BIELEFELD UNIVERSITY
FACULTY OF TECHNOLOGY
Cognitive Computer Science

BACHELOR THESIS

Machine Learning for Automated Stock Trading

Author:
Timo Mechsner

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

supervised by

First Auditor	Second Auditor
Prof. Dr. Barbara Hammer	Dr. Alexander Schulz

September 11, 2018

Statement of authorship

I hereby certify that this thesis has been composed by me and is based on my own work unless stated otherwise. No other persons work has been used without due acknowledgment in this thesis. All references and verbatim extracts have been quoted, and all sources of information, including graphs and data sets, have been specifically acknowledged. This thesis has not been presented to an examination office in the same or a similar form yet.

Bielefeld, September 11, 2018

Timo Mechsner

Contents

1 Introduction

Financial markets are highly complex structures. Stock prices are indirectly influenced by a great number of external factors that may share highly non-linear, non-transparent correlations. Although chaotic seeming price movements appear unpredictable and this intuition is even being supported by theoretical findings, the number of frameworks, guidelines and strategies promising increased stock trading success keeps growing.

With the rise of computers with ever increasing computing power, efforts of automating stock trading strategies are growing as well. Algorithms are capable of detecting complex relationships in high-dimensional spaces much better than any human. They promise to be a highly profitable tool when applied to economical and financial data. Complex artificial neural networks have been created with the purpose of finding a way to predict stock price movements, thereby increasing investment returns. However, research and experience in the machine learning field have shown that sophisticated models often, if at all, perform only marginally better than algorithms that are simple but powerful.

This thesis will thus focus on two widely used classification algorithms named k-Nearest Neighbors (k-NN) [1] and Random Forest (RF) [2] and analyze if their performance in a stock trading scenario can be improved by extensions addressing two certain properties that stock data are assumed to possess: Firstly, stock price data might be subject to various forms of concept drift, since investment principles and market psychology are not static but always evolving. For both k-NN and RF there exist enhancements that enable these models to capture and adapt to concept drift. Secondly, as stated before, stock prices are influenced by complex correlations that might not be captured by the model. Those influences will thus be perceived as random disturbances, reducing the certainty of predictions. Acting according to highly uncertain predictions is risky as it might result in a loss of capital in case the prediction was wrong. To avoid this it would be beneficial to recognize uncertain predictions. This issue can be addressed by confidence scores that prevent the trader from trusting uncertain signals.

One of the frameworks for financial analysis mentioned above is the Technical Analysis (TA) [3]. It solely utilizes historical price data to make predictions about future price movements. This framework yields a number of features, so-called Technical Indicators (TI), that are widely used by professional stock traders, yet easy to calculate. For this thesis, a subset of these Technical Indicators has been selected to be used as features of single stocks.

In a stock trading simulation the classification algorithms have been used to trade single stocks with virtual cash considering realistic transaction costs. The stocks are or have been constituents of the S&P 500 stock index. Subsequently, profit and classification performance of the different algorithms have been evaluated and compared.

The remainder of this thesis is structured as follows: chapter 2 gives a brief overview of the related research that has been done in the field of stock data prediction. Furthermore, it provides a general explanation of the classification algorithms that have been used in this work and briefly introduces Technical Analysis and some important Technical Indicators. Subsequently, chapter 3 explains the selection of data and features. It describes the setup of the simulation and evaluates the results. Finally, conclusions are drawn in chapter 4 and an outlook is provided.

2 Background

This thesis focuses on two widely used classification algorithms called k-Nearest Neighbors and Random Forest. They will be compared to extensions of these algorithms that are capable of adapting to heterogeneous concept drift. Another comparison will be made with modifications that are capable of rejecting unsure predictions. The following chapter provides background information for the experiment conducted as part of this thesis. After a short introduction to related research the next sections describe the basic ideas of concept drift and the two algorithms with their corresponding extensions. Additionally, it gives a short introduction to Technical Analysis.

2.1 Related Work

In the 1980's there has been first research on stock market prediction. Despite the low interest in artificial intelligence and machine learning during the so called AI winter, Halbert White used artificial neural networks to analyze stock prices [4]. In the following years, various machine learning techniques have been applied in portfolio optimization and stock price prediction, using a broad range of different approaches and features like sentiment analysis in social media, analysis of fundamental economical data of companies or just pure price data [5][6]. Among these approaches are many highly complex algorithms as well as simple ones. One example of the latter group is the k-NN algorithm, which has been frequently used for stock price analysis. It proved to compete quite well against more sophisticated classification algorithms like probabilistic neural networks [7]. It has also been used as part of a heterogeneous architecture for forecasting single features before using a more complex algorithm for classification based on these predictions [8].

Main inspiration for this thesis was a study by Teixeira and De Oliveira about the profitability of a k-NN-based trader based on Technical Indicators with realistic transaction costs [9]. The trader has been compared to a buy&hold strategy, a classical financial baseline strategy. They criticize that most research considers prediction performance as quality criterion, rather than the generated return under realistic conditions. Their evaluation approach showed that exceptionally low error rates are not necessary for achieving reasonable profit.

2.2 Concept Drift

In many real-world applications, like stock prices, data is available as a stream only. Data points are not available all at once but arrive one after another while their number is potentially infinite. In such a setting concept drift might occur. This term refers to changes in the underlying data generating process, leading to a slow or sudden change in statistical properties of the data stream [10]. It may thus lead to a sudden or slowly emerging reduction in quality of results of a stream processing application, like a classification algorithm.

Formally, concept drift can be characterized as a change over time in the joint distribution of features $\boldsymbol{x} \in \mathbb{R}^n$ and target variable $y \in \{1, ..., c\}$:

$$\exists \boldsymbol{x} : P_{t_0}(\boldsymbol{x}, y) \neq P_{t_1}(\boldsymbol{x}, y) \tag{2.1}$$

where

$$P_t(\boldsymbol{x}, y) = P_t(\boldsymbol{x}) P_t(y|\boldsymbol{x}) \tag{2.2}$$

Concept drift can be classified by different properties regarding the speed of change and the part of the distribution in which the actual drift occurs.

Virtual	The change occurs in the distribution of the features $P_t(\boldsymbol{x})$ without affecting the posterior distribution of the labels. For example, certain market situations have become rare, but when they occur, stocks still behave like they did before the drift.
Real	The change occurs in the relation between features and posterior distribution of the labels $P_t(y|\boldsymbol{x})$. For example, due to changing market conditions, certain types of investment are not considered safe anymore.
Reoccuring	Concepts reappear multiple times after they disappeared due to a shift in distributions, like seasonal effects.
Incremental	The change is a slowly emerging shift in distributions. For example, a new market altering financial product is slowly gaining popularity.
Abrupt	The change is a sudden shift in distributions. For example, investor behavior changes drastically as a consequence of a market crash.

To resolve this issue, many new algorithms and modifications of established algorithms have been proposed. They can be classified as active or passive.

Active	The algorithm uses a method for actively detecting concept drift, for example by monitoring statistical features of the data or the classification accuracy of the model. Usually, knowledge gathered up to the point of detection is discarded.
Passive	The model is updated incrementally with the latest data. There is no need for actively detecting drift.

2.3 k-Nearest Neighbors Estimator

2.3.1 Standard k-NN

The k-Nearest Neighbors estimator (k-NN) [1] is a non-parametric, instance-based algorithm that can be used for classification and regression. Since this thesis will focus on classification, the following description will omit the k-NN regression. Despite its simplicity, k-NN often performs quite well and thus enjoys high popularity among machine learning practitioners.

Learning for this estimator means storing all available training data. Unlabeled inputs are then classified by a majority vote of their k closest neighbors among the training data:

$$kNN(\boldsymbol{x}) = \underset{\hat{c}\in\{1,\ldots,c\}}{\arg\max} \sum_{\boldsymbol{x_i}\in N_k(\boldsymbol{x})} \delta_{y_i\hat{c}} \tag{2.3}$$

where $N_k(\boldsymbol{x})$ yields the k nearest neighbors of $\boldsymbol{x}$, according to some distance measure, in the set of training data $Z = \{(\boldsymbol{x_i}, y_i) \in \mathbb{R}^D \times \{1, ..., c\} | i = 1, ..., n\}$ and δ_{ij} is the Kronecker delta:

$$\delta_{ij} \mapsto \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

The only parameters to be chosen by the user are the constant k, determining the size of the neighborhood, and the distance metric used to find the k nearest neighbors. As illustrated in figure 2.1, the choice of the neighborhood size can be critical. Throughout this thesis the euclidean distance $d(\boldsymbol{x_1}, \boldsymbol{x_2}) = \|\boldsymbol{x_1} - \boldsymbol{x_2}\|_2$ is used.

