Master Thesis Econometrics and Management Science

Name Student: Sem Keegstra - Student ID: 475459

Quantitative Finance

Equity Trading by means of Interpretable Machine Learning

Abstract

This research concentrates on the implementation of various interpretable machine learning models within an equity trading context. The dataset consists of historical S&P 500 constituents, which are analysed on a quarterly basis over a time period stretching from 2000 to 2020. In order to assess the feasibility of rule-based classifiers, the CART, Classy, Ripper and RUG algorithm are evaluated in terms of interpretability, classification and portfolio performance. The models are also compared with the Random Forest algorithm, which serves as a state-of-the-art benchmark model. In particular, it is found that for small- and large-sized rolling windows respectively the Ripper and RUG algorithm are able to match the performance of the Random Forest model in terms of long position forecasts. Furthermore, implementing a certainty threshold for short position forecasts results in all model-based allocations providing a higher compounded return than the market portfolio while not being significantly more volatile.

Keywords: Interpretability; stock selection; classification; rule-learning; certainty threshold

ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Supervisor: Dr. Hakan Akyüz

Second assessor: MSc. Utku Karaca

Date final version: 28th of July 2021

The views stated in this thesis are those of the author and not necessarily those of the supervisor, second assessor, Erasmus School of Economics or Erasmus University Rotterdam.

Contents

1 Introduction **2**

2 Literature Review **4**

3 Data **7**

3.1 Data Cleaning 7

3.2 Feature Engineering 9

3.3 Feature Selection 10

4 Methodology **11**

4.1 Decision Trees 11

4.2 Random Forest 13

4.3 Classy Algorithm 14

4.4 Ripper Algorithm 16

4.5 Rule Generation Algorithm 17

4.6 Evaluation Metrics 19

4.7 Certainty Threshold 22

5 Results **22**

5.1 Computational Platform 22

5.2 Rule Structure 23

5.3 Classification Performance 25

5.4 Portfolio Performance 29

6 Conclusion **33**

References **35**

A Data Glossary **38**

B Python Scripts **41**

1 Introduction

In the field of quantitative finance, scientific research has primarily distinguished three stages within trading strategies, namely: stock selection (van der Hart et al., 2003), portfolio evaluation (Sharpe, 1970) and market timing (Shen, 2003). As the importance of equities increased over the past decades, differentiating between 'good' and 'bad' stocks has become a demanding part of the portfolio optimization process. In combination with the great economic uncertainty of today's financial market and low-interest rates, investors seek a reliable technique to aid them in the selection of stocks. Incorporating a classification framework that is able to consistently create a profitable portfolio structure, while identifying attractive stocks with minimum effort could be beneficial for investors and give them a competitive advantage.

Both in the industry and academia, different trading strategies have been examined and proposed based upon fundamental and technical evaluation. Markowitz (1952) introduced the so-called Modern Portfolio Theory (MPT), which has served as the groundwork for numerous innovations in the field of portfolio selection. However, such mean-variance approaches frequently concentrate on maximizing the expected risk-adjusted return and obtaining a reliable estimate for this is found to be extremely difficult in practice. In addition, the investable universe is very broad and even in the most simplified scenarios the dataset often contains more stocks than price history. Therefore, calculating risk measures such as the covariance matrix is simply not feasible. Despite there being possible solutions for this dimensionality problem, DeMiguel et al. (2009) demonstrate that the mean-variance optimal portfolio cannot consistently outperform naïve distribution approaches in the modern stock market.

After the Global Financial Crisis (GFC) of 2008, the adoption of machine learning models grew tremendously in quantitative finance as the traditional factor models lost their profitable edge. In particular, Fu et al. (2018) and Gu et al. (2018) find that these new algorithms perform extremely well in equity trading scenarios using a classification and regression framework respectively. Similarly across various fields outside of finance, classical benchmarks are significantly outperformed by the implementation of machine learning models. Two main reasons for this trend, as established by Rasekhschaffe and Jones (2019), are that they allow for better detection of non-linear interactions within the dataset and are more effective in the presence of multicollinearity as opposed to more linear orientated econometric techniques. Nevertheless, this high performance level comes with a major obstacle given that the methods often do not allow for any interpretation from a decision making point of view. Several techniques have been introduced to try and overcome this, however they are mostly based on either feature importance or reducing the complexity of well-established 'black-box' models. In terms of feature importance, the main disadvantage is that it does not offer the interpretability in such a way that one is able to understand and trust the decisions made by a predictive model. In addition, reducing the complexity of machine learning algorithms often means purposely decreasing the accuracy of the predicter which in many applications cannot be properly justified. As a result, current literature is overflowed with research that compares the predictive performance of models which are not capable of giving investors beneficial insights on the behaviour of stocks.

In order to improve the human understanding of the choices made by automatic decision-making systems, recent studies have explored the potential of algorithms that are based on decision-rules. Prior work in this area, such as Decision Trees (DT) as proposed by Breiman et al. (1984) or the Ripper algorithm of Cohen (1995), displayed great results in terms of transparency. However, these simplistic models are often no match for the complex machine learning techniques that are employed in the industry today. As an alternative, Proença and van Leeuwen (2020) introduce the Classy algorithm, which finds probabilistic rule lists that facilitate a balance between accuracy and interpretability through the Minimum Description Length (MDL) principle. Furthermore, a recent study by Akyüz and İlker Birbil (2021) takes advantage of linear programming and formulates a Rule Generation (RUG) algorithm using column generation. Both studies indicate that their approach is capable of providing a clear decision-making procedure whilst also matching the classification accuracy of state-of-the-art machine learning models. Given that these studies are very promising to the field of interpretable machine learning, further research has mostly focussed on extending such ideas by introducing alternative formulations or further optimizing the computational process. This had led to a gap in the literature, where innovations in rule-based learning algorithms have not been analysed in a variety of real-world domains that heavily rely on human decision making.

Based on the two problems described above, the research question of this paper arises:

To what extent is an interpretable classifier capable of building a financial portfolio that is both efficient and profitable?

To try and broaden the financial literature on interpretable machine learning, this research evaluates the performance of various interpretable learning algorithms within an equity trading context. In particular, the following sub-questions are taken into account:

1. Are the interpretable models able to match the classification performance of black-box models?
2. Do their resulting decision rules translate to efficient diversification strategies?
3. Can they provide this performance whilst being less volatile than the market portfolio?
4. Does a certainty threshold for short positions improve the model-based portfolio allocations?

The dataset consists of stock observations from historical S&P 500 constituents, which are analysed on a quarterly basis over a time period stretching from 2000 to 2020. In terms of interpretable classifiers, this paper evaluates the CART, Classy, Ripper and RUG algorithm. Furthermore, the Random Forest (RF) model is implemented as the benchmark algorithm due to its great empirical performance across recent studies on the subject of stock selection. Both the classification performance and portfolio results are assessed for a variety of rolling windows. In addition, this study explores the practical implications of a lower bound on the forecasting probability of short position stocks when rebalancing the portfolio.

Considering the transparency of the decision-making process, all classifiers construct their decision boundaries based on significantly less rules compared to the RF algorithm. The individual rules are also more concise on average, making them easier to interpret for investors. Furthermore, it is found that for small- and large-sized rolling windows the Ripper and RUG algorithm respectively match the classification performance of RF in terms of long position forecasts. However, the RF algorithm is more versatile as all interpretable models exhibit a sharp decrease in classification performance for short position forecasts. The RUG algorithm is the only classifier that can beat both the machine learning benchmark and the market based on average quarterly returns when only including long positions. In addition, implementing a certainty threshold for short position forecasts results in all model-based allocations providing a higher compounded return that the market portfolio. Unfortunately, none of the interpretable classifiers offers this return while concurrently being less volatile than the market.

This paper proceeds with a literature review, which explains the current state of the financial literature regarding equity trading through interpretable learning algorithms and the corresponding contributions made by this research. This is followed by a description of both the dataset and the pre-processing steps that are needed to make the observations applicable for machine learning models. Subsequently, the theoretical framework of the various classifiers is described together with the relevant metrics that are incorporated to evaluate the classification and portfolio performance. Finally, the empirical results of the experiment are examined and the research questions are discussed within the final conclusion of the research.

2 Literature Review

This research is primarily motivated by the inspiring studies of Proença and van Leeuwen (2020) and Akyüz and İlker Birbil (2021) in the field of interpretable machine learning. While there exist various rule learning approaches, Proença and van Leeuwen (2020) propose a combination of probabilistic rule lists and the MDL principle which facilitates a balance between accuracy and interpretability. As a result, their so-called Classy algorithm finds small probabilistic rule lists that, in terms of a performance-transparency trade-off, outperform state-of-the-art machine learning models. In addition, their method successfully avoids both overfitting and the need for hyperparameter tuning. Akyüz and İlker Birbil (2021) take a linear programming attitude in which they construct new rules by means of column generation. The computational time of this approach is exceptionally low, as their RUG algorithm takes advantage of an ordinary DT to retrieve the initial rule set and get approximate solutions of the pricing subproblem. Furthermore, their numerical experiments indicate that the RUG algorithm achieves an average classification accuracy similar to that of the RF model.

In the current financial literature, numerous machine learning models have been implemented within a stock selection environment. Although research is quite active, most articles in this field focus on constructing a portfolio via regression techniques. Both Chen et al. (2020) and Batres-Estrada (2015) found success by pricing individual stocks through deep learning. Wang and Luo (2012) provided a comprehensive outline of the AdaBoost algorithm to forecast equity returns and Chia-Cheng et al. (2020) assess the performance of different models using historical S&P 500 data. Comparing the empirical results of different studies, it is found that the RF model is the most consistent algorithm and efficiently predicts stock price movements based on risk-adjusted measures. This level of performance translates to a classification approach, as Fu et al. (2018) introduce a machine learning framework to distinguish ‘good’ stocks from ‘bad’ stocks and also observe that RF is capable of consistently outperforming traditional statistical learning methods. In addition, they describe the model as a more risk-neutral alternative to econometric techniques, which generally behave more radically, resulting in higher portfolio returns. This paper incorporates the RF algorithm as a machine learning benchmark in order to evaluate the stock selection results of the interpretable learning models. Moreover, ordinary DTs and the Ripper algorithm are covered to assess in what extent the newly proposed techniques can outperform more simplistic interpretable classifiers.

One disadvantage of Ensemble Learning is the drastic decrease in interpretability. Whilst RF is a combination of highly transparent Decision Trees (DTs), it increases the total number of rules significantly. As a result, the model returns a large set of overlapping decision boundaries that are no longer explicable. Recent studies have tried to counter this problem by reducing the complexity of the given boundaries. Birbil et al. (2020) introduced the Minimum Rule Cover (MIRCO) algorithm, which main objective is to extract a smaller set of rules that covers all observations and minimizes the total impurity. They find a subset that achieves this while closely matching the accuracy of the initial RF model. However, MIRCO itself is not a classification algorithm as it merely extracts the decision rules from a fully trained RF and therefore does not necessarily cover the entire feature space. Furthermore, the user study of Lakkaraju et al. (2016) indicates that the decision rules obtained from models such as DTs are far less interpretable due to their hierarchical structure.

Besides rule-based algorithms, other complex machine learning techniques often achieve high accuracy in modern datasets. These so-called 'black-box' models create tension between accuracy and interpretability during the classification process as experts struggle to comprehend the logic of the offered decisions. In response, current research has particularly focused on increasing the interpretability through feature importance. Whereas these solutions are commonly tailored to a specific classifier, Lundberg and Lee (2017) introduce Shapley Additive Explanations (SHAP) which serves as a unified approach for ranking the features of any machine learning algorithm. Centred around classical game theory, it ties the optimal credit allocations with local explanations using the Shapley values of Shapley (1953). While this approach allows for new insights in terms of the role that a feature plays when predicting a specific observation, it does not explain how the feature is incorporated. Therefore, the interpretability is not increased in such a way that one is able to understand and trust the decisions made by a predictive model.

In classical portfolio theory, equities are usually ranked by means of numerical measures which researchers believe to capture some sort of market inefficiency (Jegadeesh and Titman (2012) and Sloan (1996)). This paper follows a similar attitude by categorizing stocks with a classification function, based upon the stock selection framework that Fu et al. (2018) designed for machine learning models. Only the firm's fundamental data is considered when training the classifiers, as supported by the efficient market hypothesis of Malkiel and Fama (1970). In addition, this research applies an equally weighted portfolio strategy seeing that the empirical results of Plyakha et al. (2012) indicate that it consistently outperforms price- and value-weighted portfolios.

