalization, utilized by technical traders, might also be suited as a normalization technique for other machine learning algorithms. Thus several different technical indicators have been implemented in the C++ core framework to act as input for the machine learning algorithms.

2.1.1 Relative Strength Index

The relative strength index (RSI) transforms the exchange rate into the range $[0, 100]$.

After an arbitrary but constant timeframe p (the period of the RSI), the price movement $m = P(now) - P(now - p)$ during that period is evaluated and the absolute up and down movement (U and D respectively) are defined as:

$$U = \begin{cases} m & , m > 0 \\ 0 & , otherwise \end{cases}, D = \begin{cases} 0 & , m > 0 \\ -m & , otherwise \end{cases}$$

The actual RSI is then defined using a moving average of U and D with the smoothing parameter N as follows:

$$RSI = 100 - \frac{100}{1 + \frac{MA(U,N)}{MA(D,N)}}$$

Hyperparameters of the RSI are the timeframe p, the smoothing window N, and the choice of the moving average.

The RSI is utilized as an indicator of the current short time trend, being closer to 100 when the recent trend was strongly positive and closer to 0 when the recent trend was strongly negative. Figure 4 shows the value of the RSI over a trading day; an example agent has been implemented to generate *buy* (/and *sell)* signals when the RSI is under (/over) a set margin of 20 (/80).

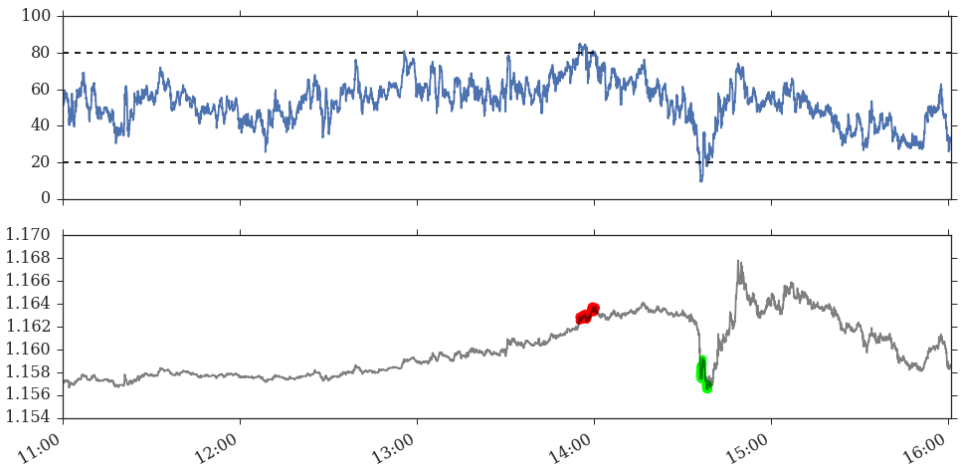

Figure 4: The upper plot shows the value of the RSI including the decision margins for an example agent that generates *buy* or *sell* signals based on the RSI. The lower plot shows the exchange rate (mean of bid and ask) with the sell signals highlighted red and and buy signals green. The day is 2015-01-21 and the timezone is GMT.

2.1.2 Commodity Channel Index

The Commodity Channel Index (CCI) measures the divergence of a price from its mean, transforming it into an unbounded value around 0. After a previously specified timeframe p (the period of the CCI), the average of the maximum price, the minimum price and the closing price of the period is calculated; this value is called p_t. The recent divergence of this value is measured by its difference to a moving average over a specified length N. The recent divergence is then scaled by the mean absolute deviation (MAD). The resulting value is then scaled by a scaling factor s, typically set to values such as $\frac{1}{0.015}$.

$$CCI = s\frac{p_t - MA(p_t, N)}{MAD(p_t)}$$

The CCI implementation in the market framework uses an online estimate of the mean absolute deviation that is reset at every start of a trading day as depicted in Algorithm 2. Hyperparameters of the CCI are the duration of one period p as well as the choice of the moving average. The scaling parameter s is not treated as a hyperparameter as it only results in a constant linear scaling.

An example agent has been implemented that generates buy and sell signals if the CCI is below -100 or above $+100$ respectively. Figure 5 shows the value of the CCI on an example day as well as the decisions of the agent based on the CCI.

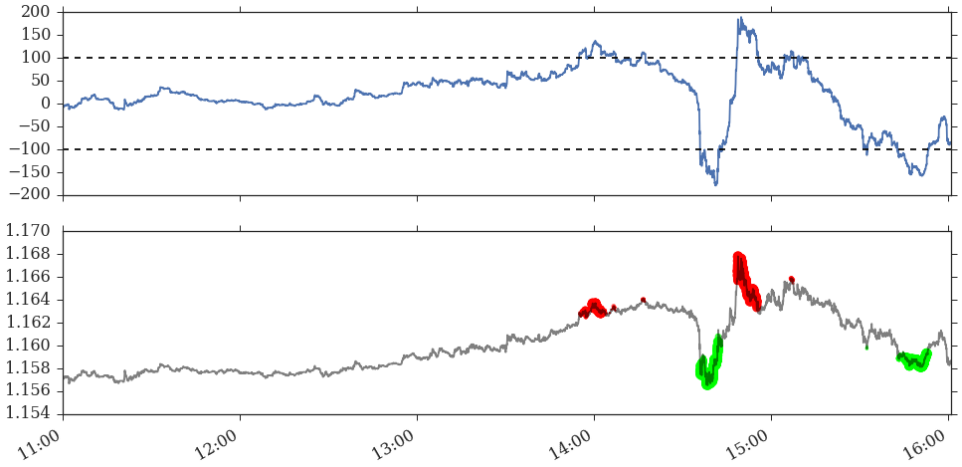

Figure 5: The upper plot shows the value of the CCI and the decision margins of the example agent; the lower plot shows the exchange rate with the buy and sell signals of the agent highlighted in green and red respectively. The day is 2015-01-21 and the timezone is GMT.

Algorithm 2 Pseudocode for an online estimate of the mean absolute deviation that utilizes a running estimation of the mean according to Welford (1962). Note that the additional square root introduces an inaccuracy, which is assumed to be negligible in this context as the indicator still functions as another normalization of the exchange rate.

```
n = 0, mean = 0, M2 = 0
def get_mean_absolute_deviation(new_value):
  n = n + 1
  delta = new_value - mean
  mean = mean + (delta / n)
  updated_delta = new_value - mean
  M2 = M2 + sqrt(abs(delta * updated_delta))
  return M2 / n
```

2.1.3 True Strength Index

The True Strength Index (TSI) transforms the exchange rate into a value in the range $[-100, +100]$.

After a fixed period length p, the difference $delta = P(now) - P(now - p)$ is calculated. The value of the TSI is then calculated by two applications of a moving average smoothing:

$TSI = 100 \frac{MA(MA(delta, N_1), N_2)}{MA(MA(|delta|, N_1), N_2)}$

Hyperparameters of the TSI are the period length p, the smoothing periods of the moving averages $N_1$and N_2, and the choice of the moving average algorithm (which typically is an exponential moving average).