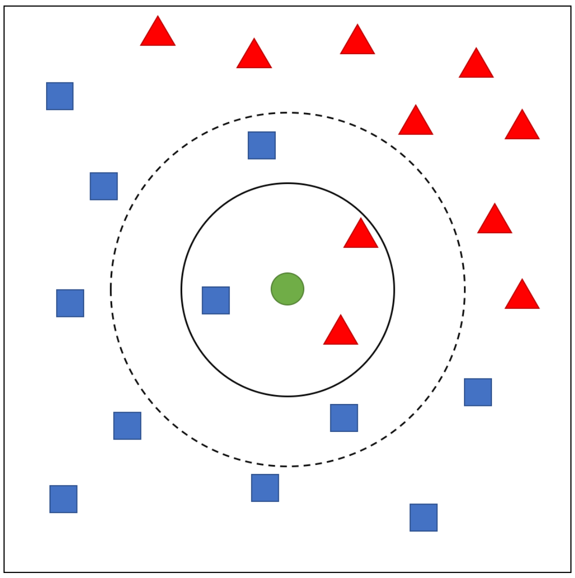

Figure 2.1: For $k = 3$ (solid line) the k-Nearest Neighbors estimator would assign the green point to the triangle class. However, for $k = 5$ (dashed line) it would be assigned to the square class.

2.3.2 Weighted k-NN

A variation of the standard k-NN decision rule is the weighted k-NN [11]. This method weights the vote of the k nearest neighbors according to their distance to the input data point to put more emphasis on closer points:

$$wkNN(\boldsymbol{x}) = \underset{\hat{c}\in\{1,\dots,c\}}{\arg\max} \sum_{\boldsymbol{x_i}\in N_k(\boldsymbol{x})} w_i(\boldsymbol{x})\delta_{y_i\hat{c}} \tag{2.5}$$

where $w_i(\boldsymbol{x})$ describes the weight of the influence of $\boldsymbol{x}_i$ on the vote:

$$w_i(\boldsymbol{x}) = \frac{1}{d(\boldsymbol{x}, \boldsymbol{x_i})} \tag{2.6}$$

Here, $d(\boldsymbol{x}, \boldsymbol{x_i})$ again denotes a distance metric, in this case the euclidean metric.

2.3.3 Confidence Scores in k-NN

A confidence score for k-NN can be derived from the viewpoint of probability density estimation [12], as k-NN is also a means of probability density estimation at a given point $\boldsymbol{x}$:

$$p(\boldsymbol{x}) = \frac{K}{NV} \tag{2.7}$$

where K is the number of neighbors to be considered, N is the total number of points in the data set and V is the volume of a sphere centered on $\boldsymbol{x}$ and containing all K neighbors. This rule can be applied separately to every class $\hat{c} \in \{1, ..., c\}$ present in the data:

$$p(\boldsymbol{x}|\hat{c}) = \frac{K_{\hat{c}}}{N_{\hat{c}}V} \tag{2.8}$$

where $K_{\hat{c}}$ denotes the number of points within the sphere that belong to class $\hat{c}$ and $N_{\hat{c}}$ denotes the total number of points belonging to that class. The prior probability of the classes is their relative frequency:

$$p(\hat{c}) = \frac{N_{\hat{c}}}{N} \tag{2.9}$$

Using the Bayes' theorem for combining the unconditional density (equation 2.7), the class density (equation 2.8) and the class priors (equation 2.9) yields the posterior probability of class membership:

$$p(\hat{c}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\hat{c})p(\hat{c})}{p(\boldsymbol{x})} = \frac{K_{\hat{c}}}{K} \tag{2.10}$$

This probability can be used to express how confident the classifier is about assigning a data point $\boldsymbol{x}$ to a class $\hat{c}$.

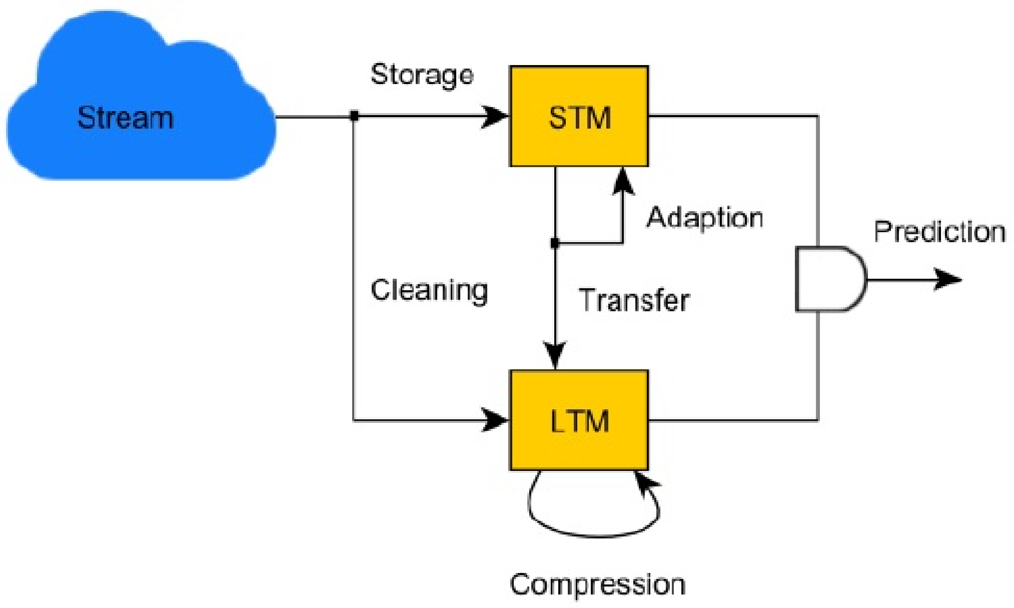

Figure 2.2: Newly arriving data points are fed into the STM and the LTM. When the STM exceeds its capacity limit, its size will be reduced and the stripped-off information is transferred to the LTM. There, the data will be compressed. For prediction, information of all memories can be considered. (Graphic source: [13])

2.3.4 Self-Adjusting Memory k-NN

Self-adjusting memory k-NN (SAM) is a classification algorithm capable of passively handling heterogeneous concept drift [13]. It features a biologically inspired two-stage memory model consisting of a long-term memory (LTM) and a short-term memory (STM). The STM has a lower capacity than the LTM. Its contained data set M_{ST} is a dynamic sliding window always containing the latest m observations of the data stream:

$$M_{ST} = \{(\boldsymbol{x_i}, y_i) \in \mathbb{R}^n \times \{1, ..., c\} | i = t - m + 1, ..., t\} \tag{2.11}$$

In contrast, the data set M_{LT} of the LTM contains a compressed form of all observations of the data stream that do not contradict the current concept represented by M_{ST}:

$$M_{LT} = \{(\boldsymbol{x_i}, y_i) \in \mathbb{R}^n \times \{1, ..., c\} | i = 1, ..., p\} \tag{2.12}$$

Additionally, SAM uses a third memory M_C with $p + m$ elements, comprising both STM and LTM:

$$M_C = M_{ST} \cup M_{LT} \tag{2.13}$$

For each memory there is a weighted k-NN estimator operating on the corresponding data set. When a new data point arrives, it will be classified by the estimator that performed best on recent data. This performance is captured by weights w_{ST}, w_{LT} and w_C for the

STM, the LTM and the combined memory, respectively.

$$\mathbf{x} \mapsto \begin{cases} wkNN_{M_{ST}}(\boldsymbol{x}), & \text{if } w_{ST} \geq \max(w_{LT}, w_C) \\ wkNN_{M_{LT}}(\boldsymbol{x}), & \text{if } w_{LT} \geq \max(w_{ST}, w_C) \\ wkNN_{M_C}(\boldsymbol{x}), & \text{if } w_C \geq \max(w_{ST}, w_{LT}) \end{cases} \tag{2.14}$$

For the case of weight equality, a priority order is defined on the estimators: $wkNN_{M_{ST}}(\cdot)$, $wkNN_{M_{LT}}(\cdot)$, $wkNN_{M_C}(\cdot)$. A schematic of the SAM architecture is given in figure 2.2. Every time a new data point is presented to the algorithm, the STM, the LTM and the weights have to be updated.

STM Update

The STM is updated to represent available knowledge about the most recent concept in an optimal way. Therefore, after a new data point has been inserted, the new memory $M_{ST_{t+1}}$ is formed by dropping old data points that are not consistent with the current concept. This is achieved by adjusting the window size to optimize the Interleaved Test-Train error of the STM. The upper limit of the window is always fixed on the most recent data point. Discarded data are denoted as $O_t = M_{ST_t} \setminus M_{ST_{t+1}}$.