Whilst distinguishing between stocks through classification, current studies often neglect the possibility of going short. Therefore, I extend the portfolio evaluation of Fu et al. (2018) by allowing both long and short positions via a class probability threshold during the stock selection process (see Section 3 for a more detailed explanation). Huerta et al. (2013) find that only using the top and bottom percentage of available equities improves the out-of-sample performance. They argue that the stocks ranked in between are more likely to be idiosyncratic and tend to follow the market trend, which adds noise to the training procedure. During this research, a 5 percent quantile selection is implemented to handle the noisy observations and make certain that the individual classifiers are only trained on stocks that hold conventional characteristics. Lastly, one problem that occurs frequently in this field of equity trading is the so-called current constituent bias as covered by Wang and Luo (2014). Due to the broad investable universe, studies often simplify their dataset by only including stocks that are part of an index at a specific point in time. However, exploiting the current index constituents involves a look-ahead bias during back testing. In response, this paper takes a more pragmatic approach by tracking the historic constituents of the S&P 500 index for each observation period.

Note that this paper makes the following contributions to the current financial literature. Firstly, while the RF algorithm seems to excel within a stock selection framework, other rule-based decision models have not yet been examined inside the same setting. Furthermore, the main focus of recent studies has been to solely maximize the accuracy of classifiers and as a result the interpretability aspect has been overlooked almost entirely. This paper tries to fill this gap by analysing the classification performance of various interpretable rule-learning models. In addition, whilst the RUG algorithm is tested on a large collection of frequently used datasets, the possible value it might add to the stock selection process is still unknown. Similarly, the Classy algorithm has only been examined on relatively simple classification datasets and therefore it is still uncertain how it will perform compared to other models within a noisy real-world scenario such as equity trading. Finally, this study explores the practical implications of a lower bound on the forecasting probability of short positions when rebalancing the portfolio to obtain an equity trading framework that is both profitable and transparent.

3 Data

In order to evaluate the added value that interpretable classifiers bring to equity trading, I analyse quarterly stock prices of available historical S&P 500 constituents from 2000 until 2020. This time period is of interest as it captures the full financial cycle of the stock market, including: recessions, a bubble and a bull market. In particular, both the Global Financial Crisis (GFC) of 2008 and the Coronavirus Crash of 2020 are covered. The observations are obtained from the Compustat database, accessible at the Wharton Research Data Services, consisting of 630 fundamental indicators and 10 technical indicators. Furthermore, the historical constituents of the index are retrieved by means of web scraping (see Appendix B for more detail). The set of unique assets in which we are allowed to invest is denoted by the so-called investable universe. Hence, during this research the investable universe within each quarter is equivalent to the historical constituents of the S&P 500 index. Before this dataset is suitable for machine learning models, the following important steps are performed: data cleaning, feature engineering and feature selection.

3.1 Data Cleaning

The data cleaning process starts by evaluating the availability of the various indicators throughout the entire observation period. All features that either have an average of at least 20 missing values across all quarters or have more than 20% of the observations missing in a particular quarter are removed. From the remaining stock observations, each data point that still contains some missing information is deleted as well. This results in a total of 40,047 observations with 107 company specific features and an average of around 482 stocks per quarter (see Appendix A for a more extensive overview of the data). The level of the different firm characteristics varies considerably, therefore all numerical features are scaled by subtracting the mean and dividing by the corresponding standard deviation. In general, the objective functions of machine learning algorithms cannot work properly if the features did not undergo such normalization, as stated by Dougherty (2012) and García et al. (2014). Apart from the company name and observation date, there are two categorical features left indicating the financial quarter and the activity status respectively. Both features are transformed into separate dummy variables such that all possible states are still covered numerically.

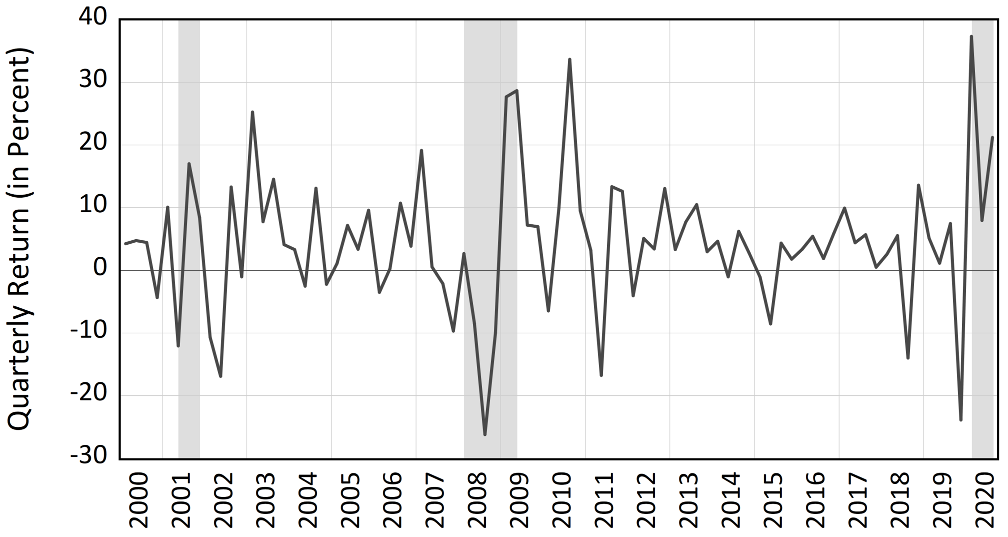

Figure 3.1: Equally weighted returns of all stocks (shaded areas are NBER-defined recession periods)

In terms of outliers, values that fall outside of 3 standard deviations from the mean are evaluated more closely before deciding if it will be part of the final dataset. Only in the third quarter of 2009 a stock observation is actually removed, which corresponds to Charter Communications, Inc. After filing for bankruptcy at the end of the first quarter that year, the company restructured financially reducing its debt by several billion dollars. As a result they obtained a documented quarterly return of around 1,118%, which is significantly higher than the average return of 7.2% amongst all other stocks that period. Figure 3.1 displays the equally weighted return of the historical S&P stocks that are part of the final dataset, indicating the total quarterly return one would have achieved when rebalancing their portfolio accordingly to the changes in the index. Apart from the periods surrounding the Lehman bankruptcy in 2008, the so-called flash crash in 2010 and the Covid Crisis at the start of 2020, the index remains mostly positive and fluctuates around a mean of roughly 4.3%. The corresponding compounded return from 2000 to 2020 lies around 363%, demonstrating the thoroughness behind the S&P admittance standards.

3.2 Feature Engineering

The feature engineering step involves the construction of the dependent variable. In particular, its structure should characterize the stock selection problem whilst also being suitable for the predictive models. Instead of individual stock returns, Rasekhschaffe and Jones (2019) state that forecasting discrete variables is more efficient in a machine learning context as it limits the risk of overfitting. Therefore, this research categorizes the out-performing and under-performing stocks through a 5 percent quantile selection. This implies that all stocks with a positive (negative) return that are part of the top (bottom) 5 percent are labelled as *long* (*short*) position stocks in the training set.

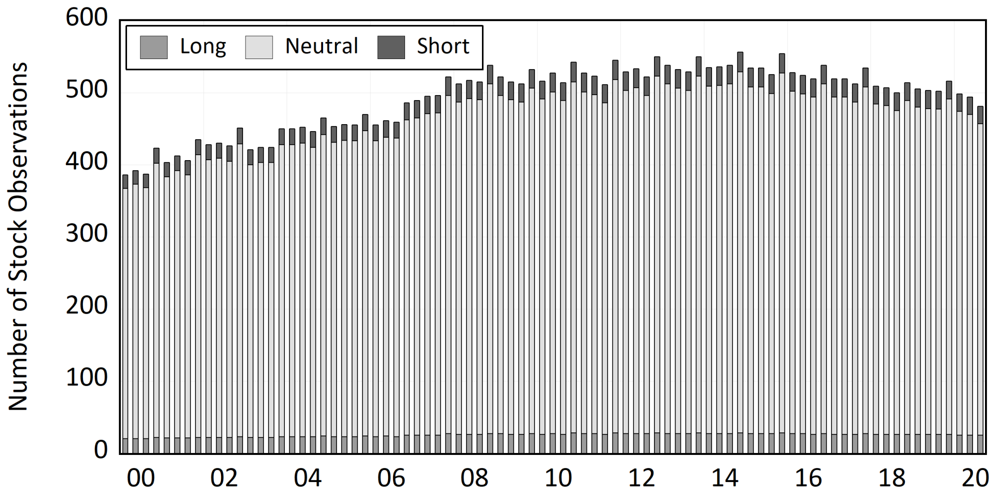

Figure 3.2: Class distribution of available stocks for a 5 percent quantile selection

Figure 3.2 depicts the resulting class distribution across the different observation periods. The stocks labelled as neutral are removed from the training process, as current literature (Huerta et al. (2013) and Fu et al. (2018)) indicates that these observations are likely to decrease the out-of-sample performance due to extra noise. Individual classifiers, therefore, only train on those stocks that hold feature values which are conventional to out- and under-performing assets. To ensure that there is no selection-bias, all available stocks are taken into account within the testing period and a *long* (*short*) position label is assigned to each stock with a positive (negative) return. This strategy simplifies the equity trading context to a binary classification problem. Due to the admittance standards of the S&P 500 index, large positive returns are far more likely to occur within this investable universe than large negative returns. As a result, incorporating short positions without any restrictions might lead to substantial performance reduction inside an equally weighted allocation.

Hence, several lower bounds on the class probability of *short* position forecasts are assessed when rebalancing the portfolio based on the model predictions. The objective behind evaluating different probability bounds is to find a lower limit that restricts the portfolio from going short in risky stocks (e.g. stocks that hold less conventional feature values for *short* positions), while also maximizing the compounded return over time. Note that this optimal certainty threshold may differ across the various classifiers as it depends on how well a model can diversify in terms of probability.

Finally, the machine learning models should be trained on observations that are likely to be representative for the upcoming financial state of the stock market. In general, quarters that are closer to the testing period consist of stocks that appertain to the same macro-economic conditions. However, due to the quantile selection smaller windows might not contain enough training observations and therefore classifiers may possibly fail to capture the applicable market conditions. Increasing the rolling window will result in more training samples, but also gives rise to the possibility of equally weighting current and outdated conditions. To optimize performance, each model is examined for different rolling-windows ranging from 1 to 9 years. For simplicity, their results are categorized in 3 size groups: small (1 to 3), middle (4 to 6) and large (7 to 9).

3.3 Feature Selection

The last step of data pre-processing involves taking care of the dimensionality problem. In particular, high-dimensional datasets often contain irrelevant and redundant features that diminish the efficiency of machine learning algorithms. Guyon and Elisseeff (2003) indicate that including such features during training commonly leads to an increase in computational time and possible overfitting. Therefore, only features that are well associated with future stock returns should be involved throughout the forecasting process in order to enhance the classification performance.

During this research the dimensionality is reduced by means of a selection framework based on feature importance. In particular, the Adaboost method is incorporated as applied by Zhu et al. (2009) with one-level DTs as the weak classifier. Taking into account the stochastic nature of the algorithm, I then use a validation set and implement stratified 5-fold cross-validation to obtain a ranking of the various stock related features through their individual error reduction. Subsequently, the first combination of features that is capable of obtaining a minimum Area Under the Curve of 65% is sought by iteratively adding the highest ranked feature using a Specific-to-General approach. Furthermore, the feature selection is completed before the construction of the interpretable classifiers, because this allows the models to be trained and evaluated on the exact same dataset.

Table 3.1: Overview of the average dimension reduction across different rolling windows

Window	**Small**	**Middle**	**Large**
Avg.	20.34	26.81	27.11
Min.	8	12	12
Max.	33	37	38

Table 3.1 gives an overview of the dimension reduction that results from this procedure. On average the number of features decreases significantly from the initial 107 to around 25 remaining variables. The total also seems to exhibit a minor increase for larger training windows. However, around periods of economic growth the number of substantial features consistently tends to a minimum of 8 to 12 variables and in the course of a recession it inclines to slightly more than 30.

4 Methodology

In this section, both the classification methods and the evaluation techniques of the research are defined. The first part concentrates on rule-learning through Decision Trees, it includes the CART algorithm and the Random Forest model which is implemented as the machine learning benchmark. This is followed by classifiers based on alternative learning approaches, namely: the Classy, Ripper and RUG algorithm. Finally, the metrics that are needed to assess the classification performance and the portfolio results are discussed.