The TSI is used similarly to the RSI to indicate the current trend, being closer to -100 when the recent trend has been negative and closer to $+100$ when the recent trend has been positive. Figure 6 shows the values of the TSI over a sample day; an example agent has been implemented to generate buy and sell signals based on the TSI using a margin.

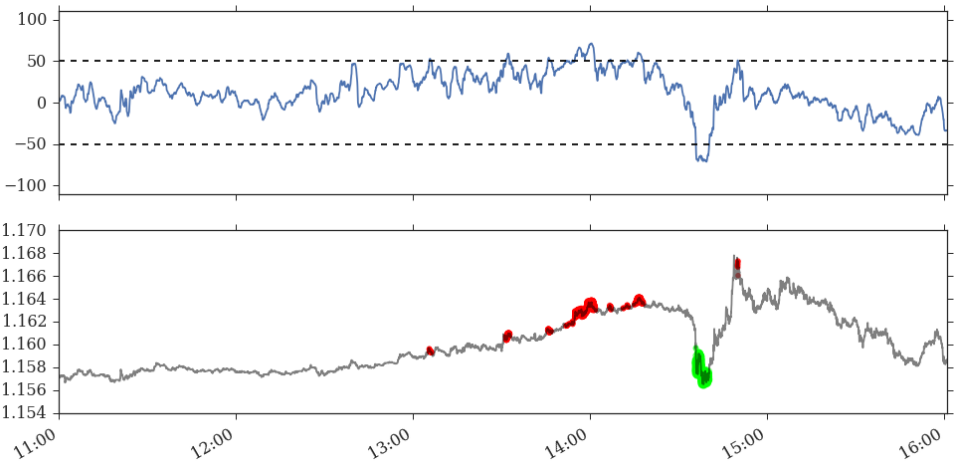

Figure 6: The upper plot shows the value of the TSI and the decision margins of the example agent; the lower plot shows the exchange rate with the buy and sell signals of the agent highlighted in green and red respectively. The day is 2015-01-21 and the timezone is GMT.

2.2 Clustering of the time-series

This agent provides a clustering of the market data by unsupervised learning of clusters based on heavily preprocessed and transformed original outputs of the market system (including for example the technical agents). The intuitive idea behind the clustering is that the market might have different properties over time which might be relevant to weight the prediction strength of other models for the final forecast.

The following subsections will each describe and evaluate a key step in the clustering process.

2.2.1 Data handling and preprocessing

The data to be clustered is the training data for the first iteration of machine learning models as described in subsubsection 1.2.2. This includes not the raw exchange rate but instead all the output of the technical agents available to the ensemble.

2.2.2 Feature reduction and transformation

Prior to the clustering, the input data is transformed, reducing the amount of features from arbitrarily many to 10. This feature transformation and reduction is achieved by using the original features as the input for a neural

network, which is trained to reduce mean squared error on the targets as described in subsubsection 1.2.3. This approach differs from the classical way of using neural networks as an alternative to classical methods such as Principal Component Analysis to reduce the dimensionality of data.

The typical setup is to use a neural network that is trained to reproduce the input fed into it while having one or more small central layers so that the network is forced to use an abstract representation of the data (Hinton and Salakhutdinov (2006)). Such a network is called an *Auto-Encoder*. Auto-Encoder have also been used for clustering by extending the optimization target to include information about the current cluster centers (Song et al. (2013)). However, it has also been shown for classical regression or classification networks that intermediate representations (i.e. the output of hidden layers before the output layer) have a meaning related to the target; applying a network to image classification Girshick et al. (2014) give evidence that such an intermediate representation closely resembles certain abstract features of the data that are relevant to the target. Again on images, Zeiler and Fergus (2014) show that such intermediate features can carry an intuitively understandable meaning. It therefore seems likely that the same applies even to problems not involving images.

The network used is a simple feed-forward network with 2 RELU layers of 64 neurons each, followed by a layer with a linear activation function of 10 neurons. These 10 neurons are the last layer before the output and will be used to get the reduced and transformed input features. The intuition behind the linear activation function is that the network should ideally learn a representation with a linear relationship to the target function; this should make it easier for successive steps (e.g. clustering) to keep the relationship intact without explicitly knowing about the target function.

To evaluate whether this feature reduction and transformation technique is helpful, it was compared to several other common preprocessing techniques: Independent Component Analysis (ICA), Principal Component Analysis (PCA), and Partial Least Squares Regression (PLS). The data was divided into 10 continuous cross-validation sets. For each cross-validation set, 75% was used for training the models and 25% was used for evaluation. For ICA, PCA, and PLS the data was normalized by subtracting the mean and dividing by the standard deviation (for each feature); note that this did not make a difference for PLS as expected. For all the comparison methods, the number of features was limited to 10 (e.g. the ten principal components that explain most of the variance).

The regression model was a regression based on K-Nearest-Neighbours (with $k = 100$). The intention behind this choice is that the projection of the data should ideally lead to a feature space with similar target values having a low euclidean distance to each other. This property would be ideal for a

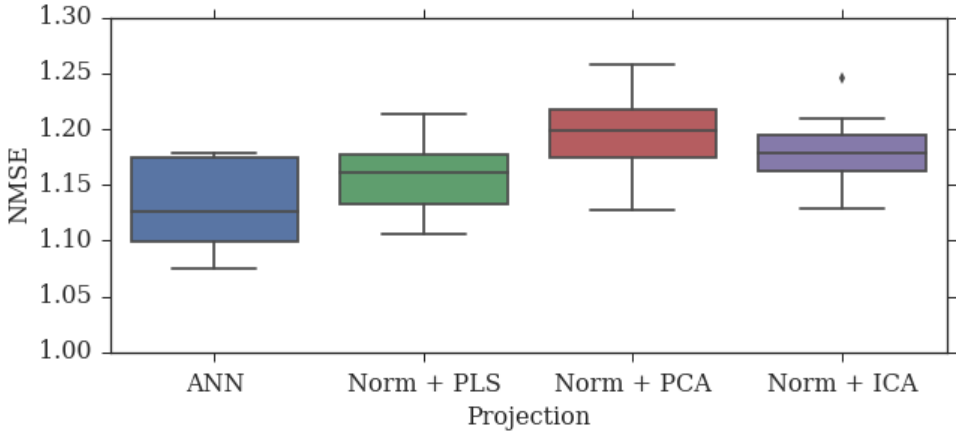

Figure 7: This boxplot shows the normalized mean squared error of a regression on the target function using K-Nearest-Neighbour regression. The data points were transformed using different methods, which are shown on the x-axis. While using an artificial neural network for the feature transformation yields a lower error in the mean case, there where large outliers in some cross-validation runs (out of the axis in this plot). Thus it could not be concluded that the method presented here significantly outperforms the comparison methods in most runs as the comparison methods tended to show a smaller spread around the mean.

subsequent clustering that is based on euclidean distances.

The error metric used was a normalized mean squared error (NMSE), defined as $\frac{MSE}{MSE_{baseline}}$. The baseline mean squared error in the denominator was a constant prediction with the mean of the training data's targets. Note that a NMSE below 1.0 indicates a result better than the baseline and above 1.0 indicates a result with a higher error than the baseline.