LTM Update

The LTM contains all previously gathered information that does not contradict the current concept represented by the STM. Therefore, a cleaning process ensures consistency of two data sets with regard to one specific data point:

$$clean\colon (A, B, (\boldsymbol{x_i}, y_i)) \mapsto \hat{A} \tag{2.15}$$

Within a threshold distance θ, remove all points from A that are in the specified point's k-neighborhood and of a different class:

$$\hat{A} = A \setminus \{(\boldsymbol{x_j}, y(\boldsymbol{x_j})) \mid \boldsymbol{x_j} \in N_k(\boldsymbol{x_i}, A), d(\boldsymbol{x_j}, \boldsymbol{x_i}) \leq \theta, y(\boldsymbol{x_j}) \neq y_i\} \tag{2.16}$$

The threshold distance θ is defined as the maximum distance of the specified point $\boldsymbol{x_i}$ to a point within its k-neighborhood in B with the same class y_i:

$$\theta = \max\{d(\boldsymbol{x_i}, \boldsymbol{x}) \mid \boldsymbol{x} \in N_k(\boldsymbol{x_i}, B \setminus (\boldsymbol{x_i}, y_i)), y(\boldsymbol{x}) = y_i\} \tag{2.17}$$

Furthermore, the cleaning process is iteratively defined for a whole set instead of just one point:

$$clean\colon (A, B) \mapsto \hat{A}_{|B|} \tag{2.18}$$

where

$$\begin{aligned} \hat{A}_0 &= A \\ \hat{A}_{t+1} &= clean(\hat{A}_t, B, (\boldsymbol{x_{t+1}}, y_{t+1})) \end{aligned} \tag{2.19}$$

This cleaning process is then applied to the LTM with regard to STM for every new observation $(\boldsymbol{x_t}, y_t)$, and to the whole set M_{ST_t}, whenever the STM is shrunk:

$$\tilde{M}_{LT_t} = clean(M_{LT_t}, M_{ST_t}, (\boldsymbol{x_t}, y_t)) \tag{2.20}$$

$$M_{LT_{t+1}} = \tilde{M}_{LT_t} \cup clean(O_t, M_{ST_{t+1}}) \tag{2.21}$$

LTM Compression

Whenever the maximum overall storage capacity L_{max} is reached, memory needs to be freed. However, as the goal of SAM is to ensure lifelong learning, old knowledge must not be discarded. Instead, the knowledge in the LTM is compressed. This is achieved by applying the k-means++ clustering algorithm [14] on each class separately. K-means++ is a vector quantization method that aims to partition a data set of size n into $k < n$ clusters, minimizing the intra-cluster variance. Each of the clusters is represented by a prototype placed at the mean of the points assigned to it. For compression of the LTM, the number of partitions k is chosen as half the number of points assigned to the considered class. Afterwards, the points of each class in the LTM are replaced by the cluster prototype they have been assigned to. Through this approach the size of the LTM can be halved without loosing much knowledge.

Weight Update

Finally, as a part of the update process, the weights w_{ST}, w_{LT} and w_C have to be adjusted. They are equal to the average accuracy of the corresponding model over the last m_t time steps, where $m_t = |M_{ST_t}|$ is the current STM size:

$$w_S^t = \frac{|\{i \in \{t - m_t + 1, ..., t\} \mid wkNN_{S_i}(\boldsymbol{x_i}) = y_i\}|}{m_t} \tag{2.22}$$

with $S \in \{M_{ST}, M_{LT}, M_C\}$.

Hyper Parameter Choice

The SAM algorithm has three adjustable hyper parameters: neighborhood size k, the minimum STM size L_{min} and the maximum overall storage capacity L_{max}. These hyper parameters do not require optimization, as they can be robustly chosen independent of the specific application. The proposed values are $k = 5$, $L_{min} = 50$ and $L_{max} = 5000$.

2.4 Random Forest Estimator

2.4.1 Decision Trees

A Decision Tree is a rooted and strictly binary tree T, describing a hierarchical axis-aligned partitioning of the feature space $\mathbb{R}^D$, $D \in \mathbb{N}$, and a prediction rule [15]. The tree consists of a finite set of nodes j that have exactly one parent node and either two

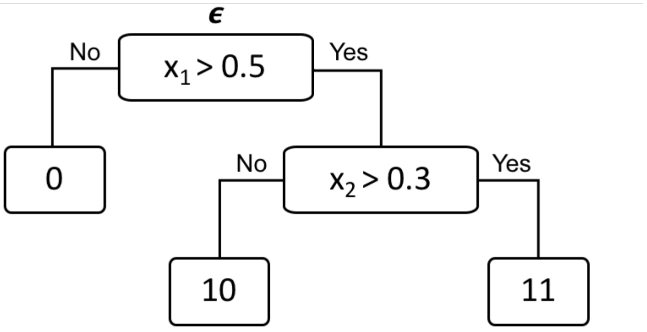

Figure 2.3: A simple Decision Tree with three leaves. The two internal nodes produce a division of the feature space.

child nodes or no child nodes at all. Only the root node ϵ has no parent node. All nodes without child nodes are called leaves and denoted by $leaves(T)$. The children of a node are denoted by $left(j)$ and $right(j)$. A simple sample tree is visualized in 2.3.

All nodes $j \in T$ are associated with disjoint regions $A_j \subset \mathbb{R}^D$ in the feature space, like in figure 2.4. The root node's region A_ϵ spans the whole feature space. Subsequently, each internal node describes a split of its parent node's region. This split is characterized by the split axis $\delta_j \in \{1, ..., D\}$ and a split location $\xi_j \in \mathbb{R}$:

$$B_{left(j)} = \{\boldsymbol{x} \in B_j | x_{\delta_j} \leq \xi_j\} \tag{2.23}$$

$$B_{right(j)} = \{\boldsymbol{x} \in B_j | x_{\delta_j} > \xi_j\} \tag{2.24}$$

The Decision Tree is characterized by the tuple $\mathcal{T} = (T, \boldsymbol{\delta}, \boldsymbol{\xi})$. To predict the label of a data point $\boldsymbol{x}$, the tree is traversed to identify the region $A_{j_{\boldsymbol{x}}}$ containing that data point. Tree traversal is defined recursively:

$$traverse_{\mathcal{T},j}(\boldsymbol{x}) = \begin{cases} j, & \text{if } j \in leaves(T) \\ traverse_{\mathcal{T},left(j)}(\boldsymbol{x}), & \text{if } x_{\delta_j} \leq \xi_j \\ traverse_{\mathcal{T},right(j)}(\boldsymbol{x}), & \text{if } x_{\delta_j} > \xi_j \end{cases} \tag{2.25}$$

The final leaf of the traversal is denoted $leaf_{\mathcal{T}}(\boldsymbol{x}) = traverse_{\mathcal{T},\epsilon_{\mathcal{T}}}(\boldsymbol{x})$. When this leaf is reached, the data point is predicted to belong to the most likely class label according to that leaf:

$$g_{\mathcal{T}}(\boldsymbol{x}) = \underset{\hat{c} \in \{1,...,c\}}{\arg\max} \sum_{\boldsymbol{x_i} \in A_{leaf_{\mathcal{T}}(\boldsymbol{x})}} \delta_{y_i \hat{c}} \tag{2.26}$$

where δ_{ij} is the Kronecker delta and $\boldsymbol{x_i} \in A_{leaf_{\mathcal{T}}(\boldsymbol{x})}$ are all points from the training data set $Z = \{(\boldsymbol{x_i}, y_i) \in \mathbb{R}^D \times \{1, ..., c\} | i = 1, ..., n\}$ contained in the region associated with the leaf $leaf_{\mathcal{T}}(\boldsymbol{x})$ of Decision Tree $\mathcal{T}$.

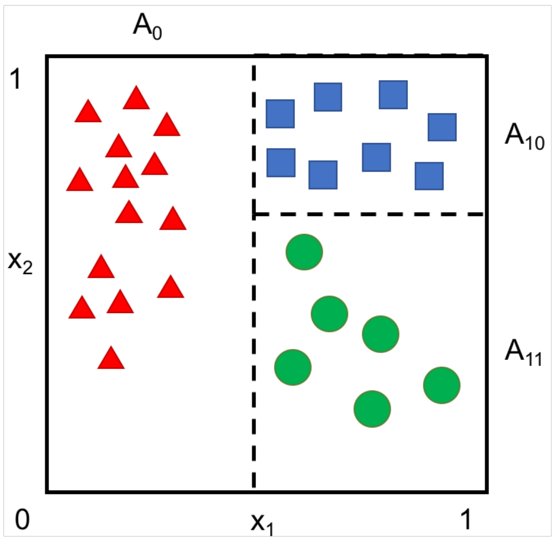

Figure 2.4: A Decision Tree divides the whole feature space into three disjunct axis-aligned partitions.

Training a Decision Tree

There are many different approaches to generating a Decision Tree. An overview can be found in the history of classification and regression trees by Loh [16]. In this work, the CART algorithm is used [15]. In the CART method the split-axis is chosen to minimize class mixture (impurity). This is achieved by choosing a split that minimizes the GINI index of the two new child nodes:

$$GINI(j) = 1 - \sum_{\hat{c}=1}^{c} p(y = \hat{c} | \boldsymbol{x} \in A_j) \tag{2.27}$$

with $(\boldsymbol{x}, y) \in Z$, Z denoting the training data set. The probability of a class in a given region is the relative frequency of training data points in that region belonging to the given class:

$$p(y = \hat{c} | \boldsymbol{x} \in A_j) = \frac{1}{N_j} \sum_{\boldsymbol{x_i} \in A_j} \delta_{y_i \hat{c}} \tag{2.28}$$

where N_j denotes the number of training examples $\boldsymbol{x_i}$ inside the given region A_j and $(\boldsymbol{x_i}, y_i) \in Z$.

A common criterion for stopping tree growth in a branch is to define a minimum number $a_{\min}$ of training examples $\boldsymbol{x_i}$ that need to be inside the leaf's region A_j. If after a split the number of examples inside the new node's region would be lower than $a_{\min}$, the split is not executed and the previous node becomes a leaf. Tree growth is also stopped when all training points inside a node's region belong to the same class.

