Course Project

Predicting Stock Returns with Machine Learning

Candidate numbers:

50

70

56

27

72

"Big Data with Applications to Finance "

FIE453

NORWEGIAN SCHOOL OF ECONOMICS

9.11.2018

Contents

List of Tables

List of Figures

Introduction

Stock market predictions has attracted a lot of attention from both the business and scholarly community. For instance, Warren Buffet's ability to consistently beat the S&P index has been viewed as a practical indicator that the stock market can be predicted. Early scholarly works on stock market predictions have focused on the Efficient Market Hypothesis (EMH) and the random walk theory. The EMH implies that stock picking will not generate more than the market return over time since financial markets are informationally efficient, and investors are assumed to be rational in their valuation of stocks. These early scholarly studies conclude that stock prices cannot be predicted on the basis on present or past prices because they are driven by new information rather than present/past prices. For instance, early tests showed strong support for the EMH and differences in return always seemed to be explained by differences in risk and nobody seemed to beat the market. Furthermore, the market seemed not to react to irrelevant news. Therefore, stock market prices follow a random walk and predictions cannot exceed the accuracy of 50%. As suggested by Burton Malkiel, 'a blindfolded monkey throwing darts at a newspaper's financial pages could select a portfolio that would do just as well as one carefully selected by experts' (Ferri, 2012).

Later studies, however, question the underlying assumptions of the Efficient Market Hypothesis and argue that the stock market can be predicted to some degree. For instance, human decision makers are not rational as argued by EMH and their mistakes and biases are not random noise that cancel out. In these later studies, patterns inconsistent with the EMH hypothesis began to emerge such as value stock and growth stocks, momentum investing, and 'no news' leading to major price changes, indicating excess volatility in stock markets. For instance, smaller company stocks and value stocks have been empirical found to outperform the market over a period. Thus, by giving a monkey sufficiently darts, they will beat the market, contradicting the EMH theory (Ferri, 2012).

In this report, we aim to construct models based on different machine learning techniques to predict whether or not a stock beats the market. The following section of this report will provide detailed description of our problem description.

Problem Description

As mentioned earlier, this report aims to predict which stocks beat the market. The research question we intend to answer is the following: *Can we predict whether or not a stock beats the market on monthly basis using different machine learning techniques*?
We have incorporated in total of 6 different machine learning techniques and constructed models based on them. Each of these techniques and built models will be outlined in detail in below sections. We intend to find the most accurate model to our research question by comparing these models against each other.

Since our research question is focused on which stocks beat the market and which do not, we have constructed a binary variable of 1 if a stock beats the market and 0 otherwise. Thus, all of the machine learning models that we have built can be used for classification problems. The way in which the binary variable is calculated will be discussed below.

In terms of evaluating whether applying machine learning techniques are worthwhile with respect to time, computing power and final results, we have decided to develop a benchmark. We used a simple pure guessing approach. We assume that what happens today will also happen tomorrow. Thus, if a stock was beating the market this month, we assumed that it would beat the market next month and vice versa. We did this for one month to the next for the whole test set and compared the "predicted" values to the actual values. We could have potentially used coin flip as our benchmark. However, since the 0s and 1s in our data set is not normally distributed, we concluded that a coin flip would not result in 50% accuracy in the given data set.

Based on this guessing method, we got the following accuracy for our benchmark:

Table 1. Accuracy of Benchmarking.

	Benchmark Accuracy
Sensitivity	44.6%
Specificity	54%
Pos Pred Value	44.8%
Neg Pred Value	53.8%
Total Accuracy	49.3%

Benchmark Confusion Matrix		
	0	**1**
0	10 234	8 785
1	8 721	7 071

The sensitivity depicts the percentage of accurately predicted positive excess return, while the specificity depicts the percentage of accurately predicted negative excess return.

Hence, our goal here is to find a model that predicts whether or not a stock beats the market more accurately than our pure guessing benchmark.

Data

This section of the paper will outline how we pre-processed and cleaned the data. A brief description of CRSP and Compustat data will be provided for the purpose of clarity. Furthermore, additional data that we have included in our project will be outlined, along with justifications for incorporating this data.