The results are shown in Figure 7. It can be seen that the transformation method used here allows the regression to achieve lower testing errors than the other preprocessing methods in most of the runs. However, due to outliers (outside of the axis in the plot), there are some runs where the method presented here performs worse that the comparison methods. It is further interesting, that no methods is able to perform better than the baseline in any cross-validation run.

2.2.3 Discretization

The transformed features are discretized into 300 different points using KMeans clustering. To achieve a higher coherency over time, the time-lagged independent components of the data are used for the clustering.

Originally introduced for signal processing (Molgedey and Schuster (1994)), Time-Lagged Independent Component Analysis (TICA) is similar to Principal Component Analysis in the sense that it tries to project the data into a dimension of maximum variance; however, TICA finds the dimensions of maximum variance over time (using a previously specified time lag). On a truly markovian process (i.e. molecule dynamics) Pe/rez-Herna/ndez et al. (2013) could show that the TICA projected subspace is well suited to discretize the slowly changing components of a system.

Indeed we can show that the KMeans clusters on the TICA projected subspace tend to be more consistent over time in the original time-series - note that the time-series information are not included in the KMeans clustering and the data points could as well be shuffled randomly prior to KMeans clustering.

To evaluate this, the output of the previous step (the neural network transformation) was divided into 10 continuous cross-validation sets. For each set, KMeans clustering was used to generate 300 clusters with either no preprocessing, a TICA projection, or a PCA projection. The evaluation metric was calculated on a subset of the data that was not used to calculate the transformation matrices for the preprocessing and the cluster centers of KMeans; this subset was chosen to be the last 25% of the data to make sure that no information was leaked due to correlation inside the time-series. Figure 8 shows the resulting mean cluster lengths per cross validation run. In significantly more runs, the preprocessing step of a TICA projection allows KMeans to find clusters that are more consistent over time.

After this step of KMeans clustering, each data point is assigned one of 300 discrete classes; this reduces the dimension of features for each point to 1. This timestep-to-cluster mapping will further be called the discretized data.

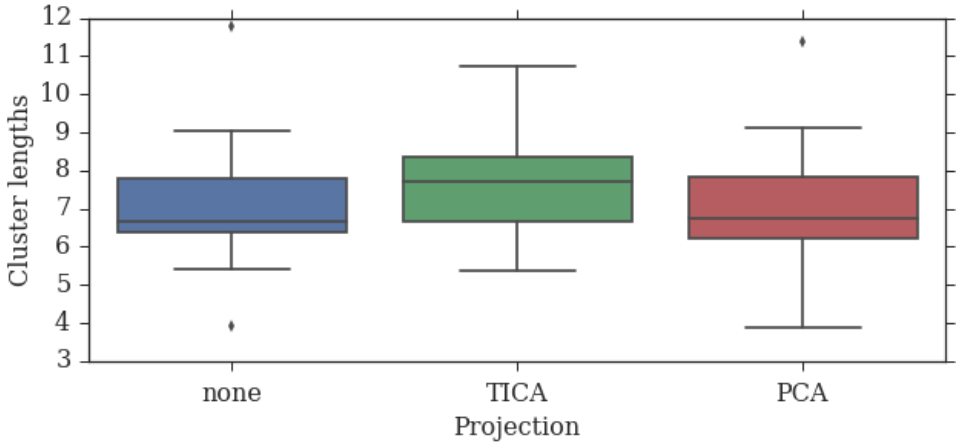

Figure 8: This boxplot shows the mean length of successive time steps that one KMeans cluster persists. The results come from 20 cross-validation runs on the data from 2013 to 2016. The mean length of the KMeans clusters that were calculated on a TICA projection is significantly higher than using no projection or PCA. The amount of data points per cross validation set was around 250′000. In 16 of the 20 runs, the TICA projection yielded the longest clusters, this is significant at a 5% significance level.

2.2.4 Markov model estimation and PCCA

The discretized data are now used to estimate a symmetric markov model. Note that the underlying process is most likely neither truly markovian nor symmetric; the goal of this step is however to find temporal processes in the data that change only slowly (and still have some relation to the target function due to the transformations prior to the clustering). In itself the resulting clusters will not allow profitable trading in any way; they will act as an additional feature for the other estimators.

Firstly, a markov model is estimated from the discretized data using the data points of different days as unique trajectories to make sure that possible gaps between days do not lead to inaccuracy. To estimate a markov model, a transition matrix has to be generated; a transition is not counted between directly successive points but between points that are a few timesteps apart; this *time lag* is chosen to be the same lag as in the TICA preprocessing step. It should further help the markov model represent slowly changing processes.

Once estimated, the stochastic transition matrix (normalized row-wise so that the entries are no longer transition counts but transition probabilities) is clustered using Robust Perron Cluster Analysis (PCCA+), which yields a mapping from every row in the transition matrix (which corresponds to

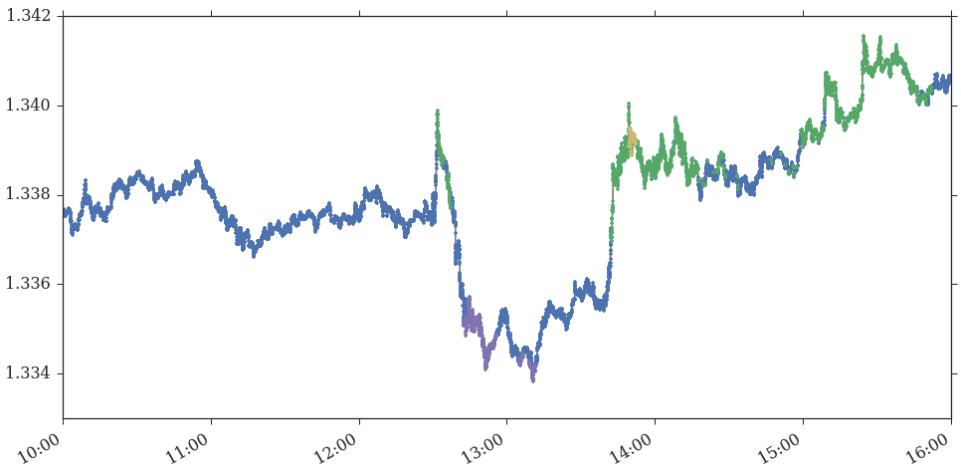

Figure 9: This plot shows the exchange rate (EURUSD) over a day, coloring the samples according to the PCCA+ cluster with the highest probability. The day is 2013-06-18 and the timezone is GMT.

one discrete point of the discretized data) to the probabilities belonging to each cluster. This probabilistic mapping is able to better handle discrete points that can not be assigned to one cluster, e.g. because the point might be a transition between two longer-lasting states. The main advantage of PCCA+ compared to other probabilistic clustering methods is that PCCA+ is able to preserve the slow time-scales of the underlying markov process (Röblitz and Weber (2013)).

Figure 9 shows the PCCA+ clusters over one example day. Visually, the clusters seem to have some temporal persistence; it is interesting to see that samples belonging to the largest cluster have a mean value of the target function of $+0.05 \pm 8.3$ while the second largest cluster's samples are at -0.28 ± 9.4. This might be evidence, though no statistically significant proof, that the clustering process kept some relation to the target function intact.