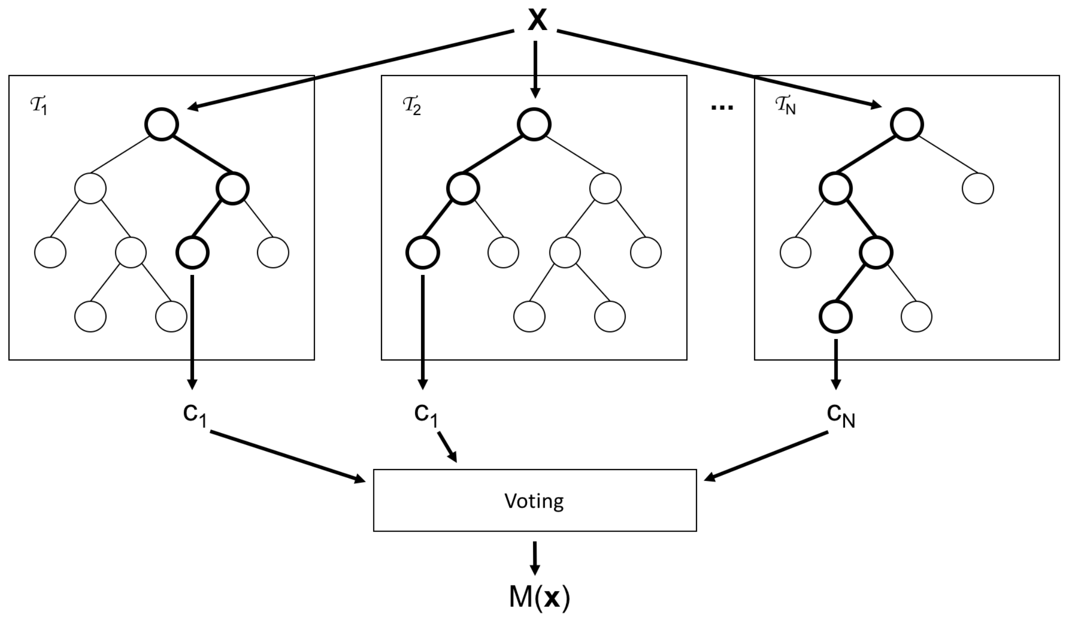

Figure 2.5: A Random Forest is an ensemble of N modified Decision Trees. The final prediction $M(\boldsymbol{x})$ is established by a vote among the predictions of all trees.

2.4.2 Ensembles of Decision Trees

A Random Forest classifier [2] utilizes an ensemble of Decision Trees to make predictions. The trees are generated using bootstrap aggregation (bagging) [17], meaning that every tree is generated using a subset of the available training data which is selected at random with replacement. The Decision Tree training itself is altered as well. Firstly, the most informative feature for node split is not selected among all features but among a random subset of features selected for that tree. Secondly, all trees are grown to the maximum size. This narrowing of available information makes every Decision Tree an 'expert' for the specific subset of the data and features on which it was trained, leading to an increased diversity of base classifiers which ultimately results in more robust predictions. Each of these base classifiers produces a binary indicator variable describing its vote for one class. The final decision is made by a majority vote among all votes:

$$M(\boldsymbol{x}) = \underset{\hat{c}\in\{1,\dots,c\}}{\arg\max} \sum_{i=1}^{N} \delta_{\hat{c},g_{\mathcal{T}_i}(\boldsymbol{x})} \tag{2.29}$$

where $g_{\mathcal{T}_i}(\cdot)$, $i \in \{1, ..., N\}$ denotes the prediction function of the N Decision Trees of the ensemble. Two hyper parameters are tuned for the application of Random Forests in this work: The number of trees to generate and the maximum number of features to be used in a Decision Tree. The latter is usually chosen as $\sqrt{D}$ with D denoting the number of features.

2.4.3 Confidence Scores in Random Forests

The confidence of a Random Forest prediction for a data point $\boldsymbol{x}$ can be computed as the average probability of the predicted class $\hat{c} = M(\boldsymbol{x})$ among the trees in the forest:

$$p(\hat{c}|\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} p(y = \hat{c}|\mathcal{T}_i) \tag{2.30}$$

The class probability of a single tree $p(y = \hat{c}|\mathcal{T}_i)$ is the relative frequency of training examples $\boldsymbol{x_i}$ in the region $A_{j_{\boldsymbol{x}}}$ of Decision Tree $\mathcal{T}_i$ belonging to the class $\hat{c}$, as defined in equation 2.28.

2.4.4 Adaptive Random Forests

Adaptive Random Forest [18] is a classifier for evolving data streams that is based on Random Forest. It uses an active warning and drift-handling algorithm (in this work: ADWIN [19]) to watch for concept drift on each tree separately. When a drift warning occurs for a tree, a new tree starts growing in parallel. If the warning escalates to a drift, the old tree is replaced with the new one.

Instead of growing the Decision Trees in sequence, an Adaptive Random Forest grows them in parallel, during only one pass over the training data set. Each point is assigned a weight or multiplicity for each tree. Weighting a training example $\boldsymbol{x}$ with a value w is equivalent to training the tree w times with that data point. In each tree $\mathcal{T}_i$, $i \in \{1, ..., N\}$, of the N trees in the forest, the training data of the corresponding leaf $leaf_{\mathcal{T}_i}(\boldsymbol{x})$ is extended by w instances of $\boldsymbol{x}$. If the number of encountered instances of a leaf exceeds a certain limit (grace period, GP) and none of the stopping criteria mentioned in section 2.4.1 is applicable, the node will be split and two child nodes will be generated. Another difference to Random Forest is that for prediction the votes of the trees are not evaluated according to a majority vote. Instead, at time step t the trees' classification accuracies over the instances they have seen since their last reset are considered:

$$M(\boldsymbol{x}_t) = \underset{\hat{c} \in \{1,...,c\}}{\arg\max} \sum_{i=1}^{N} w^t_{\mathcal{T}_i} \delta_{\hat{c}, g_{\mathcal{T}_i}(\boldsymbol{x}_t)} \tag{2.31}$$

with weights:

$$w^t_{\mathcal{T}_i} = \frac{|\{j \in \{t - n_{l,\mathcal{T}_i} + 1, ..., t\} \mid g_{\mathcal{T}_i}(\boldsymbol{x_j}) = y_j\}|}{n_{l,\mathcal{T}_i}} \tag{2.32}$$

where $n_{l,\mathcal{T}_i}$ denotes the number of instances $\boldsymbol{x}_j$ the tree $\mathcal{T}_i$ has seen since its last reset. This calculation is equivalent to the weight calculation in SAM defined in equation 2.22.

After classifying a new data point $\boldsymbol{x}_t$ observed on the data stream at time step t, the weights are updated (equation 2.32) and the trees are trained on $\boldsymbol{x}_t$ like during training (as described above). Then the drift detecting algorithm is used to check for drift warnings and drift occurrences among all trees. A warning is emitted if the detector spots a drift on tree $\mathcal{T}_i$ with a confidence above a given warning-threshold δ_w. In this case a new tree will be created and added to the set of background trees, if there is no background tree associated with $\mathcal{T}_i$ yet. If the detector spots a drift with confidence above a given drift-threshold $\delta_d > \delta_w$, the emitting tree $\mathcal{T}_i$ will be replaced with its associated background tree. That tree is then removed from the background tree set. Finally, all background trees are trained on the new observation $\boldsymbol{x}_t$.

In addition to the Random Forest hyper parameters, the Adaptive Random Forest has two parameters for controlling the drift detection method sensitivity for drift warnings and drift occurrences.

2.5 Technical Analysis

2.5.1 Technical Analysis and the Efficient Market Hypothesis

The goal of Technical Analysis (TA) is to recognize patterns in stock price movements [3]. It describes features, so-called Technical Indicators (TI), used to predict future price movements based on historical price and trading volume time series. It is rooted in Dow Theory, that was thought of by Charles Dow to recognize and describe major market movements [20]. However, academia is very skeptical about the use of TA. This skepticism is well expressed in the Efficient Market Hypothesis, which suggests that all information becoming available is being collected and analyzed by thousands of investors immediately and reflected in the price [21], [22]. Accordingly, there would be no knowledge contained in historical data that can be used to increase returns. This would make a prediction of the market impossible.

In contrast to this, TA is in fact being used in practice. An international study conducted by Menkhoff with 692 fund managers in five different markets shows that TA is being used by a vast majority of professional fund managers to some degree. For them, it becomes more important for shorter-term horizons, such as weeks [23]. The broad usage of TA among this group, which has evolved as the most important group in modern financial markets [24], implies successful investments. This contradicts the Efficient Market Hypothesis, as it suggests that historical price data does contain knowledge about future market movements. According to Menkhoff, users of TA believe that prices are determined by psychological influences. In their view, a trend-following behavior would thus be beneficial [23]. This view is supported by research like the work of Caginalp and Laurent, who discovered a statistically significant prediction capability of stock prices [25]. These findings complement other modern market theory approaches that criticize the Efficient Market Hypothesis [26], [27].

Stock traders utilizing TA are provided with a wide range of TIs that can be analyzed. For this study, a selection of TI has been chosen with respect to popularity of the indicator and ease of implementation. TIs have not been invented for automated analysis but for manual search. Thus, some of these indicators represent geometrical structures that are easy to assess with the human eye but rather difficult to detect algorithmically [3], [20].