4.1 Decision Trees

Currently, rule-based classification is regarded as the backbone of interpretable machine learning. The transparent structure that is imposed by decision rules resembles our natural language semantically, which allows them to be interpreted as independent *if-then* statements. In particular, Lakkaraju et al. (2016) define an individual rule r as a tuple $(\mathcal{S}, y^{(r)})$ consisting of an itemset $\mathcal{S}$ that filters the stocks and a general class label $y^{(r)} \in \mathcal{Y}$ that is assigned to all the observations covered by rule r. Within the binary classification framework of this research, the set of possible classes $\mathcal{Y}$ is defined as either taking a *long* or *short* position in the stock. The filtering process is constructed as a conjunction of one or more predicates that take the following form: (feature, operator, threshold). Given a set of stock observations $\mathbf{x_i} \in \mathbb{R}^n$, $i = 1, \ldots, N$, a possible predicate θ could be $(x_j > 10)$ with x_j denoting the jth feature of the stock. Hence, the structure of some rule r consisting of two predicates can intuitively be displayed as:

$$\text{Rule } r : \textbf{IF } \theta_1 \text{ and } \theta_2 \text{ hold } \textbf{THEN} \text{ assign class } y^{(r)}.$$

One of the rule-learning techniques that is frequently applied today is the so-called Classification And Regression Trees (CART) algorithm, as introduced by Breiman et al. (1984). During this approach the feature space of a training set $\mathcal{D} = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_N}, y_N)\}$ is split in binary fashion by means of recursive partitioning. Specifically, denote $Q_m \subset \mathcal{D}$ as the remaining N_m observations at the m^{th} node of the DT and predicate $\theta = (j, t_m)$ a possible split of the feature space involving feature j and threshold t_m. Based on θ the stocks in Q_m can be split into the following subsets:

$$Q_m^{(0)}(\theta) = \{(\mathbf{X}, \mathbf{y}) | x_{ij} \leq t_m\} \quad \text{and} \quad Q_m^{(1)}(\theta) = Q_m \setminus Q_m^{(0)}(\theta) \ , \tag{1}$$

where all $\mathbf{x}_i \in Q_m$ that satisfy $(x_{ij} \leq t_m)$ are included in $Q_m^{(0)}$ and the remaining observations are covered in $Q_m^{(1)}$. The CART algorithm finds the best possible predicate θ^* to split the data at node m according to the minimization of a pre-specified impurity measure $H(\cdot)$:

$$\theta^* = \text{argmin}_\theta \ G(Q_m, \theta) \quad \text{with} \quad G(Q_m, \theta) = \frac{N_m^{(0)}}{N_m} H(Q_m^{(0)}(\theta)) + \frac{N_m^{(1)}}{N_m} H(Q_m^{(1)}(\theta)) \ , \tag{2}$$

here $N_m^{(0)}$ and $N_m^{(1)}$ represent the number of observations in $Q_m^{(0)}(\theta)$ and $Q_m^{(1)}(\theta)$ respectively. In this research, the quality of each possible split is measured by means of the Gini index:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}) \quad \text{with} \quad p_{mk} = \frac{1}{N_m} \sum_{y \in Q_m} \mathbb{I}(y = k) \ , \tag{3}$$

where p_{mk} represents the ratio of observations in Q_m that belong to class $k \in \mathcal{Y}$. Hence, $H(Q_m)$ measures the probability of a certain observation being falsely classified when it is randomly chosen. After obtaining the subsets $Q_m^{(0)}(\theta^*)$ and $Q_m^{(1)}(\theta^*)$, the algorithm continues splitting iteratively until either a maximum number of nodes m_{max} is reached or the minimum amount of allowed observations N_{min} is attained. To find the optimal values for m_{max} and N_{min}, a grid-search is performed using a stratified 5-fold cross-validation. As the feature space is split into two complementary subspaces at each node through a pair of contrasting predicates, the structure of the resulting rules can be interpreted as dependent *if-then-else* statements:

$$\text{Rule } r_{1,2} : \textbf{IF } \theta_1 \text{ holds } \textbf{THEN} \text{ assign class } y^{(r_1)} \textbf{ ELSE } \text{ assign class } y^{(r_2)}.$$

Constructing an ordinary DT through the CART algorithm is almost effortless and classification of new observations can be done rapidly. In addition, the resulting decision boundaries are relatively transparent if they are built from intelligible features and the length of the rules is not too large. However, the user study of Lakkaraju et al. (2016) indicates that this nested predicate structure decreases the transparency of the rules significantly when the number of performed splits increases.

4.2 Random Forest

In general, ordinary DTs have two main disadvantages that make them less favourable in practice. Due to the hierarchical structure in which rules are constructed within the CART algorithm, each itemset s consists of a nested conjunction of predicates. As a result, individual DTs are prone to over-fitting and very sensitive to changes in the training data. To overcome said shortcomings, Breiman (2001) proposes the RF model in order to obtain a more robust tree-based classification approach. The main goal of this method is to construct an ensemble of uncorrelated DTs, which produces more robust forecasts as it allows the different trees to protect one another from their individual classification errors.

Algorithm 4.1: Random Forest

```
Input:          $\mathcal{D} = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_N}, y_N)\}$
Process:
1: for $z = 1, \dots, Z$ do
2:     Draw bootstrap sample $\mathcal{Z}$ of size $N$ from $\mathcal{D}$.
3:     Grow decision tree $T_z$ from $\mathcal{Z}$ as follows:
4:     until $m_{max}$ reached or $N_m = N_{min}$ do
5:         Select $p$ random features from the $n$ total features.
6:         Find optimal predicate $\theta^*$ among these $p$ features.
7:         Split $Q_m$ based on $\theta^*$ into $Q_m^{(0)}(\theta^*)$ and $Q_m^{(1)}(\theta^*)$.
8:     end
9: end
Output:          The ensemble $\{T_z\}_1^Z$.
Classification:  Let $\hat{y}^{(z)}(\mathbf{x_i})$ be the class prediction of T$_z$.
                 Then $\hat{y}^{(rf)}(\mathbf{x_i}) = majority\ vote\ \{\hat{y}^{(z)}(\mathbf{x_i})\}_1^Z$.
```

Note: Pseudocode of the Random Forest algorithm is based on Hastie et al. (2009).

Algorithm 4.1 illustrates the RF model within the classification scenario of this research. Given a training set $\mathcal{D}$ consisting of N stock observations, the algorithm first generates a specified number (Z) of random training samples $\mathcal{Z}$ through bootstrap aggregation (bagging).[1] For each sample $z \in \mathcal{Z}$, the model then creates an individual DT denoted as T_z. The procedure for constructing trees is similar to the CART algorithm, however the RF approach differs slightly as it searches for the best predicate θ^* amongst a random selection of $p < n$ features at each node m. Due to the unstable nature of DTs, this combination of bagging and randomization of the features results in the required uncorrelated tree structures. Finally, let $\hat{y}^{(z)}(\mathbf{x}_i)$ be the prediction from T_z for stock observation $\mathbf{x}_i$ and $\{T_z\}_1^Z$ the fully trained ensemble. The RF algorithm makes its final prediction

[1]See Breiman (1996) for a detailed explanation of the bootstrap sampling method applied to DTs.

$\hat{y}^{(rf)}(\mathbf{x}_i)$ by applying a majority vote across all $\hat{y}^{(z)}(\mathbf{x}_i) \in \{\hat{y}^{(z)}(\mathbf{x_i})\}_1^Z$. It is important to mention that the RF approach loses the transparency factor that the DTs initially offered, as it provides a large set of overlapping decision rules that are often hard to explain in practice. Conversely, this structure allows for more robust predictions and is found throughout literature to realize a higher out-of-sample performance.

4.3 Classy Algorithm

An alternative rule-based learning approach is the so-called probabilistic rule lists. Similarly to an ordinary rule r, a probabilistic rule r_p can be defined as a tuple that consists of an itemset $\mathcal{S}$ which filters the stocks as a conjunction of one or more predicates. However, instead of a general class label $y^{(r)}$ it assigns a categorical distribution $\Phi(\mathcal{S})$ over both the *long* and *short* class:

$$\Phi(\mathcal{S}) = (\phi^{long}, \phi^{short}) \quad \text{such that} \quad \phi^{long} + \phi^{short} = 1 \quad \text{and} \quad \phi^{long},\ \phi^{short} > 0. \tag{4}$$

Hence, a probabilistic rule that consists of an itemset with two predicates can be displayed as:

$$\text{Rule } r_p : \textbf{IF } \theta_1 \text{ and } \theta_2 \text{ hold } \textbf{THEN } y^{(r_p)} \sim \Phi(\mathcal{S}) = (\phi^{long}, \phi^{short}).$$

To retrieve class probabilities, ordinary decision rules integrate a frequentist approach and simply make use of the resultant accuracy of a rule during training. Alternatively, probabilistic rules take a Bayesian attitude which might lead to more precise information regarding the certainty of an out-of-sample forecast. In terms of structure, a probabilistic rule list R can be interpreted as an *ordered* list of *if-then* statements that ends with a default rule $r_{p,\varnothing}$ such that the list covers the entire feature space. For instance:

$$\begin{aligned}
&\text{Rule } r_{p,1}: \ \textbf{IF } \theta_1 \text{ and } \theta_2 \text{ hold } \textbf{THEN } \Pr(\textit{long}) = 0.65 \text{ and } \Pr(\textit{short}) = 0.35\\
&\text{Rule } r_{p,2}: \ \textbf{IF } \ \theta_3 \text{ holds } \textbf{THEN } \Pr(\textit{long}) = 0.20 \text{ and } \Pr(\textit{short}) = 0.80\\
&\text{Rule } r_{p,\varnothing}: \ \textbf{ELSE } \ \Pr(\textit{long}) = 1.00.
\end{aligned}$$

Note that a stock observation $\mathbf{x}_i$ is classified by going through the list top-down and assigning the class with the highest probability level as the preferred stock position. By design, the model automatically executes the default rule if none of the previous itemsets are applicable. For the sake of simplicity, the support of a rule its itemset $supp(\mathcal{S})$ is defined as the number of times that the predicates are satisfied within the training data $\mathcal{D}$ and the usage $U(\mathcal{S})$ is equal to the support subtracted by the number of stock observations covered by the preceding rules in R.

Proença and van Leeuwen (2020) introduce the Classy algorithm, which generates a probabilistic rule list based on the MDL principle. Given the complete space of possible lists $\mathcal{R}$, their methodology is tailored to find the optimal classifier $R \in \mathcal{R}$ that minimizes:

$$\mathcal{L}(\mathcal{D}, R) = \mathcal{L}(\mathbf{y}|R, \mathbf{X}) + \mathcal{L}(R), \tag{5}$$

where $\mathcal{L}(R)$ denotes the encoded Description Length (DL), in bits, of the probabilistic rule list and $\mathcal{L}(\mathbf{y}|R, \mathbf{X})$ represents the DL of the class labels given the rule list and the set of available stock observations in $\mathcal{D}$. The class labels are encoded by means of the Normalized Maximum Likelihood approach (Shtarkov (1987)), while the length of the rule list is retrieved by using both the universal code for integers (Rissanen (1983)) and the uniform code (Grünwald et al. (2007)). As more complex lists result in a larger total DL, this objective function imposes a trade-off between the complexity of the model and the corresponding fit.

Algorithm 4.2: Classy

```
Input: $\mathcal{D} = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_N}, y_N)\}$ and $\mathcal{C} = \{r_{p,1}, \dots, r_{p,k}\}$
Process:
1: Remove strictly redundant itemsets from $\mathcal{C}$
2: Set the default rule: $R^* = \{r_{p,\varnothing}\}$
3: until $\delta\mathcal{L}(\mathcal{D}, R^* \cup \{r'_p\}) \leq 0 \ \forall \ r'_p \in \mathcal{C}$ do
4:     $r^*_p = \text{argmax}_{r'_p \in \mathcal{C}} \ \delta\mathcal{L}(\mathcal{D}, R^* \cup \{r'_p\})$
5:     $R^* = R^* \cup \ \{r^*_p\}$
6:     Update $\mathcal{C}$
7: end
Output:   The probabilistic rule list $R^*$.
```

Note: Pseudocode of the Classy algorithm is based on Proença and van Leeuwen (2020).