The final output of this clustering that can be integrated into the ensemble and used as a feature for the final prediction are the probabilities of being in each of the PCCA+ clusters.

2.3 Gradient boosting model

The model that combines the predictions of all other indicators and models is a gradient boosting tree. It outputs a regression on the target function described in subsubsection 1.2.3 which will be thresholded and used to execute an action in the market. The implementation used is XGBoost (Chen

and Guestrin (2016)).

One advantage of a gradient boosting model (GBM) compared to other models is that the GBM can handle a combination of different types and ranges of data natively; normalizing the input data is not necessary for the model to function and a GBM model is able to handle a mixture of categorical and continuous values very well.

Another advantage is that a GBM can optimize any objective that can be described in terms of a first order gradient and a second order gradient. This allows us to integrate penalty into the regression target based on the findings described in subsubsection 1.2.3; different types of miss-classifications based on the regression target can be weighted differently. The way it was done for this example implementation is that the gradient corresponds to $w * (\hat{y} - y)$ with the weight w being set to distinct values depending on the outcome of the prediction. The weight corresponds to the strength of the influence of each sample and can be understood similar to a distinct learning rate for the different cases. For example, the impact of identifying a sample that could be used for a trade as "no action" should be relatively low compared to classifying a sample that belongs to the class "buy" incorrectly as "sell".

As this final model gets many input features, one additional step is employed to reduce the dimensionality of the feature space. This *feature selection* is described in the next section.

2.3.1 Feature selection

Feature selection is the process of removing features that are detrimental to a model's predictions prior to the training of a model. An intuitive idea of why some features can be detrimental is that there might be some features or combinations of features which allow a classifier to learn the training samples well, however which transfer badly to new, previously unseen samples (and thus endorse overfitting). In the domain of time-series classification a possible explanation for this property of features can be that the underlying distribution of the feature changes over time and thus future observations will be very dissimilar to the ones seen at the time of training.

There are multiple common ways of doing feature selection. One is the iterative feature elimination: let the amount of features $\vec{x}$ be denoted by N. N iterations are now executed, consisting of training and evaluating the model while leaving out the feature $\vec{x_i}$ for iteration i. After the iterations, the feature that led to the lowest final score of the model will be left out and the next iteration will start with $N - 1$ features. This can be repeated until the score of the model does not improve anymore by leaving out any feature. Each training and evaluation phase should be cross-validated.

This method has drawbacks: not only does it require many runs (N for the first decision, $N - 1$ for the second one, etc.), but it might also be unable to eliminate groups of features that degenerate the performance of the model if any one of these features is present. One simple example for that would be highly correlated features that carry the same amount of relevant information - removing one of those features would not influence the score of the model as it could resort to the other highly correlated features; to alleviate this latter issue, features could be removed in groups of two or more. This however increases the amount of required runs even more.

The method used in this thesis is a probabilistic adaptation of the described algorithm: prior to the decision process, a high number of iterations is executed. In each iteration, a fixed percentage of the features are dropped (e.g. 50%) and the configuration is cross-validated; afterwards, the resulting mean score and its standard deviation of this configuration is saved as well as the information about which features were dropped. After executing enough iterations, each feature is looked at in a similar way as the initial description of the algorithm: for each feature $\mathcal{F}$, the mean and standard deviation of the different cross-validation runs where $\mathcal{F}$ was dropped (along with different other dropped features) are combined, yielding a total mean and standard deviation for all runs without $\mathcal{F}$. The feature with the lowest mean (or the lower bound of a 95% confidence interval of the score) is dropped along with all cross-validation runs where this feature was dropped. This process is repeated as long as enough runs remain so that a statistically significant decision can be made.

This process should alleviate the problem of correlated groups or features that exhibit a certain behavior only in combination with other features; the reason for this is that if a special set of features leads to a low mean score in the cross-validation runs, then one of the features $\mathcal{F}$' will eventually be selected as the feature with the lowest score during the elimination process - it does not matter which one exactly. After removing $\mathcal{F}$', all the runs remain where the other (correlated) features were also removed. The selection process will then continue with the other correlated features that still lead to a lower score in combination with the absence of $\mathcal{F}$'.

Note however that this feature selection process still needs 2^n runs to select n features. To speed up the process, it can be repeated in iterations. For each successive iteration, the top n features can be automatically retained and the worst n features automatically removed.

3 Framework & software architecture

A major part of this thesis is the development of a trading framework that is able to contain and manage the discussed ensemble of agents. Additionally, it should be able to react on the agents' output and execute the actual trading in either a real market or a simulated environment.

The framework application must fulfill a core set of *functional* and *non-functional* requirements in order to be suitable for the task.

Functional requirements are properties of the system that are necessary in order to fulfill the desired functionality; they answer the question of *what* the system has to accomplish. The following functional requirements have been identified as the core functionality of the framework.

Agent communication The framework must allow the agents to communicate with each other. This does not only extent to the agents implemented directly in the core application but also to external agents that use a network interface to communicate.

Market communication The framework must be able to communicate with a broker and allow an information flow between the broker and the core application including the agents.

Reproducibility The framework must allow for a way to check the influence of different parameters or agents on (past) decisions. This implies the necessity for a virtual market system which can replay historical data to evaluate decisions.

Interpretability The framework must allow for the possibility of post-mortem[2] analysis.

Non-functional requirements are properties of the functionality of the system; they answer the question of *how* the system should accomplish its tasks.

Low latency All communication (inside the framework and to the external broker) must have as little overhead as possible. This is necessary in order to use the system in a real time environment; the system must only take the least amount of time possible for decisions in order to react as quickly as possible to market events.

[2] In this context *post-mortem analysis* stands for the analysis of decisions and available information in retrospect - e.g. by logging all variables and decisions of the agents while running.

Fast prototyping Implementing and testing new functionality must be possible without a lot of additional work. This is a necessity in environments where the concrete path to achieve the desired goal is not known beforehand; without it, a lot of work would be spent on implementing and testing new agents and decision processes.

3.1 High level overview

The core framework was written in C++ not only because of familiarity but also because this would allow for a high performance application. The core framework communicates over network sockets with a C library which is embedded into a MetaTrader *Expert Advisor*[3].

The core framework includes a collection of agents written in C++; however, it also allows for connection of external agents via a *ZeroMQ*[4] interface.

Figure 10 gives an overview about the components of the system and the programming languages that were used to implement these components. The market data originally comes from a broker and is dispatched to the various involved components using different interfaces.

[3] In the MetaTrader environment, an *Expert Advisor* denotes a script, written in the MetaQuotes scripting language (MQL), that is able to fetch information from the market and react to it, e.g. by opening, updating, and closing trades.

[4] *ZeroMQ* is a messaging library with bindings to many popular programming languages, e.g. including C++, Python, Matlab, and R.