1.1. CRSP and Compustat Dataset

The CRSP dataset is a comprehensive collection of security, price, return and volume data and contains end-of-month data for the NYSE, AMEx and NASDAQ stock markets.

The Compustat dataset we are working with consists of fundamental financial and market information. The dataset compiles raw financial reports, filed by every public company and includes current income, expenses, assets and liabilities, capital expenditure and other company data (WRDS, 2018).

1.2. Additional Data

In addition to the data provided, we have considered additional data that could be useful when predicting whether a stock would beat the market. Since there is a reciprocal relationship between the development of stock market and changes in a country's economy, we consider it important to include relevant macroeconomic data. When choosing macroeconomic data, our goal was to find indicators that would affect all companies in our data set to a certain degree. We acknowledge that industries have various factors that drive price fluctuations, i.e. different macroeconomic factors. However, we intended not to find industry specific macroeconomic factors and, thus, we have chosen more general macroeconomic indicators. In addition to macroeconomic data, we have included volatility indicator to our data set. All additional data collected in this project is U.S. specific for the time period of 2010 to 2017. In order to ensure the validity of data, we have gathered the data from official databases such as the Bureau of

Labour Statistics (BLS) and OECD. Our final selection of additional data includes the following variables:

Gross Domestic Product

Gross Domestic Product (GDP) is the monetary value of all the finished goods and services produced within a country's border within a specific time period and includes, among other things, all private and public consumption, government outlays and the foreign balance of trade. The GDP is commonly used as an indicator of a country's standard of living and growth in GDP indicates economic growth in a country. Intuitively, economic growth should suggest faster sales growth for companies because people have more money, resulting in a higher demand for stocks and thereby potentially lifting the stock market (Kleintop, 2018). However, Kleintop (2018) argues that stocks do not track economic growth closely. Kleintop, thus, concludes that GDP is not a good proxy for the stock market performance of a country. Nevertheless, GDP is a relatively important macroeconomic factor in terms of stock markets because stocks rarely rise with falling GDP when an economy is in a recession. Thus, we have included the quarterly GDP and printed the same value for each month in a specific quarter. This means that the same value would appear for all months in the first quarter (January to March). The data used in this project was extracted from the OECD (2018) database.

Unemployment rate

Unemployment rate illustrates the current state of an economy and is important for a country's further development. According to research, unemployment rate and stock market are highly correlated (Miao, Wang and Xu, 2016). The intuitive rationale for the relationship between unemployment and stock market is that the more people there are with jobs, the higher the economic output, retail sales and corporate profits and, as a result, stocks should rise with good employment news (fall in unemployment rate). However, the relationship between unemployment rate and stock market is dependent on the state of the economy (Boyd, Hu and Jagannathan, 2005). For instance, if the economy is in expansion, raising unemployment rate has a positive effect on stock prices due to expectations that interest rates will fall (Boyd, Hu and Jagannathan, 2005). Stock prices, however, fall with rising unemployment rate when the economy is in contraction (Boyd, Hu and Jagannathan, 2005). We have incorporated monthly unemployment rates, gathered from the BLS (2018a) database.

Consumer Price Index

The Consumer Price Index (CPI) is a measure of the average change over time in the prices paid by urban consumers for a market basket of consumer goods and services (BLS, 2018b). Since CPI shows the change in prices, it is an indicator of inflation. The rationale for including CPI in our data set is that CPI has an impact on stock prices through interest rates. Deviation in the target inflation rate (CPI) can lead to an ease on fiscal stimulus via raising interest rates (Shen, 2018). Here we refer to inflation targeting. Thus, a rising inflation rate results in increasing costs which will hurt profitability of companies and depress stock values. We collected the CPI monthly data from the BLS (2018b) database. The monthly data from BLS indicates the percentage change from the previous month in the average prices of the market basket.

Volatility Index

Volatility index (VIX) is a market index that represents the market's expectation of price volatility over the next 30 days (Financial Times, 2018). It is a weighted average of the prices of various S&P 500 index options contracts and is released by Chicago Board Options Exchange. It is often called as the "fear index" because it is a measure of risk, fear and stress in the stock market. This means that VIX rises when investors are concerned about future volatility (Financial Times, 2018). VIX is an important variable since it represents how much risk is associated with the market. Since there is a negative correlation between volatility and stock market returns, the VIX value moves up when the market is falling. We collected the data for monthly VIX from Yahoo Finance (2018).