In order to find the optimal list that minimizes Equation (5), the algorithm evaluates the normalized compression gain that is realized by adding an extra rule to the list:

$$\delta\mathcal{L}(\mathcal{D}, R \cup \{r_p\}) = \frac{\mathcal{L}(\mathcal{D}, R) - \mathcal{L}(\mathcal{D}, R \cup \{r_p\})}{U(\mathcal{S})}, \tag{6}$$

here the numerator denotes the absolute compression gain. Including the usage helps the model favour those rules that cover less stock observations yet provide a higher accuracy relative to the absolute gain. The Classy approach, as depicted in Algorithm 4.2, finds the best probability rule list R^* by means of the separate-and-conquer strategy. In particular, given a set of candidate rules $\mathcal{C} = \{r_{p,1}, \dots, r_{p,k}\}$ it iteratively adds the rule that inflicts the largest change in normalized compression until there are no more existing rules that lead to a positive gain. After each additional rule, the usage of the remaining candidate rules is updated by removing the stock observations they have in common with the prior added rule. To obtain the candidate rules, Proença and van Leeuwen (2020) mine frequent itemsets using the beam-search algorithm. This decreases the computational time of the mining procedure with several magnitudes compared to classical approaches such as FP-growth and the Apriori algorithm, while finding an almost identical set of rules. Furthermore, the corresponding categorical distributions are calculated using a smoothed maximum likelihood estimator:

$$\hat{\phi}^k = \frac{U(\mathcal{S}, k) + \xi}{U(\mathcal{S}) + |\mathcal{Y}|\xi}, \tag{7}$$

where $U(\mathcal{S}, k)$ represents the usage of the itemset taking into account only the training observations that are labelled as class $k \in \mathcal{Y}$ and ξ a small pseudo count to avoid division by zero errors even when there is no class-specific usage. After finalizing the candidate set, strictly redundant rules are easily removed by means of their anti-monotone support to improve the computational time of the model. Specifically, if there are two rules such that $\mathcal{S}_1 \subset \mathcal{S}_2$ then the first rule is removed as it will never be preferred due to $supp(\mathcal{S}_2) \geq supp(\mathcal{S}_1)$ which by encoding always results in a larger DL for the first rule. Lastly, the main benefit of the Classy algorithm is that, by means of the objective function, it naturally considers a trade-off between complexity and accuracy. In addition, it simultaneously avoids overfitting and the need for hyperparameter tuning.

4.4 Ripper Algorithm

The Ripper algorithm is an inductive rule learning approach that is widely adopted in the field of machine learning. As illustrated by Cohen (1995), its main goal is to reduce the classification error through pruning, while also maintaining a simplistic set of rules and preventing the model from overfitting. To accomplish this, the classification procedure consists out of two phases, namely: rule construction and rule optimization. Similarly to the framework of probabilistic rule lists, this methodology adopts a default forecast. In this research, a *short* position is set as the default class in every observation period, since in general there are slightly more stock observations with a negative return across the different quarters. Due to the binary set-up of the classification problem, the model only searches for rules that classify observations that are part of the *long* position class.

During the first phase of the Ripper classifier, rules are constructed using the Sequential Covering Algorithm (SCA). In particular, a rule r is built by greedily adding the best possible predicate θ^* to the itemset $\mathcal{S}$ until all remaining stock observations are *long* positions. To determine θ^*, each possible predicate within the feature space is evaluated by means of the FOIL's Information gain:

$$FOIL(\mathcal{S}_0, \mathcal{S}_1) = L(\log_2 \frac{l_1}{l_1 + s_1} - \log_2 \frac{l_0}{l_0 + s_0}) \ , \tag{8}$$

where l_c and s_c denote the number of *long* and *short* observations covered by the predicates in $\mathcal{S}_c$ respectively for $c = 0, 1$. Furthermore, $\mathcal{S}_0$ indicates the current itemset while $\mathcal{S}_1$ includes an additional predicate $\theta \notin \mathcal{S}_0$ and L is the total number of *long* position stocks covered by both $\mathcal{S}_0$ and $\mathcal{S}_1$. In order to prevent overfitting, the final itemset $\mathcal{S}^*$ is incrementally pruned by assessing the following cover-ratio:

$$V(\mathcal{S}) = \frac{L + 1}{L + S + 2}, \tag{9}$$

with S denoting the total number of short position stocks covered by $\mathcal{S}$. Starting at the last added predicate $\tilde{\theta}$, the model re-evaluates iteratively if a predicate should be included by replacing $\mathcal{S}$ if $V(\mathcal{S} \setminus \tilde{\theta}) \geq V(\mathcal{S})$. After finalizing a rule r, the model removes all covered stock observations from the training set and starts building a new rule until one of the stopping criteria is met:[2]

1. There are no more *long* position stock observations left in the training set.
2. The misclassification rate of the current set of rules exceeds 50%.
3. The DL of the entire set of rules is 64 bits greater than the smallest DL observed so far.

During the second phase of the Ripper algorithm, the obtained set of rules is optimized by constructing and pruning two additional variants of each individual rule from a random selection of the data. One of the new variants is constructed from an empty rule, while the other is acquired by greedily adding predicates to the original rule. The model selects the rule with the smallest DL as the final representative amongst the three variants. Instead of Equation (9), the accuracy measure is implemented as the pruning metric throughout the entire second phase of the algorithm. Note that the recurrent use of performance evaluation throughout the training process of the Ripper algorithm results in higher empirical performance as it reduces the possibility of overfitting.

4.5 Rule Generation Algorithm

Akyüz and İlker Birbil (2021) propose the RUG algorithm in which they combine the interpretive aspect of an ordinary DT with linear programming, making their approach scalable for large datasets. Compared to most other optimization-based classification approaches, this model can be applied to multivariate class scenarios. In order to accomplish this, they characterize a vector-valued mapping for the classes. Within a binary problem the formulation of this class vector and its prediction can be simplified as:

$$\vec{\mathbf{y}}(\mathbf{x}_i) = \begin{cases} (1,-1), & \text{if } y_i = \textit{long} \\ (-1,1), & \text{if } y_i = \textit{short} \end{cases} \quad \text{and} \quad \vec{\mathbf{y}}^*(\mathbf{x}_i) = \sum_{r \in \mathcal{R}_t} a_{ir} \mathbf{R}_r(\mathbf{x}_i) w_r, \tag{10}$$

here $\vec{\mathbf{y}}^*(\mathbf{x}_i)$ denotes the forecast of stock observation $\mathbf{x}_i \in \mathcal{D}$ with $a_{ir} \in \{0,1\}$ specifying if a certain rule $r \in \mathcal{R}_t$ at iteration t covers asset i and w_r a set of nonnegative weights. Moreover, vector $\mathbf{R}_r(\mathbf{x}_i)$ is constructed in the same fashion as $\vec{\mathbf{y}}(\mathbf{x}_i)$ and is assigned to observation $\mathbf{x}_i$ by rule r. Note that the largest element of $\vec{\mathbf{y}}^*(\mathbf{x}_i)$ is ultimately appointed as the final position forecast $\hat{y}_i$ of the stock.

[2]These stopping criteria correspond to those applied in the Ripper software distribution of Witten et al. (2016).

The main goal of this methodology is to minimize the total classification error by means of the following linear programming model:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i\in\mathcal{D}} v_i + \sum_{r\in\mathcal{R}_t} c_r w_r \\
\text{subject to} \quad & \sum_{r\in\mathcal{R}_t} \hat{a}_{ir} w_r + v_i \geq 1, \quad && i \in \mathcal{D}; \\
& \sum_{r\in\mathcal{R}_t} a_{ir} w_r \geq \epsilon, && i \in \mathcal{D}; \\
& v_i \geq 0, && i \in \mathcal{D}; \\
& w_r \geq 0, && r \in \mathcal{R}_t;
\end{aligned} \tag{11}$$

here the auxiliary variable $v_i, i \in \mathcal{D}$ imposes the said objective through the first set of constraints by evaluating the so-called hinge loss function:

$$\sum_{i\in\mathcal{D}} \max\left\{1 - \sum_{r\in\mathcal{R}_t} \hat{a}_{ir} w_r, 0\right\}, \tag{12}$$

where $\hat{a}_{ir} = \frac{1}{2}\left\{a_{ir}\mathbf{R}_r(\mathbf{x}_i)^T \vec{\mathbf{y}}(\mathbf{x}_i)\right\}$.[3] In addition, the cost coefficients $c_r \geq 0, r \in \mathcal{R}_t$ penalize rules with more predicates and avoid outcomes with a large number of rules holding nonzero weights. Hence, this method encourages interpretability as it tries to find a small and concise group of decision rules. The second set of constraints in model (11) ensure that all stock observations $\mathbf{x_i} \in \mathcal{D}$ are covered, which is essential for obtaining consistent and interpretable decision boundaries. Note that different values of ϵ could alter the optimal rule set, however Akyüz and İlker Birbil (2021) indicate that it is sufficient to set it equal to a small strictly positive value (e.g. 0.01) as these constraints simply regard to coverage.

```
Algorithm 4.3: Rule Generation
Input:    $\mathcal{D} = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_N}, y_N)\}$
Process:
1: Grow $T_{(e)}$ and extract initial rule set $\mathcal{R}_0$
2: until $\mathcal{R}_- = \emptyset$ do
3:     Solve Dual Model (13) and obtain optimal $\beta^{(t)}$
4:     Grow $T_{(\beta^{(t)})}$ and extract $\mathcal{R}_-$
5:     $\mathcal{R}_t = \mathcal{R}_{t-1} \cup \mathcal{R}_-$
6: end
Output:   The final set of rules $\mathcal{R}$.
```

Note: Pseudocode of the RUG algorithm is based on Akyüz and İlker Birbil (2021).

[3]The generalized formulation of Akyüz and İlker Birbil (2021) involves the term $\kappa = (|\mathcal{Y}| - 1)/|\mathcal{Y}|$, which simplifies to 1/2 for binary classification.

Algorithm 4.3 illustrates the steps of the RUG approach within the classification scenario of this research. Given a training set $\mathcal{D}$, the initial rule set $\mathcal{R}_0$ is obtained by creating a DT based on a weight vector e consisting of only ones $T_{(e)}$. Thereafter, the model iteratively generates rules through an optimization procedure known as column generation. At each iteration t, the dual formulation of (11) is solved based on the current rule set $\mathcal{R}_{t-1}$:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in \mathcal{D}} (\beta_i + \epsilon \gamma_i) \\ \text{subject to} \quad & \sum_{i \in \mathcal{D}} (\hat{a}_{ir} \beta_i + a_{ir} \gamma_i) \leq c_r, \quad && r \in \mathcal{R}_t; \\ & 0 \leq \beta_i \leq 1, && i \in \mathcal{D}; \\ & \gamma_i \geq 0, && i \in \mathcal{D}, \end{aligned} \tag{13}$$

here $\beta_i, i \in \mathcal{D}$ represent the dual variables of the loss constraints, while $\gamma_i, i \in \mathcal{D}$ correspond to the dual variables of the coverage constraints. After obtaining the optimal solution $(\boldsymbol{\beta}^{(t)}, \boldsymbol{\gamma}^{(t)})$, it tries to improve the current rules by solving its pricing subproblem. In particular, rules with negative reduced costs are identified by constructing an additional DT using the previously found $\boldsymbol{\beta}^{(t)}$. A rule $r^{'}$ that is part of the newly obtained set $\bar{\mathcal{R}}$ is said to improve the current objective value if it satisfies:

$$\bar{c}_{r'} = c_{r'} - \sum_{i \in \mathcal{D}} (\hat{a}_{ir} \beta_i^{(t)} + a_{ir} \gamma_i^{(t)}) < 0 \quad \text{with} \quad r' \in \bar{\mathcal{R}} \setminus \mathcal{R}_{t-1}, \tag{14}$$

where $\bar{c}_{r'}$ defines the reduced cost of rule $r^{'}$. Now, denote $\mathcal{R}_{-}$ as the set of rules that satisfy Equation (14) and have a negative reduced cost. The algorithm continues at iteration $t+1$ with $\mathcal{R}_t = \mathcal{R}_{t-1} \cup \mathcal{R}_{-}$ and terminates if it cannot find a single rule that improves te current objective function, that is $\mathcal{R}_{-} = \emptyset$. As previously mentioned, the construction of an ordinary DT can be completed rapidly and therefore the computational time of RUG is very low compared to state-of-the-art machine learning models. In addition, the algorithm obtains a set of rules that is larger than that of a DT, however the rules are generally shorter which supports interpretability. Furthermore, the model is found capable of performing at the same level as the RF algorithm in terms of accuracy.

4.6 Evaluation Metrics

The forecasts of the various interpretable learning algorithms are assessed with respect to both their classification and portfolio performance. The latter is evaluated by means of the average quarterly return and the total compounded return that results from re-balancing an equally weighted portfolio based on the position forecasts of the classifiers. In order to evaluate how well the models can differentiate between *long* and *short* positions, the following metrics are taken into account: the Accuracy (ACC), Recall (RCL), Specificity (SPC), F_1-score and their precision in terms of

Positive Predicted Value (PPV) and Negative Predicted Value (NPV). Note that all five measures are essentially based on the confusion matrix, depicted in Figure 4.1.