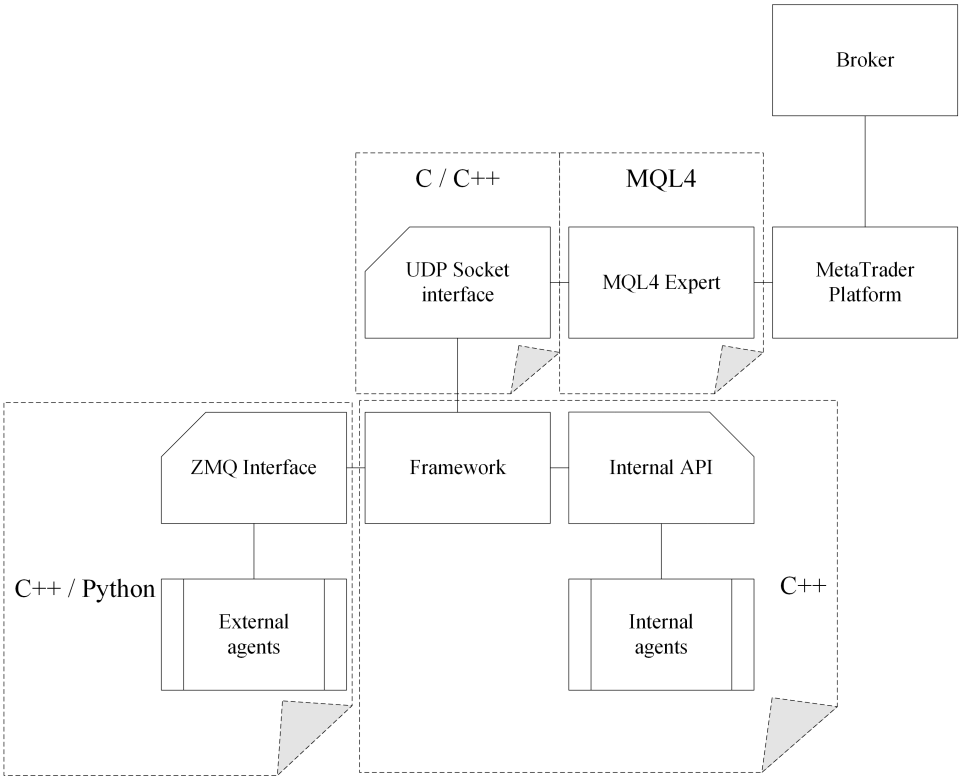

Figure 10: This figure gives a high-level overview about the core framework and its communication with the broker. Interfaces and message channels are indicated by lines. Simple squares indicate single components of the system. Double stroked squares indicate categories of components. Dashed containing boxes give information about the underlying programming language. Simple squares with an angled edge indicate an interface between multiple components.

3.2 Agent infrastructure

An *agent* in this context is a piece of software that is able to receive information, manipulate it, and finally provide new or transformed information. The major part of the available agents are implemented directly in the C++ core of the framework for additional performance. However, the ZeroMQ interface of the framework also allows for the integration of agents written in other languages. This is mainly utilized for agents written in Python to be able to fully exploit the availability of machine learning libraries, such as scikit-learn (Pedregosa et al. (2011)) or keras (Chollet (2015)). Due to the possibility of writing components of the framework in the form of agents in Python, there is nearly no additional overhead required to go from the step of having an algorithm developed and tested in Python to the step of running a backtest on historical market data under realistic conditions. There is obviously also no additional step involved when wanting to use the agent for real trading as opposed to the simulated market.

A key role in the communication process of the agents play the *latent variables* of the systems, which in this context are annotated decimal numbers that are output by different agents. A simple example for such a *latent variable* could be a moving average over the exchange rate of one currency pair or an estimate of the future direction of the market by one of the agents. Together with the *observable variables*, which are completely made up from the exchange rates themselves, they define the repertoire of possible inputs for every agent. Every agent can require a certain set of input variables and is then guaranteed that its input variables are updated before requesting an update of its output variables.

This implicitly arranges the agents in a directed acyclic graph structure with the edges defined by the agents' dependencies on each other; an example for this structure can be seen in Figure 11. A conflict-free order of updating the agents can be found by simply finding a topological sorting of this graph.

All outputs of the agents are registered with the framework and automatically logged for post-mortem analysis.

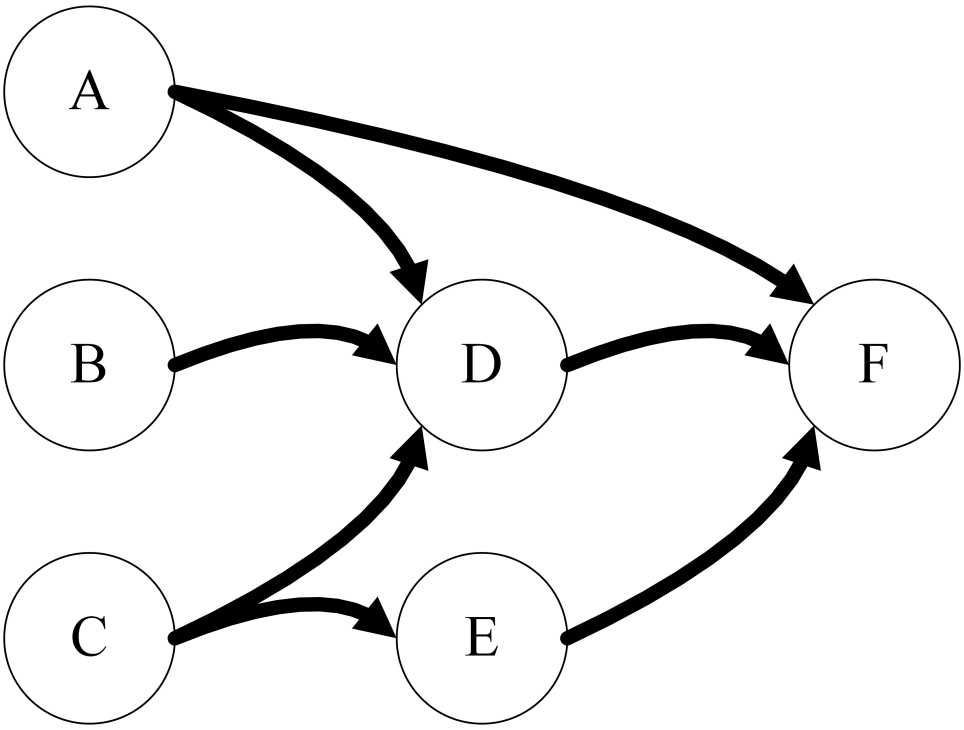

Figure 11: This graph shows the dependency graph of 6 example agents; the arrows point into the direction of the data flow (thus, e.g. F depends on D). The agents need to be evaluated in an order that ensures that no agent is evaluated before all of the its dependencies. In this example, such a topological sorting could be A, B, C, D, E, F. While every agent can provide multiple outputs and does not have to require all outputs of the agents it depends on, one agent is always evaluated as a whole. Thus, the dependencies are defined on an agent basis (as opposed to a feature basis).

3.3 Virtual market & statistical evaluation

The virtual market is the component of the framework that can take the role of the broker interface and replay historical FOREX data. It uses the same internal API that the connection to the real market would use. This allows for reliable testing under nearly equivalent conditions compared to a real time run with an active broker connected through the MetaTrader interface.