1.3. Data Cleaning

In order to run the models, we merged the different datasets into one. When merging the datasets, we had to account for the fact that the CRSP dataset consists of monthly data, while the Compustat dataset consists of annual reports. It was important for us that we were able to keep as much information as possible, while still merging the files in a way that is not introducing future information into the dataset. This was done to avoid forward-looking bias. We merged each observation in the CRSP dataset with the Compustat observation that was between 4 and 15 months before, due to the fact that the companies in the Compustat file report at different times. If we were to simply lag the CRSP file one year and then merge the files by PERMNO and year, we would in some cases match data that were almost two years apart.

After the files were matched with the correct dates, we cleaned the data. Some of the columns in the combined dataset contained a lot of missing values. Therefore, if we simply removed all the rows containing missing values, we would be left with 0 observations. To avoid this, we removed the columns and rows that contained more missing values than a given percentage. To determine the percentage, we took the median of the sum of all NAs in the dataset, divided by the number of rows in the dataset. After doing so, we went from 168 columns and 426 480 observations to 84 columns and 379 953 observations. Next, we removed all the rows containing missing values and were left with a dataset containing 288 953 observations.

In addition to removing the missing values, we standardized the dataset in order to account for the difference in values between the companies in the dataset. By standardizing the dataset, each feature contributes approximately proportional to the final distance, which makes it possible to compare the companies to each other, despite the difference in size. The standardizing of the dataset was done by subtracting the column mean from each value and then dividing the value by the standard deviation of the variable. By doing so, all the variables in the dataset got a mean equal to zero and a standard deviation of one.

We also added a binary variable to the dataset, which was 1 if the company's return was higher than the market return, for each company and each month in the dataset. It was set to zero if otherwise. This was done in order to answer our problem statement.

Lastly, we removed all the columns that were used to calculate the binary variable, e.g. price, return, market return, in addition to the variables that contained dates and company number. This left us with a dataset containing 72 variables and 288 953 observations.

1.4. Walk-Forward Testing

Since the dataset contains company- and stock time-series data and our goal was to include as much information as possible, it was important to keep the time dimension of the observations. As a result, we could not randomly split the data into groups, but rather split the data with the time-series aspect kept in mind. One way of splitting the values that also takes the time dimension of the observations into account is the walk-forward testing. By performing several optimization/test steps over time, the most recent observations will have more weight than the observations in the distant past, which can lead to the parameter values in the models being

more suited to the marked conditions that immediately follow (Amibroker, 2018). The walk-forward testing is exemplified in Table 2.

Table 2. Walk-forward testing

2010	2011	2012	2013	2014	2015		
	2011	2012	2013	2014	2015	2016	
		2012	2013	2014	2015	2016	2017

We tried to use the walk-forward testing when we ran the models, however, it required a lot of computing power and would therefore be too time consuming. To solve this, we decided to use a three-way split.

1.5. Splitting Data

We used a three-way split where the training set was the "in-sample" set that we trained our models on, the validation set was the sample we used to choose how much to penalize, and the test set was the "out-of-sample" set that we used to measure the fit.

In order to use the three-way-split while still accounting for the time dimension of the observations, we chose to use the first five years in the dataset, 2011-2015, as our train-set, the observations from 2016 as our validation set and the observations from 2017 as our test set. The train-, validation- and test set with the number of observations is visualized in Table 3.

Table 3. Splitting Data.

2010	2011	2012	2013	2014	2015	2016	2017
Train						Validation	Test
207 183 observations						43 141	38 629

However, for the deep learning model, we split the dataset into a train and test set, where the observations from 2011 - 2016 were used to train the model, and the observations from 2017 were used as a test set. This was done because the deep learning model automatically splits the train set into train and validation.