Furthermore, the present selection of indicators represents different categories, namely moving averages and oscillators. The following TI definitions and analytic approaches are described according to Murphy [20], if not stated differently. In all definitions $c_s(t)$, $l_s(t)$ and $h_s(t)$ are the closing price (last price before markets close), lowest price and highest price of stock s on day t.

2.5.2 Moving Averages

Moving averages are being used to smooth price movements and to eliminate noise. The simple moving average (SMA) used in this thesis is calculated on every trading day to form a curve that makes it easier to identify trends. The SMA is defined as follows:

$$m_{s,N}(t) = \frac{1}{n} \sum_{i=0}^{N} c_s(t-i) \tag{2.33}$$

with closing prices $c_s(i)$ on corresponding day i during a period of N days before day t. To generate trade signals two moving averages, a shorter and a longer one, can be used together, like in the double crossover method: when the shorter one crosses above the longer average, a buy-signal is produced and vice versa. Popular combinations for the moving averages are five and 20 days or ten and 50 days.

2.5.3 Oscillators

Oscillators are being used to identify momentum. When an oscillator reaches an upper extreme this is a signal for the market to be overbought, meaning that prices are likely to fall as there has been too much buying activity. In this case the oscillator generates a sell-signal. An oscillator reaching a lower extreme in turn signals an oversold market that is likely to rise, thus interpreted as a buy-signal.

Bollinger Bands

Developed by John Bollinger, this technique uses two values placed above and below a 20-day SMA, respectively. Both of these values are set in a distance of two standard deviations above and below the SMA. Standard deviation is used to measure dispersion of values around the mean of the set during the given period:

$$\delta_N = \sqrt{\frac{\sum_{i=1}^{N} |x_i - \overline{x}|^2}{N}} \tag{2.34}$$

When the price hits the upper band this is considered a signal for an overbought market, whereas it is interpreted as a signal for an oversold market, when the price hits the lower band. To handle the relation of the closing price to the upper and lower bands a min-max scaling is applied, yielding the %B oscillator defined in equation 2.35. It covers a continuous range fixed by two points: If the closing price hits the upper Bollinger Band the feature's value will equal 100. If it hits the lower Bollinger Band the feature's value will be 0. When the closing price moves between the two bands the feature will move in $[0, 100]$ accordingly and move outside this range if the closing price moves beyond the bands.

$$\%B_s(t) = 100 \cdot \frac{c_s(t) - BB_{L_s}(t)}{BB_{U_s}(t) - BB_{L_s}(t)} \tag{2.35}$$

where $BB_{U_s}(t)$ and $BB_{L_s}(t)$ denote the values of the upper and lower Bollinger Band on day t, respectively.

Relative Strength Index

J. Welles Wilder wanted to address two issues many momentum indicators using price differences were suffering from. They often showed erratic movements caused by sharp price changes in the past, even if current prices have a low volatility. Furthermore, comparison was difficult since many of them were lacking a constant range. With the RSI Wilder proposed a method providing the necessary smoothing as well as a constant range of 0 to 100. Values above 70 indicate an overbought market, values below 30 indicate the opposite [28]. The RSI is defined as follows:

$$RSI_{s,N}(t) = 100 - \frac{100}{1 + RS_{s,N}(t)} \tag{2.36}$$

where

$$RS_{s,N}(t) = \frac{avgGain_{s,N}(t)}{avgLoss_{s,N}(t)} \tag{2.37}$$

Here, $avgGain_{s,N}(t)$ and $avgLoss_{s,N}(t)$ denote the average gain and the average loss over the last period of N days before day t. A typical value for N is 14.

$$avgGain_{s,N}(t) = \frac{1}{N} \sum_{i=1}^{N} \max\{c_s(t-i+1) - c_s(t-i); 0\} \tag{2.38}$$

$$avgLoss_{s,N}(t) = -\frac{1}{N} \sum_{i=1}^{N} \min\{c_s(t-i+1) - c(t-i); 0\} \tag{2.39}$$

Stochastic Oscillators

George Lane observed that when prices increase, closing prices tend to move in the upper part of the price range of the last N trading days. In turn, they tend to move in the lower part as prices decrease. He proposed two indicators, the %K line and the %D line. The %K line, which is the more sensitive one, is calculated according to equation 2.40.

$$\%K_{s,N}(t) = 100 \cdot \frac{c_s(t) - l_{s,min_N}(t)}{h_{s,max_N}(t) - l_{s,min_N}(t)} \tag{2.40}$$

where $l_{s,min_N}(t)$ and $h_{s,max_N}(t)$ denote the lowest low and the highest high price of stock s over the last N days before t:

$$l_{s,min_N}(t) = \min_{i \in \{t-N,\ldots,t\}} \{l_s(i)\} \tag{2.41}$$

$$h_{s,max_N}(t) = \max_{i \in \{t-N,\ldots,t\}} \{h_s(i)\} \tag{2.42}$$

A common value for N is 14. %D is a 3-period moving average of %K, thus called slow statistics, while %K is called fast statistics. These indicators oscillate within a range of $[0, 100]$. A buy signal is generated when the %D line moves above 80 while prices increase. On the opposite, a sell signal is generated when the %D line moves below 20 while prices decrease.

3 Experiment

3.1 Simulation Setup

To compare the stock predicting capabilities of the chosen algorithms, a stock trading simulation has been set up. In this simulation the classifiers' task is to predict whether the price of a stock will rise or fall within the next two weeks containing ten trading days (there is no trading on weekends). Accordingly, for the class labels the price of the stock s on day t is compared to the price on day $t + 10$:

$$y_s(t) \mapsto \begin{cases} 1, & \text{if } c_s(t+10) - c_s(t) > 0 \\ 2, & \text{otherwise} \end{cases} \tag{3.1}$$

The code for the simulation can be found on `https://github.com/tmechsner/automated-stock-trading`.

3.1.1 Stock Data

The simulation was run on a set of 100 stocks after tuning the algorithms' hyper-parameters on another 20 stocks. Most of the stocks used in this work are (historical) constituents of the S&P 500, a stock index covering the 500 biggest listed US companies, weighted by public float [29], [30]. The chosen stocks have been publicly available for the complete 15-years simulation period from 2002-06-01 to 2017-05-31. It is important to note that a choice of stocks that have been available on the market for a certain period is subject to survivor-bias. Excluding companies from the simulation that went bankrupt or that took their shares private results in better performance, as so-called survivors tend to have higher returns and lower volatility [31]. To counteract this tendency, some of the stocks that performed worst during the eight-year boom period since 2009 have been included in addition to the S&P 500 constituents [32]. The simulation period includes booming as well as declining market periods (so-called bull markets and bear markets) [33], [34]. All stock price data was obtained from the public Quandl API [35]. Complete lists of the stocks that have been used for simulation and hyper-parameter tuning are given in table 3.1 and 3.2, respectively. The following features of the data obtained from Quandl are relevant for this work: opening price, highest price, lowest price, closing price and trading volume, each adjusted and non-adjusted. In this work only the adjusted features have been used. This means that these features have been modified to be adjusted to splits, dividends and new emissions. Thus, these effects that cause abrupt price slumps and rises in the non-adjusted price and volume data can be ignored.

1. A	21. BAX	41. EXC	61. JNJ	81. MCD
2. AAP	22. BBT	42. EXPD	62. JNPR	82. MCHP
3. AAPL	23. BBY	43. F	63. LB	83. MCK
4. ABC	24. BDX	44. FAST	64. LEG	84. MCO
5. AEE	25. BEN	45. FCX	65. LEN	85. MDLZ
6. AEP	26. BF.B	46. FDX	66. LH	86. MDR
7. AES	27. BIIB	47. FE	67. LLL	87. MDT
8. AET	28. BK	48. FFIV	68. LLY	88. MET
9. AFL	29. BLK	49. FIS	69. LNC	89. MGM
10. AGN	30. BLL	50. INTU	70. LNT	90. MHK
11. AIG	31. BMY	51. IP	71. LOW	91. RIG
12. AIV	32. BRK.B	52. IPG	72. LRCX	92. T
13. AJG	33. BSX	53. IR	73. LUK	93. XL
14. AKAM	34. BWA	54. IRM	74. LUV	94. XLNX
15. ALB	35. BXP	55. IT	75. M	95. XOM
16. AVY	36. DISH	56. ITW	76. MAA	96. XRAY
17. AXP	37. DLTR	57. IVZ	77. MAC	97. XRX
18. AZO	38. DOV	58. JBHT	78. MAR	98. YUM
19. BA	39. DRE	59. JCI	79. MAS	99. ZBH
20. BAC	40. EW	60. JEC	80. MAT	100. ZION

Table 3.1: Stock symbols (tickers) of the stocks that have been used for simulation.

1. ABMD	5. C	9. FL	13. PNC	17. WBA
2. ABT	6. COF	10. GPC	14. TTWO	18. WDC
3. ACN	7. COL	11. PKI	15. TXN	19. WEC
4. ADBE	8. COO	12. PLD	16. TXT	20. WFC

Table 3.2: Stock symbols (tickers) of the stocks that have been used for hyper-parameter tuning.