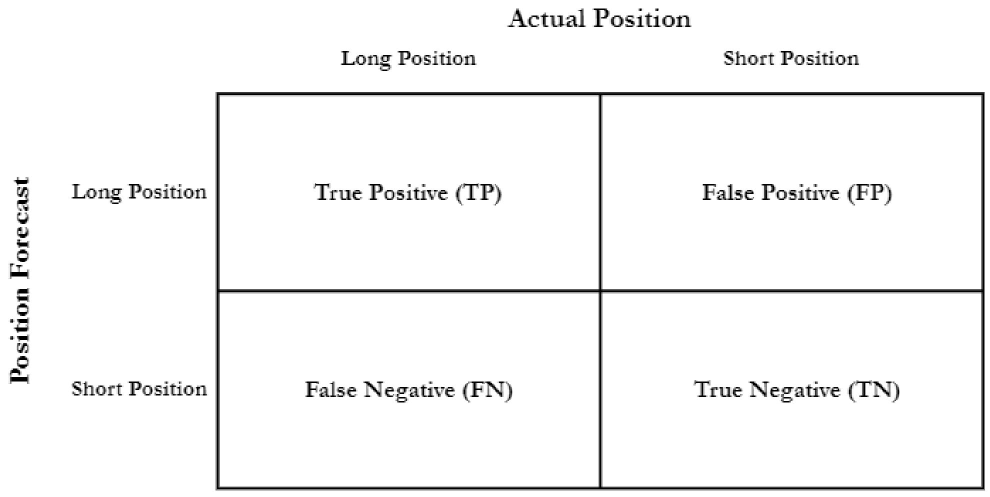

Figure 4.1: Illustration of the confusion matrix within a stock selection framework

In the context of this research, the positive and negative class correspond to *long* and *short* positions respectively. Therefore, a True Positive (TP) observation is defined as the prediction of a *long* position for a stock that exhibits a positive return over the next quarter. Conversely, True Negative (TN) observations are defined as predicting *short* positions for stocks that have negative future returns. Both scenarios lead to a positive return on their investment, while False Positive (FP) and False Negative (FN) position forecasts clearly hold negative returns.

The ACC of a classifier indicates the proportion of stocks that are assigned a forecast position that coincides with its future return, in other terms:

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}, \tag{15}$$

where a higher value implies that the model captures a larger portion of the underlying relations between a firm's fundamental characteristics and its future price. However, if a model does not distinguish between *long* and *short* positions and only concentrates on one particular class it could still obtain a high ACC in periods of economic expansion or turmoil. Therefore, it is also beneficial to consider the fraction of correctly classified stock positions for each individual class:

$$RCL = \frac{TP}{TP + FN} \quad \text{and} \quad SPC = \frac{TN}{TN + FP}, \tag{16}$$

here the RCL measures the fraction of correctly classified *long* positions and the SPC the fraction of correctly classified *short* positions. Furthermore, portfolio construction is a practice in which investors need to be overly cautious with their investment strategies as taking positions with too much risk may harm the trust of their customers. Note that the trade-off between RCL and SPC does not demonstrate to what extent a model is more exact in classifying a certain position class. For that reason, the corresponding forecast precision is expressed by means of the PPV and NPV:

$$PPV = \frac{TP}{TP + FP} \quad \text{and} \quad NPV = \frac{TN}{TN + FN}, \tag{17}$$

where the PPV illustrates the proportion of *long* position forecasts that actually exhibit a positive return over the next quarter, while the NPV concentrates on short positions for stocks with negative future returns. Finally, the F_1-score is used to exemplify the trade-off between this precision and the fraction of correctly classified stocks for a certain investment position:

$$F_1^{(k)} = \begin{cases} 2 \times \frac{PPV \times RCL}{PPV + RCL}, & \text{if } k = \textit{long} \\ 2 \times \frac{NPV \times SPC}{NPV + SPC}, & \text{if } k = \textit{short} \end{cases}, \tag{18}$$

which can be interpreted as the corresponding harmonic mean. In particular, for higher RCL (SPC) or PPV (NPV) values the F_1-score of *long* (*short*) positions goes to 1 while for lower values it comes closer to 0.

To assess if the provided return series of the model-based allocations are significantly different from the market portfolio, their respective means are tested for equality using the two-sample t-test introduced by Snedecor and Cochran (1989):

$$\frac{\bar{R}_C - \bar{R}_M}{\sqrt{2(s^2/P)}} \sim t_{1-\alpha/2,v} \quad \text{with} \quad s^2 = \frac{\sum_{p=1}^{P}(R_C^{(p)} - \bar{R}_C)^2 + \sum_{p=1}^{P}(R_M^{(p)} - \bar{R}_M)^2}{2(P-1)}, \tag{19}$$

here P indicates the total number of periods that are investigated and the test statistic follows a Student's t-distribution with v degrees of freedom at a significance level of α. Furthermore, $R_C^{(p)}$ and $R_M^{(p)}$ denote the return on investment at quarter p provided by the allocation of the classifier and market respectively, while $\bar{R}_C$ and $\bar{R}_M$ are their corresponding average quarterly returns. The volatility of the model-based portfolios is compared with the volatility of the market allocation by means of an F-test for equality of two variances as covered by Heij et al. (2004):

$$\frac{\sum_{p=1}^{P}(R_M^{(p)} - \bar{R}_M)^2}{\sum_{p=1}^{P}(R_C^{(p)} - \bar{R}_C)^2} \sim F(v_M, v_C), \tag{20}$$

where the test statistic follows an F-distribution with the degrees of freedom for both the model-based allocation (v_C) and market portfolio (v_M) equal to $P - 1$. Note that all statistical tests are performed using the EViews 10 software.

4.7 Certainty Threshold

The current financial literature regarding stock classification has primarily focussed on constructing portfolios based on *long* positions. Recall that the investable universe, being the historical constituents of the S&P 500 index, denotes the set of assets in which we are allowed to invest. As previously explained, large positive returns are far more likely to occur within the investable universe of this research than large negative returns. Including *short* positions at the same rate as *long* positions could therefore harm the portfolio, as misclassifying *short* positions might possibly have a more damaging effect than misclassifying *long* positions.

To solve the issue, this paper evaluates the addition of a minimum class probability that needs to be exceeded before a *short* position forecast is included in the portfolio. Using such a 'certainty threshold' will expectantly force the model-based allocations to take fewer and less risky *short* positions. The optimal threshold for each classifier is set as the minimum lower bound that achieves a NPV of at least 80% during training amongst the short position forecasts that lie above the threshold. Here, the general performance is estimated using stratified 5-fold cross validation and the lower bound is chosen from the set $\{5x \mid x \in \{11, \ldots, 19\}\}$. Note that the certainty threshold might differ across the various learning algorithms, as the range of class probabilities depends on how well a model is capable to differentiate between safe and risky investments.

5 Results

This section describes the computational platform of the experiment and investigates the performance of the classification models in three stages. The first part concentrates on the interpretability of the algorithms with respect to the RF model by evaluating the structure of their resulting rule sets. This is followed by assessing the classification performance regarding both *long* and *short* positions for different rolling windows. Finally, the portfolio returns of the interpretable learning models are compared with the market allocation and the impact of including *short* positions through a certainty threshold is analysed.

5.1 Computational Platform

All algorithms are implemented in Python 3.8.5 using the Spyder IDE. The modules are executed on a Windows 10 pro computer with an Intel Core i7-2600 CPU at 3.40 GHz, 16.0 GB of RAM and a 64-bit Operating System. Table 5.1 provides an overview of the specific packages that were used to construct each classifier. As previously mentioned, the hyperparameters of the models are optimized in each quarter by means of a grid-search (if applicable). The general performance of each possible combination of parameters is estimated within the training set using stratified 5-fold cross validation. After the optimal set of hyperparameters is found, the performance of the optimized model is determined by training the classifier on all available training observations. Note that the model-based portfolio allocations are formed using this final class prediction.

Table 5.1: Overview of the Python packages that are used to construct the rule-learning models

Model	**Package**	**Distribution**	**Additional Requirements**
RF / CART	scikit-learn	anaconda/pip	scipy, joblib, numpy and threadpoolctl
Classy	rulelist	pip	scipy, typing, scikit-learn and gmpy2
Ripper	python-weka-wrapper3	pip	javabridge, OpenJDK 11 and Weka 3.9.5
RUG	rulediscovery	anaconda	numpy, pandas, scikit-learn, cvxpy, cvxopt and gurobi

Regarding the CART algorithm, m_{max} and N_{min} are both evaluated for all integer values in the range of $[2, 20]$. The optimal combination of hyperparameters that is found for CART is also used for the RUG algorithm, as the RUG utilizes a DT based on CART to construct its rules. Moreover, the Gurobi solver is used to solve the linear programming problems from RUG. The number of trees used in the RF model is chosen from the set $\{25s \mid s \in \{1, \ldots, 12\}\}$ and the maximum depth from $\{None, 1, 2, 3, 4, 5s \mid s \in \{1, \ldots, 12\}\}$. The only important parameter for the Ripper algorithm is the number of folds used during pruning, however changing this value has a negligible effect on the classification performance and is therefore set to 5. Finally, Classy avoids the need for hyperparameter tuning as shown in the original paper by Proença and van Leeuwen (2020). All of the remaining parameters that exist within the packages are set to their default values in this experiment. Furthermore, the computational time is denoted in seconds and both the average quarterly and compounded return of the portfolios in percent.

5.2 Rule Structure

As previously mentioned, the number of rules that a model produces and the average amount of predicates that are associated with these rules are two of the main drivers of interpretability. While the RF algorithm is known for its great empirical performance, it often does not perform well in these two areas and produces a large set of overlapping decision rules that is too complicated for investors to understand. This also becomes clear from Table 5.2, which shows its average number of rules and predicates across the different observation periods. The number of rules seems to increase with the length of the rolling window, however for small windows it already utilizes over 6000 rules to construct the decision boundaries. In addition, the rules can contain up to 8 or 9 predicates which adds another layer of complexity. Besides the clear lack of interpretation, the computational time for an individual quarter already takes roughly 7 minutes for windows with 700 to 900 training observations. Expanding the investable universe or grid-search can therefore have detrimental effects on the feasibility of the RF algorithm in terms of training time.

Table 5.2: Average number of rules, predicates and computational time of the RF algorithm

Window	**Rules**	**Predicates**	**Time (sec.)**
Small	6,231.68	9.23	219
Medium	9,981.28	8.86	305
Large	14,895.83	8.20	449

Figure 5.1 compares the average number of rules created by the interpretable learning algorithms with that of the RF model. Note that for the sake of visualization it depicts the logarithm as the other classifiers construct a significantly lower amount of rules. In general, their mutual ratio stays consistent when increasing the length of the rolling window. Note that for the tree-based models there is a sizeable increase in the number of rules for larger windows, while the Ripper and Classy algorithm experience an almost negligible effect. Furthermore, the tree-based models seem to find a considerably larger set of rules. Both the Classy and Ripper algorithm only yield a hand full of rules, yet they differ in terms of their rule mining abilities. Ripper is capable of consistently finding a small set of relations in each observation period, whilst the Classy algorithm seems to underperform in a more complex data environment. On average it only finds one additional rule in 7.8% of the observation periods for small windows, however this proportion increases for bigger windows with 15.6% and 56% for middle and large windows respectively. Hence, it becomes apparent that the Classy algorithm requires more training observations compared to the other classifiers in order to perform well within more complex datasets.

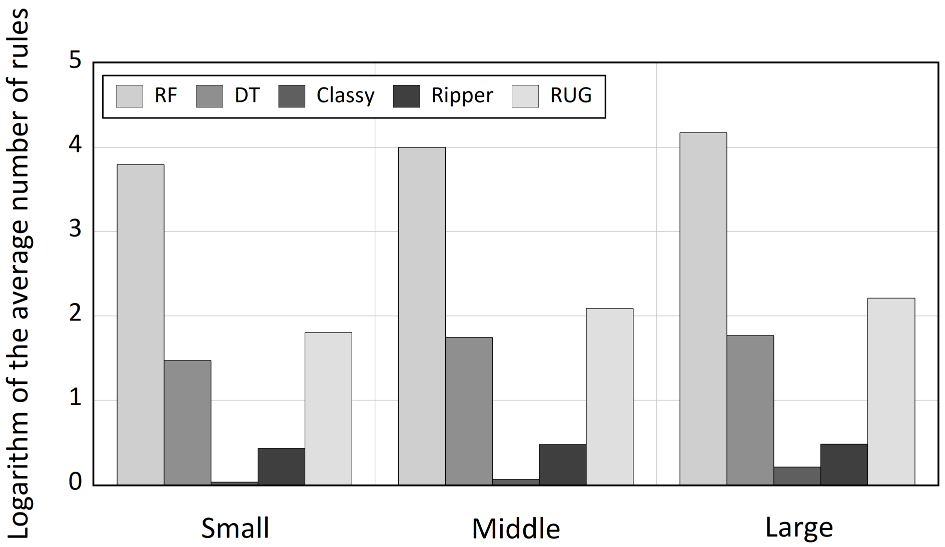

Figure 5.1: Average number of rules from 2000Q1 to 2020Q3

To get a better grasp of the difference in rule structure between the interpretable classifiers, an overview is given in Table 5.3 containing their average composition and computational time for the various rolling windows. By construction the RUG algorithm results in a larger set of rules than CART, nevertheless this increase coincides with a decrease in predicates. In particular, there is an average difference of around 2 predicates within an individual rule between the models. As a result, the RUG algorithm constructs more concise rules in this experiment which makes them easier to comprehend for investors. Nonetheless, both classifiers construct their decision boundaries on a significantly smaller set of rules compared to RF. The Ripper algorithm assembles the most transparent set of rules in this scenario, averaging around 3 decision rules per forecasting period. Moreover, most of its rules only consist of 1 or 2 predicates which translates to relatively simplistic decision boundaries. Whilst this is a beneficial characteristic from a clarity perspective, it could also have an adverse effect on the classification and portfolio performance.