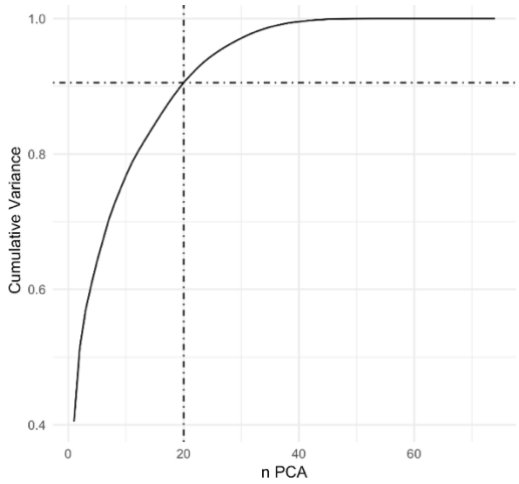
1.6. Principal Component Analysis

Since the mentioned dataset consists of more than 280 000 observations and 72 variables in total, running all the models requires a lot of computing power. To solve this, we decided to use Principal Component Analysis (PCA) in order to reduce the necessary computing power, while still being able to explain a lot of the variance in the dataset. PCA is a method of extracting important variables from a large set of variables available in a dataset. It extracts low dimensional set of features, from a high dimensional dataset, in form of components.

PCA reduces the dimensions. Since PCA is performed on a symmetric correlation or covariance matrix, we made the entire dataset numeric and standardized the data, as described in the data cleaning. In short, a principal component is a normalized linear combination of the original predictors in a dataset (Analytics Vidhya, 2016).

As Figure 1 shows, by using PCA on the train dataset, we were able to describe more than 90% of the variance, using 20 PCA components. This in turn drastically reduced the runtime for each model and in some cases, improved the prediction, compared to when we ran the models on the whole dataset. A possible explanation for the increased accuracy of some models may be the curse of dimensionality, which states that beyond a certain point, the inclusion of additional parameters leads to higher probabilities of error.

Figure 1. Principal Component Analysis.



Machine Learning Techniques and Models

This section of the report will outline the machine learning techniques used for developing our models. Some of these techniques are less restrictive, which comes at the expense of interpretability. For instance, feature selection is performed when running a lasso model, which sets some of the predictors to 0. This leads to easier interpretation, as a smaller subset of predictors explains the predicted output variable. Literature also suggests that more restrictive models such as Lasso, Ridge and GAM, should be used when the research aim is to investigate the relationship between the predictive variables and the response variable. More flexible models like Random Forest, Decision Trees and GBM should be used when prediction is the only focus. While our paper's problem statement tries to solve the latter, literature has shown that shown that sometimes, the restrictive models can lead to higher predicted accuracy, as these methods are less prone to overfit (Hastie et al., 2017).

To approach our problem statement and find the most effective solution, we decided to experiment with both restrictive and flexible models. Detailed explanation on how we constructed the models will be provided. In total, we have used 6 different machine learning techniques:

1) Ridge and Lasso Classification
2) Generalized Additive Model
3) Classification Trees
4) Random Forest
5) Gradient Boosting Machine (GBM)
6) Deep Learning – Neural Networks

2.1. Ridge and Lasso Classification

Lasso, Ridge and Elastic Net are supervised machine learning techniques, which can be used for regression and classification problems and are therefore relevant for our problem statement. Importantly, these models are regularized linear models. This allows to address the immense problem of overfitting the data, as every sample tends to have misleading variables. Overfitting is not a problem of variable count, as a model can overfit with as much as ten variables. Hence, ignoring the possibility of overfitting the data can potentially result in a predictive model that puts more weight on the noise and not the actual trends. Alongside preventing overfitting, these models lower the error between predicted and actual observations (Hastie et al., 2017, p. 22).

While Lasso and Ridge can handle overfitting with integrated regularization methods, it would be of importance to note that they do so in a different way. Ridge pushes the coefficients of the different variables towards zero, but all coefficients remain non-zero. Lasso on the other hand, pushes the coefficients towards zero and set the irrelevant features to zero. In this sense, Lasso is performing feature selection alongside regularization (Hastie et al., 2017).

2.1.1. Explanation and Implementation of Model

To find the best model to predict on the test set, we need to identify whether to use a Ridge, Lasso or Elastic Net model. The R Package "glmnet" incorporates the possibility to perform the desired models.