The virtual market can read tick data from either a third party source or from the data the framework collected from a real trading day. After communicating the tick data to the core framework, it can then react on orders received from the agents. The virtual market then manages trades, including properties such as the stop loss or take profit settings for a trade, and finally calculates the profit of a trade when it has been closed.

On the available machine, the virtual market can simulate a full trading year in less than a day, providing an efficient way of backtesting[5] under nearly real conditions with a lot of data. The amount of data that can be processed is the major advantage compared to the alternative: testing on the real market but with a dummy trading account. In the latter case, the actual broker would be used on life data instead of the virtual market on recorded data. This would then be a test under factually real conditions - still under the assumption that our actions are too small to influence the market. This real test would however have a relatively slow throughput - especially if the goal is to collect enough information to be able to make statistically significant statements about the gained profit.

[5] *Backtesting* is the process of testing a system on historical data.

4 Evaluation

The prediction framework consists of an ensemble of agents that each provide an own estimation of what the optimal action would be for each point in time, represented by their *mood*. This section gives an overview about the two possible methods to evaluate the performance of an agent: the first one is to check how well the mood values match the target function described in subsubsection 1.2.3; as the target function is a proxy for how suitable the current point in time is for trading, this should give an idea of how well the agent would perform in a live system. The target function is also used during training and thus performing well in this proxy evaluation is the training goal of some agents.

The second method to evaluate the performance of an agent is to run a live simulation and trade based on the agent's decisions. While a lot slower than the artificial evaluation described above, this process represents live trading more closely and thus gives a good insight of how well an agent would perform on a real market. The final test of an agent after having shown promising in-training results, would be to evaluate it in this environment before letting it trade on a live market.
Due to the great care that went into developing the internal API of the framework in such a way that no information can be leaked and the agents only have those pieces of information that would also be available in a live market, this also verifies that the training results were not an effect of incorrect data handling.

4.1 Evaluating the mood against the target function

As introduced in section 2, each agent can provide a mood value that indicates its assumption about the behavior of the exchange rate in the near future. Through the logging capabilities of the virtual market system, it is now possible to evaluate the value of the different agents' mood values post-mortem after a simulation run.

To evaluate the mood values in a comparable way, the target function as described in subsubsection 1.2.3 has been used. The resolution of the logged data is one second. Thus the evaluation was done in one-second resolution as well. During the period used for evaluation, which was distinct from the period used for training, each sample where an agent's absolute mood was above a certain threshold (here: 0.3), the agent's opinion was said to be *correct* when the magnitude of the target function was over the threshold of 5.0 (as discussed in subsubsection 1.2.3) and the sign of the mood was equal to the sign of the target function. Otherwise, the sample was said to be *wrong* for the agent.

The ratio of correct hits for the agent was now defined as its accuracy (see subsubsection 1.2.1) among all samples where the prediction was different from *no_action*. An agent that would randomly guess every sample would be expected to have a ratio of correct hits equal to around one half of the total number of samples where the absolute target function was above the threshold of 5, assuming that both positive and negative peaks occur similarly often. To verify this, an agent with a mood based on randomly drawing values in the set of $(-1,\ 0,\ +1)$ was included in the ensemble while running the virtual market simulation (labeled *dummy_mood*). Additionally, normally distributed mood values where added during the analysis (labeled *X_normal_random_mood*). Two more comparison mood series were included in the analysis: one that is always $+1$ (representing a strategy that always tries to open long positions) and one that is always -1 (similarly representing a strategy that always tries to open short positions), labeled *X_always_buy_mood* and *X_always_sell_mood* respectively. Another agent, here called *MoodAgreement*, based on a combination of three technical agents (CCI, RSI, TSI) was included as well.

The a-posteriori probability of opening a correct trade was used for a significance test (a two-sided binomial test) to see whether the hit ratio of the agents differ significantly from random.

To give an estimation about how volatile the results are, 5th and 95th percentiles were calculated using a technique similar to bootstrapping. However, instead of randomly drawing samples from the whole population, the bootstrap populations were distinct, continuous intervals from the population. This should give a better estimate when dealing with time-series data as successive samples are likely correlated (a comparison test with classical bootstrapping would yield intervals very close to the mean, which probably comes from the high correlation of temporally close samples).

The results are shown in Table 2. It can be seen that while the overall variance of the hit ratios over different periods is very high, there are agents that consistently outperform a random prediction.

Mood variable	Hit ratio	5th perc.	95th perc.	improvement over random
MoodAgreement_mood	42.1%*	0.0%	82.6%	100.5%
TA_CCI_mood	35.1%*	15.9%	51.8%	66.9%
TA_RSI_mood	34.5%*	13.5%	52.6%	64.1%
TA_TSI_mood	32.6%*	19.7%	47.9%	54.9%
TA_StochOsc_mood	25.9%*	20.3%	32.6%	23.5%
X_always_sell_mood	21.2%*	14.7%	27.2%	0.9%
X_normal_random_mood	21.0%	15.3%	26.9%	-0.0%
TA_Renko_mood	20.9%*	14.6%	27.8%	-0.5%
X_always_buy_mood	20.8%*	14.0%	27.6%	-0.9%
dummy_mood	20.6%	13.8%	28.5%	-2.0%

Table 2: This table shows the mean hit ratio of the different agents that provide *mood* values. Dummy agents (starting with an X) were included for comparison purposes. The star marks significant means at a 1% significance level. Bootstrapped 5th and 95th percentiles are included. It can be seen that a few agents significantly outperform a random prediction.

4.2 Evaluating simulated trades

As pointed out before, the only sensible way to evaluate the profitability of a trading system is to either run a simulation of the market using historical data or to connect the system to a real broker and let it trade.

The developed framework allows for both tests. In addition, it saves records of executed trades and the state of the different agents, which can be analyzed post-mortem. Testing an agent in this environment on data coming from a period that the agent had not been trained on should give a good estimation of how well the agent's prediction generalize to new data and how it would perform on real live data.

A set of evaluation scripts has been developed to be able to quickly evaluate the outcome of a simulation run based on the logged data and e.g. see whether the trades that were opened by the framework coincide with the expected range of the target function. To give an comparative example, trades were executed based on three agents based on technical analysis (CCI, RSI, TSI) with a trade being opened when the three agents' moods agree. The normalized confusion matrix is shown in Table 3; it can be seen that this simple ensemble is able of a prediction better than random. However, many trades are executed when the correct action would be to not do anything.

	Result of the prediction	
Ground truth	Long trade	Short trade
No action	28.5%	27.9%
Long trade	12.3%	9.1%
Short trade	9.6%	12.7%

Table 3: The table shows the normalized confusion matrix of a simulated trading run based on three technical agents (CCI, RSI, TSI). A trade was opened when all three agents agreed on the direction of the exchange rate. The total number of trades was 10432. It can be seen that many times a situation where no trade would have been the best choice ("no action" row), the output of the prediction led to a trade. In all situations where a trade was the correct choice, this simple ensemble gets about 60% of the trades correct.