Eight stocks were not able to compensate the average annual inflation of 1.96% [36]. Only seven stocks lost in value, so their price in 2017 was lower than in 2002. Among these the American International Group Inc. (stock symbol: AIG) performed worst among all analyzed stocks with a loss of 93.4%. The median of the 15-year performance of all 100 stocks is 363%. However, the third quartile is 592% which is very close to the performance mean of 580%, meaning that 75% of all stocks under consideration performed worse than average. This is due to a small number of very well performing stocks that pull up the mean: Best performed Apple Inc. (stock symbol: AAPL) with a rise of 10,348.89%, followed by Akamai Technologies, Inc. (stock symbol: AKAM) with a performance of 2,393.4% and F5 Networks, Inc. (stock symbol: FFIV) with a performance of 2,209.13%. All in all, 84 of the 100 stocks used in the simulation at least doubled their value during the 15-year period. That is equivalent to an average annual performance of at least 5%.

These numbers demonstrate why it is important to compare the performance of all classifiers to some baseline classifier and a baseline trader: the distribution of class labels is imbalanced towards a positive price development. Accordingly, even a simple majority classifier that for example always votes for the majority class of the last three years will produce a very well seeming classification accuracy, and even a trader that invested all available capital in 2002 and sold all stocks in the end of the simulation period in 2017 will find his or her capital almost sextupled.

3.1.2 Features

The features used as input for the classification algorithms are derived from the TIs described in section 2.5. Data points with features containing invalid values due to division by zero have been erased. Invalid values occur since some TIs like SMAs require a certain number of historical data available to operate on, which is not provided for the first days in the first training data set. The features are named $f_{s,i}(t)$ for feature i of stock s on day t.

The first four features are normalized versions of RSI, %K, %D and %B. They oscillate in the interval $[0, 100]$ and have been scaled to $[-1, 1]$ to roughly match the range of the other features and not dominate the distance metric.

To enable the algorithms to assess price- and volume-SMAs according to the double crossover method, a measure is needed to determine whether one line moves above or below the other. However, a simple difference could dominate the distance metric in k-NN algorithms when it becomes too big. Thus, the difference is scaled to $[-1, 1]$ using the tangens hyperbolicus as described in equations 3.2 to 3.5. For the short SMA a five-day period and for the long SMA a 20-day period was used.

$$f_{s,4}(t) = tanh(m_{s,5}(t) - m_{s,20}(t)) \tag{3.2}$$
$$f_{s,5}(t) = tanh(c_s(t) - m_{s,5}(t)) \tag{3.3}$$
$$f_{s,6}(t) = tanh(m_{V_s,5}(t) - m_{V_s,20}(t)) \tag{3.4}$$
$$f_{s,7}(t) = tanh(V_s(t) - m_{V_s,20}(t)) \tag{3.5}$$

where $V_s(t)$ denotes the trading volume of stock s on day t, while $m_{V_s,N}(t)$ is a SMA over the trading volume $V_s(\cdot)$ on a N-day period before day t like it is defined for $c_s(\cdot)$ in equation 2.33.

3.1.3 Procedure

Due to the reasons given at the end of section 3.1.1, a three-year majority classifier and a buy&hold (B&H) strategy trader will be simulated serving as comparison baseline for the algorithms described in chapter 2. The buy&hold trader will invest all available capital on the first trading day and sell all available stocks on the last trading day, without any trading activity in between.

Since fund managers mostly use TA for shorter-term horizons like weeks [23], the trading interval has been set to ten trading days or two weeks, respectively. However, all algorithms are trained on a daily basis to increase the amount of training data. The classification target remains the same. Trading starts ten days after a training period of three years to avoid look-ahead bias. For the majority baseline classifier and the batch-learners (all except SAM k-NN and ARF) the training period will be shifted forward after one year of trading to achieve actuality through a sliding window approach. The stream learners continuously update their model with every new data point but with a delay of ten trading days since the correct label of a data point is not known earlier.

For each classification algorithm and for the baselines a virtual trader is created that trades one single stock s over the whole simulation period. On every tenth simulation day each trader's classifier is asked to predict whether the price of the stock s will rise or fall within the next ten trading days (two weeks). If the predicted label is 'rise', a buy-signal is generated telling the trader to invest all available cash for buying as many stocks of s as possible at the current day's closing price. In turn, if the predicted label is 'fall', a sell-signal is generated causing the trader to sell all available stocks at the current price. All confidence-aware traders just do nothing ('hold') when their classifiers' confidence score is below their confidence threshold value.

To approach a realistic automated trading scenario the choice of transaction fees is an important factor for evaluation. If the trends are detected correctly, but the exploited price jumps amount to a value lower than the transaction costs, all profit will be used up by the fees resulting in a negative return despite correct price predictions. The trading will be simulated to be executed by Interactive Brokers at Xetra, an important German electronic trading venue opened by Frankfurter Wertpapierbörse [37]. 90 % of all stock trades in Germany are executed at this venue. Interactive Brokers is an online broker that offers a trading API to the public. At this broker, trading at Xetra costs 0.1 % of trading volume, minimum 4.00 €, maximum 99.00 € per order [38].

As stated in section 2.3.4, SAM k-NN comes with problem-independent hyper-parameters. All other classifiers' hyper-parameters have been tuned on a set of 20 stocks. This tuning set and the simulation set are disjunct. The hyper-parameters that have been found to be optimal for this setting are given in table 3.3. The confidence thresholds for the two confidence-aware algorithms have not been tuned since a higher confidence threshold implies more correct classifications but also more hold-signals. To be able to

Random Forest

Number of trees:	10
Max. features:	3

k-NN

Neighborhood size k:	21
Weighted:	no

Confidence Random Forest

Number of trees:	100
Max. features:	3
Confidence threshold:	.6, .65, .7

Confidence k-NN

Neighborhood size k:	21
Weighted:	no
Confidence threshold:	.6, .65, .7

Adaptive Random Forest

Number of trees:	100
Max. features:	3
Grace period:	20

SAM k-NN

Neighborhood size k:	5
Weighted	yes
Min. STM size L_{min}:	50
Max. storage size L_{max}:	5000

Table 3.3: Choice of hyper-parameters for all algorithms used in this work.

compare the different behaviours of traders with different thresholds three instances of each classifier have been tested with thresholds of 0.6, 0.65 and 0.7, respectively.

3.1.4 Evaluation Methodology

As mentioned before, the use of classification accuracy as single performance measure for stock prediction has been criticized by Teixeira and De Oliveira [9]. In a real-world application the algorithm would be rated on the profit it was able to achieve in contrast to other (human) traders. Hence, this work considers not only classification accuracy, as in the present application scenario a low accuracy classifier with correct buy predictions on high price jumps can be more profitable than a high accuracy classifier that fails to predict the most profitable opportunities. All classifiers are compared to a buy&hold strategy regarding profit and to a three-year majority classifier regarding F1-score.

Additionally, the concept drift-handling and confidence-aware algorithms are compared to the original k-NN or RF trader, respectively. In the comparison, outperformance in terms of profit and F1-score will be analyzed by using Wilcoxon's signed rank test. These concepts will be explained in more detail in the following section. In the evaluation context, buy-signals are regarded as positives, so a correctly generated buy-signal is a true positive while a buy-signal on falling prices is regarded as a false positive.

Evaluation of Classification Performance

Selling when the price will rise is a missed profit opportunity. In turn, buying when the price will fall causes a real loss of capital. Accordingly, besides correctly predicting many rising prices (maximize true positives), the goal is to avoid buying when the price will fall (minimize false positives). Thus, instead of maximizing prediction accuracy, the goal is to maximize the fraction of buy-signals that are followed by rising prices (implicitly reducing costly false positives) and the fraction of all price-rises that have been correctly predicted (avoid becoming too risk-averse). These fractions are called precision and recall and can be maximized by optimizing the F-measure or F1-score, proposed by Van Rijsbergen, as it calculates the harmonic mean of these two values [39]:

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.6}$$

where

$$precision = \frac{tp}{tp + fp} \tag{3.7}$$

$$recall = \frac{tp}{tp + fn} \tag{3.8}$$

with tp denoting the number of true positives, fp denoting the number of false positives and fn denoting the number of false negatives.

Statistical Testing for Classifier Comparison

According to Bifet, the most popular statistical test for comparing classifiers is McNemar's test [40]. However, he shows that its results can be misleading in a stream learning setting. The reason for this is assumed to be the fact that it considers the classification results of single data points instead of whole folds and thus is more susceptible to performance fluctuations. Instead, it is recommended to use Wilcoxon's signed-rank test, which generated the most reliable results among the examined tests (McNemar, Wilcoxon's signed-rank test and sign test).