Table 5.3: Average number of rules, predicates and computational time of the interpretable models

	CART			**Classy**			**Ripper**			**RUG**		
	S	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*
Rules	29.74	56.04	58.75	1.08	1.15	1.61	2.69	2.99	3.02	63.80	123.04	162.53
Predicates	6.71	6.98	5.81	2.33	3.71	5.43	1.46	1.67	1.83	4.77	5.26	4.36
Time (sec.)	20.28	43.44	58.46	4.47	4.93	6.61	0.04	0.14	0.24	1.24	8.04	24.41

In terms of practical feasibility for investors, all models terminate on average within 1 minute making their implementation almost effortless. Note that the RUG algorithm is initialized using the optimal hyperparameters obtained from the CART grid-search. Therefore, the denoted computational time of the RUG can be understood as the additional time it takes after tuning the maximum allowed nodes and minimum allowed observations parameters. In addition, the training time of the CART algorithm increases at a lower rate than the RF model for larger rolling windows. Hence, broadening the investable universe or increasing the number of unique parameters in the grid-search is less problematic for the CART and RUG approach.

5.3 Classification Performance

The classification performance of the interpretable machine learning models is evaluated in three stages. First their overall ability to differentiate between stocks with positive and negative future returns is assessed by means of the average *ACC* measure for the different rolling window groups. This is followed by a more detailed investigation of the forecasts, evaluating the classification of *long* and *short* positions from an individual perspective.

Table 5.4: Average accuracy of the classification models across the rolling windows

Model	Small	Middle	Large
RF	**0.59**	**0.57**	0.57
CART	0.49	0.50	0.51
Classy	0.50	0.49	0.49
Ripper	0.54	0.51	0.51
RUG	0.50	0.53	**0.60**

Note: the best performing classifier in terms of average accuracy for a specific group of rolling windows is indicated in bold.

Recall that the ACC indicates the proportion of stocks that is assigned a forecast position leading to a positive return on investment in the next quarter. Hence, it indicates to what extent a classification model can capture the underlying relations between the fundamental characteristics of a firm and its future stock return. Table 5.4 contains the average ACC of the different classifiers. For the interpretable models, its range lies between 0.49 and 0.60. Note that especially for the RUG and Classy algorithm this outcome is substantially lower than the presented results of their respective introductory papers. However, this is most likely explained by the high complexity of stock selection problems as the ACC of the RF model lies within the same range. Regarding the size of the rolling window, only the CART and RUG algorithm seem to separate well between market conditions of different periods as they achieve their highest ACC level for the large window group. The remaining models in this experiment perform better within small sample scenarios, where the training data mostly contains stock observations that are close to the testing period. For rolling windows ranging from 1 up to 6 years (e.g. small and medium windows), the RF model outperforms all interpretable classifiers based on ACC while for the larger windows it comes second after the RUG algorithm. In particular, RUG is the only interpretable machine learning model that is capable of beating the benchmark in this aspect and achieves the highest ACC obtained (0.60) across the different training scenarios for any model. Furthermore, while on average the Ripper algorithm constructs its decision boundaries based on 93.5% less rules than CART it never underperforms in terms of ACC. In fact, this more generalized decision approach outperforms all the other interpretable learning models for small window sizes and comes only second to RUG for the middle windows.

Table 5.5 gives an overview of the classification performance with respect to long position forecasts, in order to assess to what extent their classification contributes towards the ACC of the models. As the RCL denotes the average fraction of correctly classified long positions, it is clear that the RUG algorithm captures the most stocks that exhibit a positive future return. In particular, it consistently outperforms all other classifiers independent of the rolling window size. For

Table 5.5: Average classification performance of the learning algorithms in regard to long positions

	RF			**CART**			**Classy**			**Ripper**			**RUG**		
	S	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*
RCL	0.59	0.56	0.57	0.48	0.45	0.47	0.51	0.50	0.45	0.53	0.51	0.51	**0.64**	**0.66**	**0.68**
PPV	**0.64**	0.62	0.60	0.63	**0.63**	0.63	0.37	0.34	0.49	0.60	0.58	0.58	0.53	0.61	**0.76**
F_1*-score*	**0.61**	0.59	0.57	0.51	0.49	0.51	0.40	0.38	0.42	0.58	0.54	0.54	0.57	**0.63**	**0.71**

Note: the best performing model for each metric per group of rolling windows is denoted in bold.

short- and middle-sized windows this comes at the cost of precision as it respectively covers a smaller proportion than the RF and CART algorithm in terms of PPV. The rule sets from RUG in these training scenarios clearly favour long positions too excessively, which is a possible motive for the considerably lower ACC in shorter windows. Nevertheless, the F_1-score (0.71) indicates that the RUG algorithm for large rolling windows realizes the best trade-off between quantity and precision as it on average captures 68% of the long stocks correctly with only an average of 24% of its long positions making a negative return on investment.

Whilst the CART model is used to construct the initial rule set of RUG, the algorithm itself achieves the lowest RCL out of all classifiers and is even outperformed by the Classy algorithm. This is partly explained by a lower total of forecasted long positions as the PPV of CART indicates a competitive level of precision relative to the RUG and RF model. Still, the proportion of correctly classified long positions lies on average below 50% which might suggest that it is beneficial for investors to follow the market portfolio in this investable universe when only looking at long positions instead of the selection made by CART. In addition, note that by construction the Classy allocation closely follows the market allocation for rolling-windows up to 6 years as in these training scenarios it barely finds any decision rules additional to the default forecast. For larger windows it finds rules more often, yet on average it captures less of the total long positions. The increase in F_1-score shows that this does not mean that the model necessarily performs worse, since the decrease in RCL is offset by an increase in PPV resulting in more accurate predictions of long position stocks. Furthermore, the Ripper algorithm performs reasonably well considering that it only uses a hand full of rules. On average it captures 52% of the total long positions with around 59% of its long position forecasts actually having a positive return on investment.

In the overview from Table 5.6 it becomes clear that the RF model is far more versatile than the interpretable learning algorithms, as it maintains a similar performance level for short position forecasts. Whilst the RUG and Ripper algorithm seemed competitive in some areas for long position stocks, all interpretable classifiers display a substantial reduction in their forecasting precision (lower NPV) when it comes to short positions. In particular, the RF model outperforms the other

Table 5.6: Average classification performance of the learning algorithms in regard to short positions

	RF			**CART**			**Classy**			**Ripper**			**RUG**		
	S	M	L	S	M	L	S	M	L	S	M	L	S	M	L
SPC	**0.60**	**0.58**	**0.57**	0.51	0.54	0.52	0.49	0.51	0.55	0.53	0.51	0.51	0.32	0.33	0.38
NPV	**0.56**	**0.52**	**0.54**	0.36	0.37	0.36	0.20	0.20	0.29	0.46	0.41	0.41	0.41	0.38	0.30
F_1-*score*	**0.57**	**0.55**	**0.56**	0.39	0.40	0.39	0.27	0.27	0.34	0.49	0.43	0.43	0.36	0.36	0.33

Note: the best performing model for each metric is denoted in bold per group of rolling windows.

classifiers in all aspects independent of rolling window size. On average it captures 58% of the total short positions with more than half of these positions providing the investor a positive return. This consistent classification performance across both short and long positions is the reason why the RF algorithm outperforms the other models in terms of ACC for short- and middle-sized windows. Note that RUG is the only classifier that has a considerable decrease in correctly classified stocks, while the other models obtain a similar proportion for SPC compared to RCL. Nevertheless, all interpretable learning algorithms suffer from a substantial reduction in forecast precision and therefore obtain a far lower F_1-score for short stocks.

Note that this low performance for short position forecasts further emphasizes how well the RUG algorithm finds long investment opportunities for large rolling windows, as it still achieves a higher ACC than the RF model in this scenario. The big difference in its F_1-score between both classes implies that the decision boundaries from RUG are more tailored towards the prediction of long positions. In fact, it performs second to worst out of all models from a short perspective for small- and middle-sized windows and even attains a lower F_1-score than Classy for larger widows. Out of all the interpretable learning models, the Ripper algorithm finds the best trade-off between total captured short positions and the corresponding precision. Unfortunately, it only finds an average of 52% of the total short positions with roughly 57% of the short positions that it takes resulting in a negative return on investment. Recall that the investable universe, being the historical constituents of the S&P 500 index, denotes the set of assets in which we are allowed to invest. As previously explained, large positive returns are far more likely to occur within the investible universe of this research than large negative returns. This status quo in combination with a large proportion of the forecasted short positions exhibiting a positive future return on average could supposedly lead to a significant reduction in quarterly returns for equally weighted allocations. Since the performance of the RF model also is slightly lower for short positions, this might explain why the current literature of stock classification has only focused on constructing portfolios based on the predictions of positive future returns.

5.4 Portfolio Performance

Although the average F_1-score regarding long position forecasts only lies around 54% across the various machine learning models, this does not necessarily imply that the models perform poor in terms of portfolio return. The market allocation itself already has an average quarterly return of roughly 6.05%, resulting in a compounded return of 284.35% over a period of almost 12 years from 2009Q1 to 2020Q3. Hence, if the balance between RCL and PPV is more than sufficient, the correctly taken long positions might possibly compensate for the losses received by the FP observations and improve upon the market return. This development can also be seen in Table 5.7, which contains a summary of the portfolio allocations that are constructed by the classifiers with respect to long positions. All model-window combinations that obtained an F_1-score above 58% (see Table 5.5) provide a higher average quarterly return than the market and therefore result in a larger compounded return. Note that there are two exceptions to this rule, as the CART and RUG algorithm both deliver an average quarterly return slightly higher than the market allocation for small- and large-sized windows respectively while their corresponding F_1-score is below 58%. This could be a result of their more aggressive approach towards long positions when compared to the other classifiers. In particular, these two cases exhibit an increase of around 30% in the average number of involved stock positions with respect to the other model-based allocations.

Table 5.7: Average portfolio performance of an equally weighted allocation in terms of long positions

	RF			**CART**			**Classy**			**Ripper**			**RUG**		
	S	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*
Quarterly (%)	6.50	**6.25**	5.68	6.10	6.04	5.88	2.90	3.26	5.18	**6.56**	6.01	6.18	5.98	6.14	**7.18**
Compounded (%)	306	294	2.67	287	2.84	2.76	1.36	1.53	2.44	308	2.83	291	2.81	288	338
Positions	262	264	274	324	257	259	261	267	236	246	234	342	258	276	243

Note: average number of long positions taken per allocation is rounded to the closest integer; the highest average quarterly return per rolling window is denoted in bold; and the underlined values indicate a larger compounded return than the market allocation.

As expected from the classification results, the RUG algorithm is the best performing model for large rolling windows and provides an average quarterly return of 7.18% resulting in a compounded return of roughly 338%. In addition, all model-based allocations are substantially smaller than the S&P 500 index and on average consist out of 47% fewer stocks. For the small- and middle-sized windows the Ripper approach (6.56%) and RF model (6.25%) respectively achieve the highest average quarterly return. Therefore, the simplistic attitude of the decision boundaries created by Ripper continues to perform well for small rolling windows in terms of its practical application. Due to the lack of rules mined by the Classy algorithm, it is the only classifier that cannot obtain a higher quarterly return than the market independent of window size. Nevertheless, the big increase in returns for the larger windows is promising and shows signs of investment potential when more observations are included in the training process.