4.3 Further considerations

There are many places were additional work would be required to be able to say that everything has been evaluated with the possible depth and the strictest scientific rigor; however, while everything that was important for the framework to function has been implemented, more time would be required to go into depth in certain areas.

This section will list some open problems, additional ideas for experiments, as well as some low hanging fruits.

4.3.1 Exhaustive hyperparameter optimization

Many components of the framework and many of the implemented agents have parameters that influence their calculations and possibly their general contribution to the solution of the problem. *Hyperparameter optimization* is the (automated) process of finding the best values for these parameters.

While employed in several agents and analyses, there are still many places where parameter values were decided arbitrarily; for example by looking at the typical values from other publications or by looking at a few hand-picked samples.

There are several open problems, however. At the time of writing, the framework logs over a hundred variables for every timestep. Each of those variables is calculated using one or more hyperparameters. Assuming the combinations of only two possible values for every parameter would be tested, there would be more than 2^{100}simulation runs to be done. This is not only unrealistic with the computers and clusters at hand, but is also unlikely to be possible in the future. Thus, a feasible solution would likely optimize the parameters independently from one another or at least in groups, reducing the complexity.

Another problem is the actual evaluation of how much the different values of the hyperparameter influence the final outcome. In the ideal case, a virtual simulation run would be executed with the hyperparameters set to a specific value; the logged data would then be used to train all the primary classifiers, which would then require another market run to be included in the variable logs. Only then a final estimator could be trained to check how much the hyperparameter value improved the final accuracy. Due to the time required to run a market simulation and train the classifiers, this is also unfeasible. A solution would have to be found that could either decide locally (without training the full pipeline) how much the modified parameter affects the outcome; or a solution that requires only a subset of the data for evaluation (see e.g. Klein et al. (2016)).

The hyperparameter search could be either implemented directly into the framework or into a wrapper around the framework that decides on a set of parameters and passes them into the market simulation.

4.3.2 Additional features and agents

A few example agents have been implemented in the framework. The collection is not exhaustive and there are some which are especially promising.

1D convolutional networks Van den oord et al. (2016) recently presented an autoregressive deep neural network utilizing 1D convolutions to predict the next point in the audio waveform of spoken text, *WaveNet*. Convolutional networks have long been used for time-series classification (Homma et al. (1987)) and have since then been generalized to achieve state-of-the-art results in several domains starting with Lecun et al. (1998).

Similar to long short-term memory networks (LSTM) introduced by Hochreiter and Schmidhuber (1997), 1D convolutional networks can learn temporal patterns in the time-series. Thus, implementing and evaluating an agent based on 1D convolutional networks, e.g. with a similar architecture to that of WaveNet, seems like a sensible step.

During the implementation great care has to be taken to make sure that the time steps in the input data are always the same, possibly interpolating data points if necessary.

Genetic programming *Genetic algorithms* have seen extensive use in the prediction of the exchange rate (Deng et al. (2013), Slaný (2009), Rehman et al. (2014), Dempster et al. (2001), Ni and Yin (2009), Ravi et al. (2012)). In a genetic algorithm, an algorithm is encoded in a representation that allows random changes to change the behavior of the algorithm. An initial (often random) set of algorithms is generated and then used to generate a score (in the context of FOREX, the algorithm could be used to trade in a simulation of the market). The scores are then used to select the best-performing algorithms from the set and applying random changes to then repeat the process; to mimic biological evolution, the encodings of different algorithms are mixed in most versions of genetic algorithms as an equivalent to biological reproduction.

One advantage of genetical algorithms in the context of FOREX trading is that the algorithm does not need a differentiable target function (unlike e.g. neural networks and gradient boosting models); the only requirement is a way of calculating a number that represents the goodness of a model. This could for example be the actual profit in a simulation of the market.

An implementation of an agent based on genetic programming could use a selection of the variables generated by the market framework (such as the technical indicators) and utilize a set of different mathematical operations or a decision-tree-like structure to generate an action (such as *buy*, *sell*, or *no action*).

Interesting to see would be if overfitting could be controlled. To the best of my knowledge the only direct controlling parameter to influence overfitting in the case of genetic algorithms would be a limit on model complexity.

Reinforcement learning Another possible experiment is to replace the final estimator with a reinforcement learning based algorithm. This would allow for the manual target function to be replaced; even though the target as described in subsubsection 1.2.3 was shown to be sensible, it is another parameter of the system which might influence the outcome.

Reinforcement learning (possibly based on a neural network as the model) would allow to train a classifier by simulating its action throughout market days. This would additionally allow for the inclusion of open trades as a feature and thus might enable the model to learn when to close trades again, which is an open issue in the current framework.

References

H. Allen and M. P. Taylor, "Charts, Noise and Fundamentals in the London Foreign Exchange Market," *The Economic Journal*, vol. 100, no. 400, p. 49, 1990. [Online]. Available: http://www.jstor.org/stable/2234183?origin=crossref 1.1.1

J. Armstrong and R. Fildes, "On the selection of error measures for comparisons among forecasting methods," *Journal of Forecasting*, vol. 14, no. August 1994, pp. 67–71, 1995. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/for.3980140106/abstract 1.2.1

Bank for International Settlements, *Foreign exchange and derivatives market activity in 2007*, 2007, no. December. 1.1

Bank of International Settlements, "Triennial Central Bank Survey - Foreign Exchange Turnover in April 2013: Preliminary Global Result," *Bank of International Settlements Review*, no. April, p. 24, 2013. [Online]. Available: www.bis.org 1.1

L. Breiman, "Stacked regressions," *Machine Learning*, vol. 24, no. 1, pp. 49–64, 1996. 1.3

P. H. K. Chang and C. L. Osler, "Methodical Madness: Technical Analysis and the Irrationality of Exchange-rate Forecasts," *The Economic Journal*, vol. 109, no. 458, pp. 636–661, oct 1999. [Online]. Available: http://doi.wiley.com/10.1111/1468-0297.00466 1.1.1

T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," mar 2016. [Online]. Available: http://arxiv.org/abs/1603.02754http://dx.doi.org/10.1145/2939672.2939785 2.3

F. Chollet, "Keras," 2015. [Online]. Available: https://github.com/fchollet/keras 3.2

R. T. Clemen, "Combining forecasts: A review and annotated bibliography," *International Journal of Forecasting*, vol. 5, no. 4, pp. 559–583, 1989. 1.3

M. P. Clements and D. F. Hendry, "On the limitations of comparing mean square forecast errors," *Journal of Forecasting*, vol. 12, no. 8, pp. 617–637, dec 1993. [Online]. Available: http://doi.wiley.com/10.1002/for.3980120802 1.2.1

J. G. De Gooijer and R. J. Hyndman, "25 years of time series forecasting," *International Journal of Forecasting*, vol. 22, no. 3, pp. 443–473, jan 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169207006000021 1.2.3, 1.3

M. H. Dempster, T. W. Payne, Y. Romahi, and G. P. Thompson, "Computational learning techniques for intraday FX trading using popular technical indicators." *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 12, no. 4, pp. 744–54, jan 2001. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=935088 4.3.2