Wilcoxon's signed-rank test [41] is a non-parametric test that can be used to compare classifiers. The two-tailed test tests the null hypothesis that the two classifiers perform equally well. To find out which of the classifiers performs better, the one-tailed version can be used as post-hoc test. It tests the null hypothesis that the first classifier performs worse or equally well as the second. Thus, rejection here means that the first classifier performs better than the second (as in this case equality has been disqualified by the two-tailed test already). For each fold, in this case for each stock, it calculates the performance difference of two classifiers and ranks the absolute values of these in ascending order. Afterwards, the ranks of positive differences and the ranks of negative differences are summed up. Zero-differences are excluded in this calculation. For the two-tailed test, the statistic W is the smaller of the two sums. For the one-tailed test, W is the sum of the negative differences. For a sample size $n > 20$ the test statistic W approximates a normal distribution. Hence, a z-score can be calculated:

$$z = \frac{W - \mu_W}{\sigma_W} \approx \mathcal{N}(0, 1) \tag{3.9}$$

where $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution, μ_W denotes the mean and σ_W denotes the standard deviation of the distribution of W [42]:

$$\mu_W = \frac{1}{4}n(n+1) \tag{3.10}$$

$$\sigma_W = \sqrt{\frac{n(n+1)(2n+1)}{24}} \tag{3.11}$$

If rank ties occur, meaning two or more pairs having the same absolute difference, they will be assigned with the average of the ranks they span [43]. In this case, the test statistic has to be adjusted:

$$\frac{W - \frac{1}{4}n(n+1)}{\sqrt{\frac{n(n+1)(2n+1)}{24} - \sum_{i=1}^{n} \frac{t_i^3 - t_i}{48}}} \approx \mathcal{N}(0,1) \tag{3.12}$$

where t_i denotes the number of sample pairs with the same rank as the pair (x_{i1}, x_{i2}). Furthermore, for $n \leq 60$ a continuity correction has to be applied. This is irrelevant for this work, as sample size is $n = 100$.

The null hypothesis can be rejected if the actual computed value of the statistic $\hat{W}$ is unlikely according to its distribution. This unlikeliness is defined by a threshold value α that is usually chosen between 5% and 1%. The data statistically disproves the null hypothesis if $\mathcal{N}(\hat{W}|\mu = 0, \sigma^2 = 1) < \alpha$.

3.2 Evaluation

3.2.1 Experimental Results

For each trader and each stock the final cash balance after selling all stocks, the number of buy-, sell- and hold-signals and the F1-score over all predictions concerning that stock have been recorded during the simulation. These results are displayed in aggregated form in table 3.4.

To be able to compare the traders the Wilcoxon's signed rank statistic has been calculated on F1-scores and final balances for the most interesting combinations of stocks ($\alpha = 5\%$). The resulting p-values are given in tables 3.5.

3.2.2 Analysis

Almost all of the classifiers performed significantly worse than the majority classifier regarding F1-scores and also significantly worse than the buy&hold trader regarding final balances. Only the confidence-aware k-NN with a confidence threshold of 70% performed approximately equally well in terms of classification and profit. However, this classifier on average produced 248 hold-signals and only 55 trade-signals. There should be a lower chance of miss-classification among these 55 predictions. This is reflected by the fact that it has the highest average F1-score after the majority classifier. However, although it additionally achieved the highest maximum F1-score of all classifiers, it also produced

	F1-score			**Better**	**#Signals**		
	min.	**max.**	**mean**	**than B&H**	**Buy**	**Sell**	**Hold**
Majority	.42	.76	**.67**	.19	253	48	–
k-NN	.44	.69	.60	.26	201	100	–
.60 Confidence k-NN	.36	.76	.63	**.36**	116	36	149
.65 Confidence k-NN	.25	.80	.64	.33	76	18	207
.70 Confidence k-NN	.21	**.83**	.66	.29	47	8	248
SAM k-NN	.42	.69	.59	.26	187	114	–
RF	**.48**	.70	.60	.24	202	99	–
.60 Confidence RF	.43	.75	.60	.29	104	50	147
.65 Confidence RF	.34	.74	.60	.30	68	29	203
.70 Confidence RF	.24	.78	.62	.34	41	15	244
Adaptive RF	.46	.67	.57	.25	179	122	–

Table 3.4: This table shows the lowest, highest and average F1-score of each trader among all stocks. Additionally the fraction of simulations in which the trader had a better final balance than the buy&hold trader and the average number of buy-, sell- and hold-signals are given.

the lowest minimum F1-score. The low number of trades finally leads to a low profit compared to the baseline. It outperformed the buy&hold approach in only 29% of simulations and produced overall significantly worse returns according to the test results. With a fraction of 36% the confidence-aware k-NN with a threshold of 60% most frequently outperformed the buy&hold trader. This low maximum value of that column is supported by the results of the comparison of final balances. It yields a result similar to the test on F1-scores. Without exception, all classifiers performed significantly worse than the buy&hold baseline approach.

The advanced classifiers have also been compared to their basic implementation counterpart. For k-NN the confidence-aware classifiers had higher average F1-scores, but also lower minimum values among their F1-scores. According to the results of Wilcoxon's test this F1-score outperformance is significant. They also more often outperformed the buy&hold baseline. However, they all rejected the prediction on at least half of all trading days. On the remaining allegedly confident predictions the classification performance is rather low, though. The comparison of the confidence-aware Random Forests to the basic Random Forest looks similar, with the exception that there is no difference in the average F1-score. Only the classifier with a 70% confidence threshold had a slightly higher average F1-score than the standard Random Forest. This difference is small but significant according to the test results. Regarding profit all confidence-aware classifiers outperformed k-NN and Random Forest, respectively. Improved performance in situations in which the classifier considers itself more confident indicates that there is some knowledge to extract from the selected features. However, the high number of rejections and the small increase in profit and classification accuracy reveal that these specific situations are rare.

	Majority	k-NN	RF
k-NN	0.00 1.00	–	–
.60 C. k-NN	0.00 1.00	0.00 0.00	–
.65 C. k-NN	0.0002 0.9999	0.00 0.00	–
.70 C. k-NN	0.2301	0.00 0.00	–
SAM k-NN	0.00 1.00	0.0173 0.9913	–
RF	0.00 1.00	–	–
.60 C. RF	0.00 1.00	–	0.7155
.65 C. RF	0.00 1.00	–	0.4682
.70 C. RF	0.00 1.00	–	0.0024 0.0012
ARF	0.00 1.00	–	0.00 1.00

F1-score comparison

	Buy & hold	k-NN	RF
Majority	0.00 1.00	–	–
k-NN	0.00 1.00	–	–
.60 C. k-NN	0.0001 0.9999	0.0003 0.0001	–
.65 C. k-NN	0.0001 1.00	0.0003 0.0001	–
.70 C. k-NN	0.00 1.00	0.00 0.00	–
SAM k-NN	0.00 1.00	0.5543	–
RF	0.00 1.00	–	–
.60 C. RF	0.0002 0.9999	–	0.0727
.65 C. RF	0.0001 1.00	–	0.0175 0.0088
.70 C. RF	0.0044 0.9978	–	0.0121 0.0060
ARF	0.00 1.00	–	0.3204

Final balance comparison

Table 3.5: The tables show Wilcoxon's signed rank test p-values. The left table contains the results of F1-score comparison, the right table shows the results of the comparison of final balances. Single numbers are non-significant results of two-tailed Wilcoxon's signed rank tests. When two numbers are given, the upper one is the significant result of the two-tailed test and the lower one is the result of the one-tailed test. The values are rounded to four decimal places. $\alpha = 5\%$. Not all algorithms have been compared.

Both algorithms aware of concept drift (Adaptive Random Forest and SAM k-NN) did not perform better than their non-aware counterparts. Regarding final balances their performance did not differ significantly and their F1-scores are even significantly worse than those of k-NN and Random Forest. The drift-handling aspect might become more useful when the overall predictability of the time series is higher. There seems to be very little knowledge extractable from the chosen features in the present case. Accordingly, most prediction error fluctuations are more likely to be caused by random noise than by a concept drift. This would then lead to a random adaption of the memories in SAM k-NN and to random re-training and replacement of trees in Random Forest without any advantage for prediction quality.

4 Conclusion

The goal of this thesis was to find out whether the performance of classification algorithms in the context of financial markets can be improved by enhancing them with the ability to handle concept-drift and to reject predictions in uncertain situations. The feature space was chosen to be constructed of Technical Indicators that are originated from the theory of Technical Analysis: simple to compute features based purely on historical stock prices and trading volumes. Subject to the analysis were the k-Nearest Neighbors and Random Forest estimators and two variations of each: one implementation providing a measure of confidence for their output and one capable of handling concept drift (Self-Adjusting Memory k-Nearest Neighbors and Adaptive Random Forest, respectively). Furthermore, a majority classifier and a buy&hold trading strategy have been implemented serving as baseline to compare the algorithms to. In contrast to many other publications in this realm not only the pure classification performance was analyzed but also the virtually achievable profit in a simulated stock trading environment. Therefor, a stock trading simulation has been set up that makes use of realistic transaction costs. With this system 100 different stocks have been traded during a period of 15 years from 2002 to 2017. The resulting measures for prediction performance (F1-score) and profit (final cash balance after the simulation) have been compared utilizing Wilcoxon's signed-rank test yielding statistically significant statements about performance differences.

The prediction of stock prices through a selection of Technical Indicators proved to be very difficult. None of the analyzed algorithms produces better results than the baselines, neither regarding F1-scores nor regarding profit. Yet, confidence-awareness significantly improves performance in respect of profit. In turn, the ability of handling concept drift is not advantageous. It could become beneficial when the variation in performance of selected algorithms on selected features can be reduced.