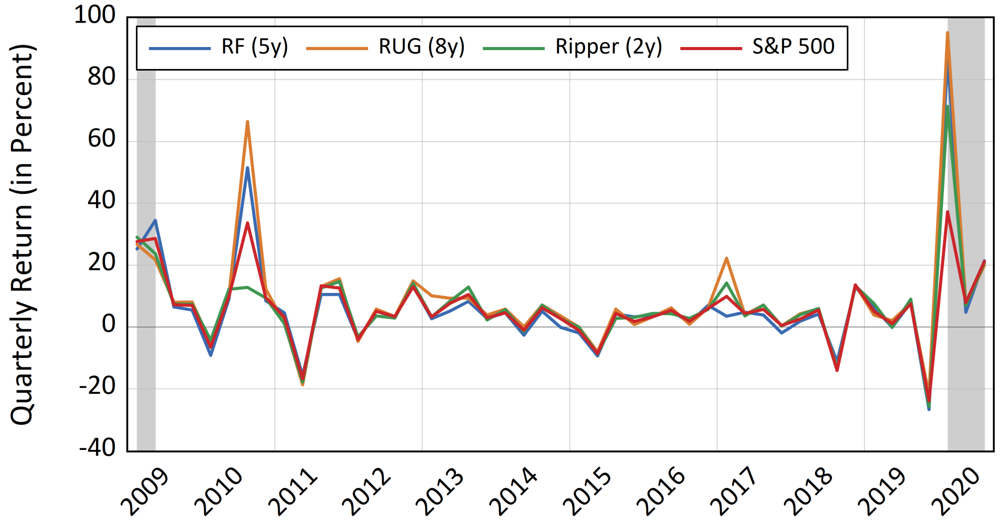

Figure 5.2: Average quarterly returns of the equally weighted long position allocations from the best classifier per rolling window group (shaded areas are NBER-defined recession periods)

Whilst some of the machine learning models outperform the market allocation, it does not need to result in a return series that is significantly different from the S&P 500. In particular, the mean equivalence test actually indicates that all of the models fall outside of the 10% significance level. To demonstrate this situation, Figure 5.2 depicts the average quarterly return series of the best classifier per group of rolling windows. Here, the RUG algorithm uses 8 years of training observations and has an average quarterly return around 8.61%, while the RF model (6 years) and Ripper (2 years) respectively have an average return of roughly 6.81% and 6.58%. Note that even with a higher average return of 2.56 percent points per quarter the RUG still closely follows the market portfolio. Most of the difference in returns clearly results from three specific periods, namely: the first half of 2010 and the second half of 2016 and 2019. Nevertheless, these intervals of high returns still hold some value for investors as the allocation of the RUG algorithm helps surpass the compounded return of the equally weighted market allocation by 66.67 percent points. In terms of consistency, all models experience a statistically significant difference in variance at the 5% confidence level except for the Classy algorithm. Unfortunately, as the classifiers closely follow the market allocation during most periods in terms of quarterly returns, it suggests that on average none of the model-based allocations are less nor more volatile than the market portfolio.

Table 5.8: Average portfolio performance of an equally weighted allocation including short positions

	RF			**CART**			**Classy**			**Ripper**			**RUG**		
Quarterly Return (%)	*S*	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*	*S*	*M*	*L*
No certainty threshold	**0.83**	0.11	**0.41**	0.31	-0.32	0.33	-0.85	-0.72	0.13	0.47	**0.73**	-0.51	-0.15	-0.20	-0.22
Certainty threshold	**9.79**	**7.81**	7.01	6.51	6.33	6.05	-0.90	-0.78	6.27	7.39	6.34	6.43	6.11	7.22	**9.01**
Optimal lower bound (%)	79	85	86	92	95	90	65	65	65	71	75	80	75	81	80

Note: the highest average quarterly return for both strategies per rolling window is denoted in bold and the optimal lower bound per group of rolling windows is rounded to the closest integer.

Next, Table 5.8 considers the addition of short positions to the model-based allocations. As expected, including short position forecasts at the same rate (meaning no certainty threshold) as the long investments from Table 5.7 drastically decreases the average quarterly return of the portfolios. Moreover, none of the model-based allocations is able to reach a 1% quarterly return and in 7 out of the 15 cases the portfolios even exhibit a negative return on investment. This further substantiates the notion that within the investible universe of this experiment, misclassifying long positions as short position stocks is more costly than going long in stocks that exhibit a positive future return.

To reduce the risk of the model-based portfolios in terms of short position forecasts, this research introduced a certainty threshold. Note that, as shown in Table 5.8, the optimal lower bound differs across the learning algorithms. The Classy algorithm requires the lowest certainty threshold as it assigns a consistent lower bound of 65% across all quarters independent of window size. However, this is mostly due to the lack of diversification power between safe and risky stocks. In particular, it assigns 100% probability to the default class if the algorithm was not able to find more rules and consistently assigns a probability of 66% to the most likely class if there are multiple rules within its rule list. The remaining models find their best certainty threshold roughly in the range of 70% to 85%, while CART has it slightly higher between 90% and 95%.

Note that apart from the Classy algorithm for small- and middle-sized windows, all model-based allocations now provide a higher average quarterly return compared to their long position alternative in Table 5.7. Overall, the RF model now achieves the highest possible earning on investment with an average quarterly return of 9.79% for small rolling windows. This results in an average increase of around 175.6 percent points in compounded return with respect to the equally weighted market portfolio. This dominant performance of the RF algorithm for both small- and middle-sized windows is likely due to its consistent classification performance across short and long positions. As a consequence, it diversifies better between the risky short position stocks in contrast to the interpretable learning models and therefore shows signs of a larger increase in returns. However, the considerable decrease in portfolio returns remains when the size of the rolling window is increased. This suggests that the model still struggles to differentiate between the market conditions of previous and recent periods. Hence, the RUG algorithm still outperforms all remaining classifiers for rolling windows from 7 to 9 years with an average quarterly return of 9.01%. Note that on average the Ripper algorithm never provides a lower return than CART after including a certainty threshold, while it only implements roughly 3 rules to construct its decision boundaries.

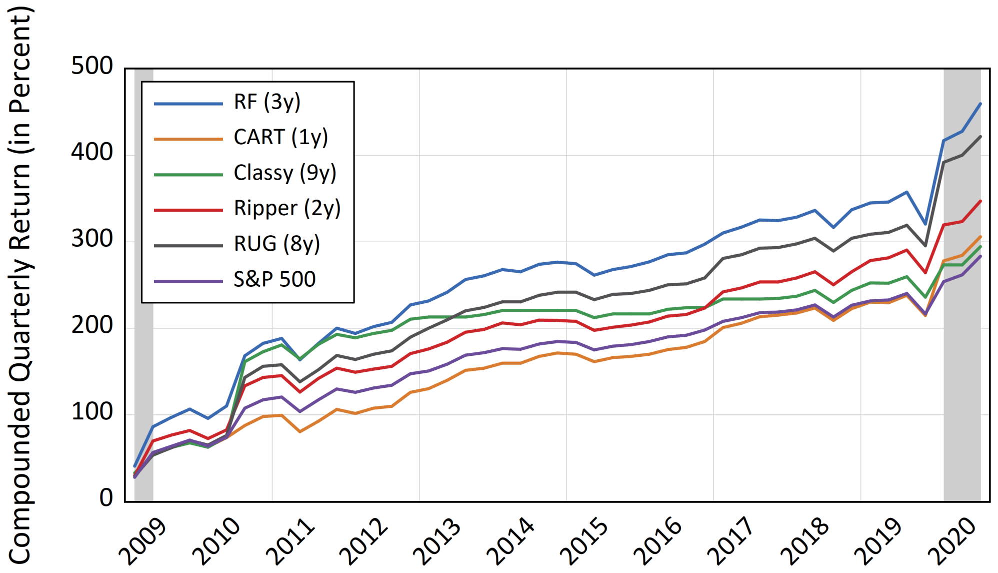

Figure 5.3: Compounded returns of the equally weighted allocations with certainty threshold for the best rolling window per classifier (shaded areas are NBER-defined recession periods)

Whilst the majority of the model-based allocations display an increase in portfolio returns, they still do not significantly differ from the S&P 500 index based on the equality test. Figure 5.3 depicts the compounded return of the market portfolio and the best performing allocation of each machine learning algorithm. It is evident that each model-based portfolio follows the market allocation to some extent, yet during several short periods exhibit a slight positive difference in returns resulting in a higher return on investment in the long-run. In particular, all classifiers provide a higher compounded return for their optimal training window than the historical constituents of the S&P 500 index over a period stretching from 2009Q1 till 2020Q3. After the inclusion of short positions, none of the interpretable classifiers outperform the RF model. However, the RUG algorithm comes reasonably close with an average difference in quarterly returns of 0.78%.

6 Conclusion

The current financial literature is overflowed with research that compares the predictive performance of models that are not capable of providing investors beneficial insights on the behaviour of stocks. In addition, there is large gap regarding rule-based learning algorithms, as the performance of new innovative models has not been analysed in a variety of real-world domains. To broaden the financial literature on the topic of interpretable machine learning, this research analysed the performance of various interpretable learning algorithms within an equity trading context. The dataset consisted of quarterly stock observations from historical S&P 500 constituents, which are available at the Wharton Research Data Services for a time period stretching from 2000 to 2020. To assess the feasibility of the classifiers, their execution was analysed in terms of interpretability, classification, and portfolio performance. Furthermore, this study explored the practical implications of a lower bound on the forecasting probability of short position stocks when rebalancing the portfolio.

Considering the transparency of the decision-making process, all classifiers construct their decision boundaries using substantially less rules than the RF algorithm independent of rolling window size. Their rules are also more concise as they contain less predicates on average, making them easier to interpret for investors. The tree-based algorithms show a sizeable increase in the number of rules for larger rolling windows, while the remaining models exhibit a negligible effect. In general, the Ripper algorithm provides the most understandable rule set and the Classy algorithm struggles to find enough sufficient rules due to a possible lack of training observations.

Both Ripper and RUG perform reasonably well regarding the classification of long position forecasts. It is found that for small- and large-sized rolling windows they respectively match the classification performance of RF. This also translates over to their portfolio returns, as they provide the highest average quarterly return out of all models for these respective window groups when only including long positions. In particular, the RUG algorithm is the only classifier that can beat both the machine learning benchmark and the market allocation in this scenario.

However, incorporating short positions without any restrictions drastically decreases the average quarterly return of all model-based portfolios. The reason for this is two-fold. Firstly, all interpretable models exhibit a sharp decrease in classification performance for short position forecasts. In addition, misclassifying long positions as short position stocks is more costly within the investible universe of this experiment. Implementing a certainty threshold for short position forecasts appears to be a successful solution to both problems. As a result, all model-based allocations obtain a higher compounded return than the market portfolio after incorporating the threshold. However, none of the interpretable classifiers offers this return while concurrently being less volatile than the market.

This is largely explained by the fact that the return series of the model-based allocations closely follow the return of the market portfolio in most quarters and only exhibit a limited amount of periods in which they significantly outperform the market.

Nevertheless, the stock selection procedure of the interpretable learning algorithms still holds value for some financial investors. Those that closely follow the S&P 500 index can especially benefit in the long-run, as the average quarterly return does not significantly differ, yet 47% less positions are required to form the model-based portfolios. Furthermore, classifiers such as the Ripper and RUG algorithm outperform the market by a substantial margin in several quarters while never performing significantly worse in others. As a result, one could have received up to 139 percent points more in terms of compounded return from 2009Q1 till 2020Q3 by following an equally weighted selection of RUG instead of the S&P 500. In addition, the decision boundaries might still give valuable insights on the general behaviour of stocks which could help improve more sophisticated investment strategies.

For future research it might be interesting to deviate from the fundamental data attitude of this experiment and evaluate the model allocations based on well-known factors from the field of portfolio management. Including more developed performance measures as features could potentially lead to a return series that behaves significantly different from the market. Recall that the Classy algorithm showed small signs of potential for the larger windows. Hence, increasing the data-frequency could possibly lead to significant improvements in terms of its rule-learning capability without adding noise from more period dependent market conditions. Besides an alternative data approach, different variations of the interpretable classifiers can be evaluated as well. For instance, future research could investigate the portfolio allocations of RUG while employing other rule-based models to initialize and optimize its rule set. Note that such classifiers require the application of class weights within the rule-learning process. Lastly, more refined feature selection procedures such as the Genetic algorithm might reduce noise more sufficiently than the AdaBoost framework.

References

Akyüz, M. H. and İlker Birbil, (2021). Discovering classification rules for interpretable learning with linear programming.

Batres-Estrada, B. (2015). Deep learning for multivariate financial time series. Master thesis, KTH Royal Institute of Technology.

Birbil, S. I., Edali, M., and Yuceoglu, B. (2020). Rule covering for interpretation and boosting.

Breiman, L. (1996). Bagging predictors. *Mach. Learn.*, 24:123–140.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA.

Chen, L., Pelger, M., and Zhu, J. (2020). Deep learning in asset pricing. Working paper, Stanford University.

Chia-Cheng, C., Chun-Hung, C., and Ting-Yin, L. (2020). Investment performance of machine: Analysis of S&P 500 index. *International Journal of Economics and Financial Issues*, 10(1):59–66.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, ICML'95, page 115–123, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

DeMiguel, V., Garlappi, L., and Uppal, R. (2009). Optimal versus naive diversification: How inefficient is the 1/n portfolio strategy? *The review of Financial studies*, 22(5):1915–1953.

Dougherty, G. (2012). *Pattern Recognition and Classification: An Introduction.* SpringerLink : Bücher. Springer New York.

Fu, X., Du, J., Guo, Y., Liu, M., Dong, T., and Duan, X. (2018). A machine learning framework for stock selection.