S. Deng, K. Yoshiyama, T. Mitsubuchi, and A. Sakurai, "Hybrid Method of Multiple Kernel Learning and Genetic Algorithm for Forecasting Short-Term Foreign Exchange Rates," *Computational Economics*, vol. 45, no. 1, pp. 49–89, nov 2013. [Online]. Available: http://link.springer.com/10.1007/s10614-013-9407-6 4.3.2

E. F. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work," *The Journal of Finance*, vol. 25, no. 2, p. 383, may 1970. [Online]. Available: http://www.jstor.org/stable/2325486?origin=crossref 1.1.1

R. Fildes and S. Makridakis, "The Impact of Empirical Accuracy Studies on Time Series Analysis and Forecasting," *International Statistical Review*, vol. 63, no. 3, pp. 289–308, 1995. [Online]. Available: http://www.jstor.org/stable/1403481 1.1.1

N. A. Gershenfeld, A. S. Weigend, N. A. Gershenfeld, and A. S. Weigend, "The future of time series," *Time Series Prediction: Forecasting the Future and Understanding the Past*, pp. 1–70, 1993. [Online]. Available: www.santafe.edu$\delimiter"026E30F$nftp://161.24.19.221/ele/cairo/CU-CS-670-93.pdf$\delimiter"026E30F$nhttp://econpapers.repec.org/RePEc:wop:safiwp:93-08-053 1.2.1

R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," nov 2014. [Online]. Available: http://arxiv.org/abs/1311.2524 2.2.2

G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks." *Science (New York, N.Y.)*, vol. 313, no. 5786, pp. 504–7, jul 2006. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/16873662 2.2.2

S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, nov 1997. [Online]. Available: http://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735 4.3.2

T. Homma, L. E. Atlas, and R. J. Marks II, "An artificial neural network for spatiotemporal: application to phoneme classification," pp. 31–40, 1987. 4.3.2

E. Hurwitz and T. Marwala, "Common Mistakes when Applying Computational Intelligence and Machine Learning to Stock Market modelling," p. 5, aug 2012. [Online]. Available: http://arxiv.org/abs/1208.4429 1.2.2, 1.2.3, 1.2.3

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets," may 2016. [Online]. Available: http://arxiv.org/abs/1605.07079 4.3.1

M. LeBlanc and R. Tibshirani, "Combining estimates in regression and classification," *Journal of the American Statistical Association*, vol. 91, no. 436, pp. 1641–1650, 1996. 1.3

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=726791 4.3.2

A. W. Lo, "The Adaptive Markets Hypothesis." *Journal of Portfolio Management*, vol. 30, no. 617, pp. 15–29, 2004. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true{&}db=bth{&}AN=31238396{&}site=ehost-live{&}scope=site 1.1.1

Y.-H. Lui and D. Mole, "The use of fundamental and technical analyses by foreign exchange dealers: Hong Kong evidence," *Journal of International Money and Finance*, vol. 17, no. 3, pp. 535–545, 1998. 1.1.1

S. Makridakis, A. Anderson, R. Carbone, R. Fildes, M. Hibon, R. Lewandowski, J. Newton, E. Parzen, and R. Winkler, "The accuracy of extrapolation (time series) methods: results of a forecast competition," *Journal of Forecasting*, vol. 1, pp. 15–19, 1982. 1.3

S. Makridakis and M. Hibon, "The M3-Competition: results, conclusions and implications," *International Journal of Forecasting*, vol. 16, no. 4, pp. 451–476, 2000. 1.3

S. Makridakis, C. Chatfield, M. Hibon, M. Lawrence, T. Mills, K. Ord, and L. F. Simmons, "The M2-competition: A real-time judgmentally based forecasting study," *International Journal of Forecasting*, vol. 9, no. 1, pp. 5–22, 1993. 1.3

L. Molgedey and H. G. Schuster, "Separation of a mixture of independent signals using time delayed correlations," *Physical Review Letters*, vol. 72, no. 23, pp. 3634–3637, jun 1994. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevLett.72.3634 2.2.3

H. Ni and H. Yin, "Exchange rate prediction using hybrid neural networks and trading indicators," *Neurocomputing*, vol. 72, no. 13-15, pp. 2815–2823, aug 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231209001210 4.3.2

P. Patel, N. Patel, and A. Patel, "Factors affecting Currency Exchange Rate, Economical Formulas and Prediction Models," *International Journal of Application or Innovation in Engineering & Management*, vol. 3, no. 3, pp. 53–56, 2014. [Online]. Available: http://ijaiem.org/volume3issue3/IJAIEM-2014-03-05-013.pdf 1.1.1

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011. 3.2

G. Pe/rez-Herna/ndez, F. Paul, T. Giorgino, G. De Fabritiis, and F. Noe/, "Identification of slow molecular order parameters for Markov model construction," *The Journal of Chemical Physics*, vol. 139, no. 1, p. 015102, 2013. [Online]. Available: http://scitation.aip.org/content/aip/journal/jcp/139/1/10.1063/1.4811489 2.2.3

V. Ravi, R. Lal, and N. R. Kiran, "Foreign Exchange Rate Prediction using Computational Intelligence Methods," vol. 4, pp. 659–670, 2012. 4.3.2

M. Rehman, G. M. Khan, and S. A. Mahmud, "Foreign Currency Exchange Rates Prediction Using CGP and Recurrent Neural Network," *IERI Procedia*, vol. 10, pp. 239–244, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212667814001312 4.3.2

S. Röblitz and M. Weber, "Fuzzy spectral clustering by PCCA+: application to Markov state models and data classification," *Advances in Data Analysis and Classification*, vol. 7, no. 2, pp. 147–179, jun 2013. [Online]. Available: http://link.springer.com/10.1007/s11634-013-0134-6 2.2.4

K. Slaný, "Towards the automatic evolutionary prediction of the FOREX market behaviour," *Proceedings of the 2009 International Conference on Adaptive and Intelligent Systems, ICAIS 2009*, pp. 141–145, 2009. 4.3.2

C. Song, F. Liu, Y. Huang, L. Wang, and T. Tan, "Auto-encoder Based Data Clustering," *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications SE - 15*, vol. 8258, pp. 117–124, 2013. [Online]. Available: http://link.springer.com/10.1007/978-3-642-41822-8 2.2.2

J. H. Stock and M. W. Watson, "Combination forecasts of output growth in a seven-country data set," *Journal of Forecasting*, vol. 23, no. 6, pp. 405–430, sep 2004. [Online]. Available: http://doi.wiley.com/10.1002/for.928 1.3

A. Van den oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," sep 2016. [Online]. Available: http://arxiv.org/abs/1609.03499 4.3.2

B. P. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products," *Technometrics*, vol. 4, no. 3, p. 419, aug 1962. [Online]. Available: http://www.jstor.org/stable/1266577?origin=crossref 2

D. H. Wolpert, "Stacked Generalization," *Neural Networks*, vol. 87545, no. 505, 1992. 1.3

M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I.* Springer International Publishing, 2014, pp. 818–833. [Online]. Available: http://link.springer.com/10.1007/978-3-319-10590-1{_}53 2.2.2