4.1 Prospectus

Next steps could include improvements or variations on the algorithms as well as regarding the simulation setup. As the predictability of the analyzed time series is weak given the presented features, better features should be investigated that also take economic factors and investor sentiment into account. For example, one could analyze companies' fundamental economical values, interest rates and social media and news articles. To further improve the k-NN based algorithms the euclidean metric could be replaced with a metric that is more specific to the chosen feature space. Such a metric can be found utilizing metric learning approaches. If these measures lead to an improved overall predictability and if this turns the concept drift-handling ability into an advantage, these algorithms could be combined with the confidence score approach. For being able to optimize the threshold of the confidence-aware traders opportunity costs for keep-signals could be introduced. This would avoid the tendency of just rejecting the majority of predictions by making high risk-averseness less valuable.

The presented simulation procedure is not perfectly precise in determining the price at which stocks are bought and sold. Here, the simulation could get more realistic by taking into account the current price volatility to determine the actual buying and selling price. Furthermore, the trade timing should be optimized. To decide whether an order should be placed in the morning or better in the afternoon, intra-day features on the trade-day could be analyzed. For confidence-aware algorithms the evaluation approach in respect to profit could be replaced by evaluating the performance of a simulated investment fund which buys stocks that most certainly will experience a rise in price and sells those that will most certainly lose value.

5 Bibliography

[1] T. Cover and P. Hart, “Nearest Neighbor Pattern Classification,” *IEEE Trans. Inf. Theor.*, vol. 13, no. 1, pp. 21–27, Sep. 2006. DOI: 10.1109/TIT.1967.1053964.

[2] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324.

[3] A. Lo, H. Mamaysky, and J. Wang, “Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation,” *Journal of Finance*, vol. 55, no. 4, pp. 1705–1765, 2000. DOI: 10.1111/0022-1082.00265.

[4] H. White, “Economic prediction using neural networks: the case of IBM daily stock returns,” *IEEE International Conference on Neural Networks*, vol. 2, pp. 451–458, 1988. DOI: 10.1109/ICNN.1988.23959.

[5] M. K. Pareek and P. Thakkar, “Surveying stock market portfolio optimization techniques,” in *5th Nirma University International Conference on Engineering (NUiCONE)*, Nov. 2015, pp. 1–5. DOI: 10.1109/NUICONE.2015.7449613.

[6] Y. Li, J. Wu, and H. Bu, “When quantitative trading meets machine learning: A pilot survey,” in *13th International Conference on Service Systems and Service Management (ICSSSM)*, Jun. 2016, pp. 1–6. DOI: 10.1109/ICSSSM.2016.7538632.

[7] S. Lahmiri, M. Boukadoum, and S. Chartier, “Information Fusion and S&P500 Trend Prediction,” *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, 2013. DOI: 10.1109/AICCSA.2013.6616488.

[8] Y. Chen and Y. Hao, “A feature weighted support vector machine and K-nearest neighbor algorithm for stock market indices prediction,” *Expert Systems with Applications*, vol. 80, pp. 340–355, 2017. DOI: 10.1016/j.eswa.2017.02.044.

[9] L. A. Teixeira and A. L. I. De Oliveira, “Predicting stock trends through technical analysis and nearest neighbor classification,” *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pp. 3094–3099, 2009. DOI: 10.1109/ICSMC.2009.5345944.

[10] J. Gama, I. liobait, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A Survey on Concept Drift Adaptation,” *ACM Comput. Surv.*, vol. 46, no. 4, 44:1–44:37, Mar. 2014. DOI: 10.1145/2523813.

[11] S. A. Dudani, "The Distance-Weighted k-Nearest-Neighbor Rule," in *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, Apr. 1976, pp. 325–327. DOI: 10.1109/TSMC.1976.5408784.

[12] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Heidelberg: Springer-Verlag, 2006.

[13] V. Losing, B. Hammer, and H. Wersing, "KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift," in *IEEE 16th International Conference on Data Mining (ICDM)*, Barcelona, 2016, pp. 291–300. DOI: 10.1109/ICDM.2016.0040.

[14] D. Arthur and S. Vassilvitskii, "K-means++: The Advantages of Careful Seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. DOI: 10.1145/1283383.1283494.

[15] L. Breiman, *Classification and regression trees*. Belmont, Calif: Wadsworth International Group, 1984, p. 358. DOI: 10.1002/widm.8.

[16] W. Loh, "Fifty Years of Classification and Regression Trees," *International Statistical Review*, vol. 82, no. 3, pp. 329–348, Nov. 2014. DOI: 10.1111/insr.12016.

[17] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug. 1996. DOI: 10.1023/A:1018054314350.

[18] H. M. Gomes, A. Bifet, *et al.*, "Adaptive random forests for evolving data stream classification," *Machine Learning*, vol. 106, no. 9, pp. 1469–1495, Oct. 2017. DOI: 10.1007/s10994-017-5642-8.

[19] A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing," in *Proceedings of the 2007 SIAM International Conference on Data Mining*, 2007, pp. 443–448. DOI: 10.1137/1.9781611972771.42.

[20] J. J. Murphy, *Technical Analysis of the Financial Markets*. New York Institute of Finance, 1999. DOI: 10.2139/ssrn.566882.

[21] E. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work," *Journal of Finance*, vol. 25, pp. 383–417, 1970. DOI: 10.1111/j.1540-6261.1970.tb00518.x.

[22] E. F. Fama, "Efficient Capital Markets II," *Journal of Finance*, vol. 46, no. 5, pp. 1575–1617, 1991. DOI: 10.2307/2328565.

[23] L. Menkhoff, "The Use of Technical Analysis by Fund Managers: International evidence," *Journal of Banking and Finance*, vol. 34, 11 Nov. 2010. DOI: 10.1016/j.jbankfin.2010.04.014.

[24] E. Davis and B. Steil, *Institutional Investors*. Cambridge, Mass. [u.a.]: MIT Press, 2001.

[25] G. Caginalp and H. Laurent, "The Predictive Power of Price Patterns," vol. 5, 1998, pp. 181–206. DOI: 10.1080/135048698334637.

[26] R. Haugen, *The New Finance: The Case Against Efficient Markets*. Prentice Hall, 1998. DOI: 10.2307/2329361.

[27] C. Los, "Nonparametric Efficiency Testing of Asian Stock Markets Using Weekly Data," *SSRN Electronic Journal*, vol. 14, Oct. 2004. DOI: 10.2139/ssrn.143950.

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

[29] B. Seidl. (2016). Der echte Gradmesser der US-Börsen, [Online]. Available: https://boerse.ard.de/boersenwissen/boersenwissen-fuer-fortgeschrittene/s-und-p-500-100.html (visited on Sep. 3, 2018).

[30] (2018). iShares S&P 500 UCITS ETF, [Online]. Available: https://www.ishares.com/de/privatanleger/de/produkte/251900/ (visited on Jul. 23, 2018).

[31] T. Schneeweis, R. Spurgin, and D. McCarthy, "Survivor bias in commodity trading advisor performance," *Journal of Futures Markets*, vol. 16, no. 7, pp. 757–772, 1996.

[32] P. Van Doorn. (2017). 10 best, 10 worst stocks in the eight-year bull market, [Online]. Available: https://www.marketwatch.com/story/10-best-10-worst-stocks-in-the-eight-year-bull-market-2017-03-09 (visited on Jul. 13, 2018).

[33] M. Egan. (2016). America's 7-year bull market: Can it last? [Online]. Available: https://money.cnn.com/2016/03/09/investing/stocks-bull-market-turns-seven/index.html (visited on Sep. 3, 2018).

[34] (2008). INSTANT VIEW: Dow industrials enter bear market territory, [Online]. Available: https://www.reuters.com/article/us-usa-economy-bearmarket-instant/instant-view-dow-industrials-enter-bear-market-territory-idUSN2732335520080627 (visited on Sep. 3, 2018).

[35] (2018). Quandl Financial Data API, [Online]. Available: https://www.quandl.com/tools/api (visited on Aug. 6, 2018).

[36] (). CPI - All items less food and energy in U.S. city average, [Online]. Available: https://data.bls.gov/timeseries/CUUR0000SA0L1E?output_view=pct_12mths (visited on Aug. 8, 2018).

[37] C. Heldt and S. Schöning. (2018). Xetra - Ausführliche Definition, [Online]. Available: https://wirtschaftslexikon.gabler.de/definition/xetra-48199/version-271457 (visited on Aug. 1, 2018).

[38] (2018). Stocks, ETFs (ETPs) and Warrants - Fixed Pricing Structure. https://www.interactivebrokers.com/en/index.php?f=1590&p=stocks1, retrieved 2018-08-11.

[39] C. J. V. Rijsbergen, *Information Retrieval*, 2nd. Newton, MA, USA: Butterworth-Heinemann, 1979. DOI: `10.1002/asi.4630300621`.

[40] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient Online Evaluation of Big Data Stream Classifiers," *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, no. April 2016, pp. 59–68, 2015. DOI: `10.1145/2783258.2783372`.

[41] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945. DOI: `10.2307/3001968`.

[42] R. Lowry. (2013). Concepts & Applications of Inferential Statistics, [Online]. Available: `http://vassarstats.net/textbook/ch12a.html` (visited on Sep. 3, 2018).

[43] J. W. Pratt, "Remarks on Zeros and Ties in the Wilcoxon Signed Rank Procedures," *Journal of the American Statistical Association*, vol. 54, no. 287, pp. 655–667, 1959. DOI: `10.2307/2282543`.

6 List of Figures

7 List of Tables