García, S., Luengo, J., and Herrera, F. (2014). *Data Preprocessing in Data Mining.* Intelligent Systems Reference Library. Springer International Publishing.

Grünwald, P. D., Grunwald, A., and Rissanen, J. (2007). *The Minimum Description Length Principle.* Adaptive computation and machine learning. MIT Press.

Gu, S., Kelly, B., and Xiu, D. (2018). Empirical asset pricing via machine learning. Working Paper 25398, National Bureau of Economic Research.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3(null):1157–1182.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer series in statistics. Springer.

Heij, C., de Boer, P., Franses, P. H., Kloek, T., and van Dijk, H. (2004). *Econometric Methods with Applications in Business and Economics.* Oxford University Press.

Huerta, R., Corbacho, F., and Elkan, C. (2013). Nonlinear support vector machines can systematically identify stocks with high and low future returns. *Algorithmic Finance*, 2(1):45–58.

Jegadeesh, N. and Titman, S. (2012). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, 48(1):65–91.

Lakkaraju, H., Bach, S. H., and Leskovec, J. (2016). Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1675–1684, New York, NY, USA. Association for Computing Machinery.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc.

Malkiel, B. G. and Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):383–417.

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, 7(1):77–91.

Plyakha, Y., Uppal, R., and Vilkov, G. (2012). Why does an equal-weighted portfolio outperform value- and price-weighted portfolios? Working paper, Social Science Research Network.

Proença, H. M. and van Leeuwen, M. (2020). Interpretable multiclass classification by mdl-based rule lists. *Information Sciences*, 512:1372–1393.

Rasekhschaffe, K. C. and Jones, R. C. (2019). Machine learning for stock selection. *Financial Analysts Journal*, 75(3):70–88.

Rissanen, J. (1983). A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431.

Shapley, L. S. (1953). A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317.

Sharpe, W. F. (1970). *Portfolio theory and capital markets.* McGraw-Hill, New York.

Shen, P. (2003). Market timing strategies that worked. *The Journal of Portfolio Management*, 29(2):57–68.

Shtarkov, Y. M. (1987). Universal sequential coding of single messages. *Problems of Information Transmission*, 23(3):3–17.

Sloan, R. (1996). Do stock prices fully reflect information in accruals and cash flows about future earnings? *The Accounting Review*, 71(3):289–315.

Snedecor, G. W. and Cochran, W. G. (1989). *Statistical Methods.* Iowa State University Press, 8th edition.

van der Hart, J., Slagter, E., and van Dijk, D. (2003). Stock selection strategies in emerging markets. *Journal of Empirical Finance*, 10(1):105–132. Emerging Markets S.I.

Wang, S. and Luo, Y. (2012). Signal processing: The rise of the machines. Technical report, Deutsche Bank Quantitative Strategy.

Wang, S. and Luo, Y. (2014). Signal processing: Seven sins of quantitative investing. Technical report, Deutsche Bank Quantitative Strategy.

Witten, I. H., Hall, M. A., and Frank, E. (2016). *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, San Francisco, 4th edition.

Zhu, J., Zou, H., Rosset, S., and Hastie, T. (2009). Multi-class adaboost.

A Data Glossary

Table A.1: Glossary of the dataset

Variable Identifier	**Description**
FRETURN	Future Quarterly Return (in Percent)
FQTR	Fiscal Quarter (divided into 4 separate Dummy Variables)
TIC	Ticker Symbol
AJEXQ	Adjustment Factor (Company) - Cumulative by Ex-Date
AJPQ	Adjustment Factor (Company) - Cumulative by Pay-Date
ACOQ	Current Assets - Other - Total
AOQ	Assets - Other - Total
APQ	Account Payable/Creditors - Trade
ATQ	Assets - Total
CAPSQ	Capital Surplus/Share Premium Reserve
CEQQ	Common/Ordinary Equity - Total
CHEQ	Cash and Short-Term Investments
COGSQ	Cost of Goods Sold
CSH12Q	Common Shares Used to Calculate Earnings Per Share - 12 Months Moving
CSHFD12	Common Shares Used to Calc Earnings Per Share - Fully Diluted - 12 Months Moving
CSHFDQ	Com Shares for Diluted EPS
CSHOQ	Common Shares Outstanding
CSHPRQ	Common Shares Used to Calculate Earnings Per Share - Basic
CSTKQ	Common/Ordinary Stock (Capital)
DILADQ	Dilution Adjustment
DILAVQ	Dilution Available - Excluding Extraordinary Items
DLTTQ	Long-Term Debt - Total
DOQ	Discontinued Operations
DVPQ	Dividends - Preferred/Preference
EPSF12	Earnings Per Share (Diluted) - Excluding Extraordinary Items - 12 Months Moving
EPSFI12	Earnings Per Share (Diluted) - Including Extraordinary Items
EPSFIQ	Earnings Per Share (Diluted) - Including Extraordinary Items
EPSFXQ	Earnings Per Share (Diluted) - Excluding Extraordinary items
EPSPI12	Earnings Per Share (Basic) - Including Extraordinary Items - 12 Months Moving
EPSPIQ	Earnings Per Share (Basic) - Including Extraordinary Items
ESOPCTQ	Common ESOP Obligation - Total
IBADJQ	Income Before Extraordinary Items - Adjusted for Common Stock Equivalents
IBCOMQ	Income Before Extraordinary Items - Available for Common
IBQ	Income Before Extraordinary Items

Table A.2: Glossary of the dataset (continued)

Variable Identifier	**Description**
ICAPTQ	Invested Capital - Total - Quarterly
INVTQ	Inventories - Total
LCOQ	Current Liabilities - Other - Total
LOQ	Liabilities - Other
LSEQ	Liabilities and Stockholders Equity - Total
LTMIBQ	Liabilities - Total and Non-controlling Interest
LTQ	Liabilities - Total
NIQ	Net Income (Loss)
NOPIQ	Non-Operating Income (Expense) - Total
OEPF12	Earnings Per Share - Diluted - from Operations - 12MM
OEPS12	Earnings Per Share from Operations - 12 Months Moving
OEPSXQ	Earnings Per Share - Diluted - from Operations
OIADPQ	Operating Income After Depreciation - Quarterly
OPEPSQ	Earnings Per Share from Operations
PIQ	Pretax Income
XIDOQ	Extraordinary Items and Discontinued Operations
PSTKNQ	Preferred/Preference Stock - Nonredeemable
PSTKQ	Preferred/Preference Stock (Capital) - Total
PSTKRQ	Preferred/Preference Stock - Redeemable
RECTQ	Receivables - Total
REQ	Retained Earnings
REVTQ	Revenue - Total
SALEQ	Sales/Turnover (Net)
SEQQ	Stockholders Equity - Parent - Index Fundamental - Quarterly
SPIQ	Special Items
TSTKQ	Treasury Stock - Total (All Capital)
TXTQ	Income Taxes - Total
XOPRQ	Operating Expense- Total
XIQ	Extraordinary Items
XOPRQ	Operating Expense- Total
ACCHGY	Accounting Changes - Cumulative Effect
CHECHY	Cash and Cash Equivalents - Increase (Decrease)
COGSY	Cost of Goods Sold
CSHFDY	Com Shares for Diluted EPS
CSHPRY	Common Shares Used to Calculate Earnings Per Share - Basic
DILADY	Dilution Adjustment

Table A.3: Glossary of the dataset (continued)

Variable Identifier	**Description**
DILAVY	Dilution Available - Excluding Extraordinary Items
DVPY	Dividends - Preferred/Preference
DVY	Cash Dividends
EPSFIY	Earnings Per Share (Diluted) - Including Extraordinary Items
EPSFXY	Earnings Per Share (Diluted) - Excluding Extraordinary items
EPSPIY	Earnings Per Share (Basic) - Including Extraordinary Items
EPSPXY	Earnings Per Share (Basic) - Excluding Extraordinary Items
EXREY	Exchange Rate Effect
FIAOY	Financing Activities - Other
FINCFY	Financing Activities - Net Cash Flow
IBADJY	Income Before Extraordinary Items - Adjusted for Common Stock Equivalents
IBCOMY	Income Before Extraordinary Items - Available for Common
IBY	Income Before Extraordinary Items
IVACOY	Investing Activities - Other
IVNCFY	Investing Activities - Net Cash Flow
NIY	Net Income (Loss)
NOPIY	Non-Operating Income (Expense) - Total
OANCFY	Operating Activities - Net Cash Flow
OEPSXY	Earnings Per Share - Diluted - from Operations
OIADPY	Operating Income After Depreciation - Year-to-Date
OPEPSY	Earnings Per Share from Operations
PIY	Pretax Income
REVTY	Revenue - Total
SALEY	Sales/Turnover (Net)
SPIY	Special Items
TXTY	Income Taxes - Total
XIDOY	Extraordinary Items and Discontinued Operations
XOPRY	Operating Expense- Total
COSTAT	Active/Inactive Status Marker (divided into two seperate Dummy Variables)
CSHTRQ	Common Shares Traded - Quarter
DVPSPQ	Dividends per Share - Pay Date - Quarter
DVPSXQ	Div per Share - Exdate - Quarter
PRCCQ	Price Close - Quarter
PRCHQ	Price High - Quarter
PRCLQ	Price Low - Quarter
ADJEX	Cumulative Adjustment Factor by Ex-Date

B Python Scripts

IndexScraper.py - This module retrieves the indices of the historical S&P 500 constituents by means of web scraping using the BeautifulSoup environment. Needed as the subscription of the Erasmus University at the Wharton Research Data Services does not have access to the historical indices. The script consists of the following functions:

1. *scraper(path)*: Retrieves information and constructs two Dataframes containing information on the current constituents and all the historical changes made over time.
   - *path* = url of the web page containing the information of interest.
2. *timespan(first, last)*: Constructs a list with all observation dates on monthly basis.
   - $first$ = first observation date within period of interest.
   - *last* = last observation date within period of interest.
3. *write(sheet, row, date, indices)*: Saves the given S&P constituents to a given excel sheet.
   - *sheet* = worksheet in which the indices are saved.
   - *row* = specific row to which the indices are saved.
   - *date* = date corresponding to a specific group of the historical S&P 500 constituent.
   - *indices* = list of ticker values representing the S&P 500 constituents.
4. *snapshot(index, changes)*: Constructs a monthly snapshot of the S&P 500 index constituents.
   - *index* = dataframe containing the current S&P 500 constituents.
   - *changes* = dataframe containing the historical changes made to the S&P 500 index with corresponding date.

DataPreperation.py - This module combines the scraped historical S&P 500 constituents with the stock data retrieved from wharton and pre-processes the data accordingly, constructs the target variable and saves the training (based on rolling window) and test sets in separate files. The script consists of the following functions:

1. *quarterSPX()*: Constructs quarterly observations of the stock data based on the scraped historical constituents of the S&P 500 index.
2. *futureReturn(obs)*: Calculates feature quarterly return of a certain firm.
   - *obs* = all stock observations of a single firm.

3. *targetStock(indic1, indic1)*: Constructs a class variable using a 5-percent quantile selection using future quarterly return.

   – $indic1$ = list of remaining indicators after manual pre-processing steps (regarding missing values and outliers).

   – $indic2$ = list of indicators that need to be standardized.

FeatureSelection.py - This module performs feature selection for all rolling windows in each prediction period. The feature selection is performed using the training data. Finally, a new version of the training and test set is saved in separate files. The script consists of the following functions:

1. *main(wind per)*: Performs feature selection for given rolling window size and prediction period.

   – $wind$ = size of the rolling window.

   – per = prediction period of interest.

2. *dataSelector(wind, per, path)*: Given the size of the rolling window and period to be predicated, retrieves the pre-processed training and test set of interest.

   – $wind$ = size of the rolling window.

   – per = prediction period of interest.

   – $path$ = location of folder with pre-processed stock data files.

3. *importanceCalculator(data)*: Costructs a ranking of the features based on their mean decrease in impurity estimated from stratified 5-fold cross-validation.

   – $data$ = set of training observations.

4. *featureSelector(train, test, mdi)*: Performs specific-to-general approach to reduce the dimensions of the original training and test set based on the feature importance ranking.

   – $train$ = set of training observations.

   – $test$ = set of test observations.

   – mdi = feature importance ranking.

StockClassification.py - This module performs the actual stock classification of the experiment. Besides some additional functions that for structuring the experiment and functions that evaluate the results, this script consists of the following main function:

1. *main(windowLength, testPeriod, model)*: First retrieves the training and test set of interest. Next, optimizes the assigned classification model using its corresponding prediction function and makes the final prediction. This is followed by obtaining all evaluation metrics of interest and saving the results in csv worksheet.

   - $windowLength$ = size of the rolling window used during training.

   - $testPeriod$ = prediction period of interest.

   - $model$ = classification model of interest.