To experiment with the different models, we had to specify alpha in the glmnet function. Alpha had to be set to 0 for Ridge, to 1 for Lasso, and to any number in the range of 0 to 1 to construct a hybrid between Ridge and Lasso, known as an elastic net model. Most importantly, we needed to specify lambda, the regularizing parameter. When lambda is set to 0, the model won't penalize overfitting. The higher the lambda the closer the coefficients of the predictive

variables will be shrinked to 0. For instance, it is crucial to prevent overfitting, hence tuning the regularizing parameter, i.e lambda, was an essential part of the task to find the best performing model (Hastie et al., 2017, p. 219). To do so, cross-validation had to be performed on the train dataset for different models. We decided to use the cv.glmnet function to perform a 10 fold cross-validation on the different models and find the lambda that minimizes the predictive error. Cross validation is used to validate the reliability of a model. What cross-validation does is picking random samples of the dataset. If the model is told to perform a 10-fold cross-validation, the model will do the process of cross-validation 10 times. Throughout this learning process the model optimizes the parameters so that it fits the data as best as possible.

Finally, as our dependent variable is binary, we need to set the family parameter to "binomial" in the code.

When it comes to implementation of the model, it is important to note that the dataset used for analysis had an extensive number of variables, which limited this papers analysis efficiency due to extensive computing time. We therefore chose to simplify the dataset, by first performing a Lasso model for feature selection on the train set. The resulting features with non-zero coefficients have been selected from the initial dataset, such that the dataset could be reduced from 73 to 31 variables (Table 4). We then split into a train-, validation- and test set.

Table 4. Lasso Variables.

LASSO VARIABLES		
Market Capitalisation	Common-Ordinary Stock (Capital)	Retained Earnings – Other Adjustments
Unemployment Rate	Common Stock Equivalents	Revenue – Total
Consumer Price Index	Employees	Sales-Turnover
Volatility Index	Intangible Assets – Total	Special items
GDP	Noncontrolling Interest	Treasury Stock – Common
Assets - Other	Noncontrolling Interest – Total	Treasury Stock – Number of Common Shares
Book Value Per Share	Notes Payable	Treasury Stock – Preferred
Capital Expenditures	Property, Plant, Equipment – Total	Income Taxes – Total

Common Equity – Tangible	Preferred Stock	Extraordinary Items and Discontinued Operations
Cost of Goods Sold	Preferred-Preference Stock – Redeemable	
Operating Expenses – Total	Retained Earnings	

Furthermore, we used the variables resulting from the PCA, to perform the same analysis. The PCA dataset has been split similarly.

The next step was to evaluate 20 different models on both subsets, in other words, experiment with 20 different alpha values to find the best model. To do so, a parameter grid has been created and implemented into a coded loop. This loop finds the optimal lambda values for the 20 alpha values, and in each run also use the resulting lambda to run the model for each alpha value. Meanwhile, each model, and its corresponding alpha and lambda have been stored.

We then proceeded to the evaluation of each model. To do so, a loop has been coded for the length of the number of computed models. At each run, one model has been used to predict on the validation set. The resulting prediction has then been compared to the actual values in a confusion matrix to retrieve the accuracy of each model. This accuracy was stored for each of the models.

Finally, to compare and evaluate the performance of the different models, we choose to visualize the accuracy of the different models, performed on the Lasso and PCA subsets, separately. The results have therefore been plotted, where the x-axis represents the different alpha values, while the y-axis depicts the corresponding accuracy of prediction. The plots have been coded such that the model with the highest accuracy is highlighted in each case.

2.1.3. Results from Validation Set

The analysis led us to the conclusion that the best performing model for both subsets would be a Ridge model, as depicted in Figure 2 and Figure 3. Hence the hyper parameter should be set to alpha = 0 for further analysis. In terms of accuracy, using the PCA subset or LASSO subset, did not change the accuracy. However, two different lambda values have been used, namely

lambda_LASSO = 7.7209, lambda_PCA = 6.4754. Therefore, we proceed to predicting on the test set with both subsets using a Ridge model.

Figure 2. Accuracy of Different Models using Lasso Subset.

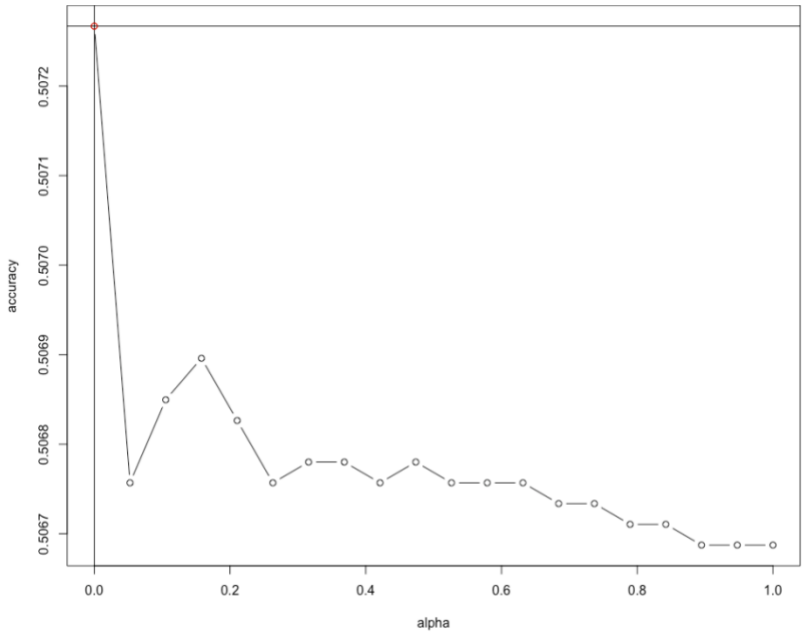
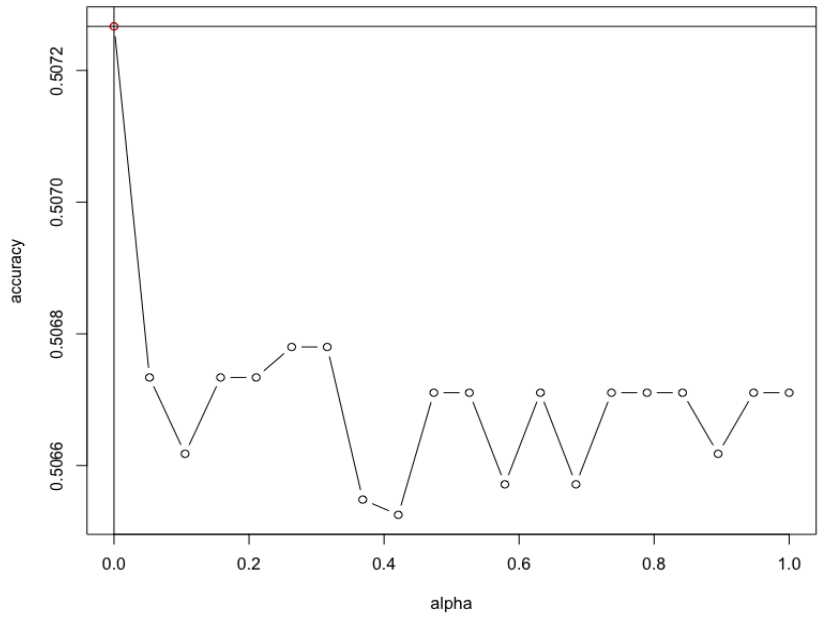


Figure 3. Accuracy of Different Models using PCA Variables.



2.1.4. Results from Test Set

To evaluate the performance of the best model on the test set, confusion matrices have been computed, which revealed that both models predicted identically and has been summarized in Table 5.

Table 5. Confusion Matrix

	Reference	
Prediction	*NegativeER*	*PositiveER*
NegativeER	*21109*	*17520*
PositiveER	*0*	*0*

Sensitivity	*0%*
Specificity	*100%*
Pos Pred Value	*0%*
Neg Pred Value	*54.65%*
Accuracy	*54,65%*
AUC	*50%*

The confusion matrix depicts that both models predicted overall correctly 54,65% of the time. Due to the constellation of the test set, we have an accuracy and negative predicted value of 54.65%, since 54.65% of the actual test excess return was actually negative (0) and 45,35% was positive (1). However, when we take a better look at the confusion matrix we can see that the models predicted that none of the stocks would beat the market. They predicted those stocks that actually beat the market, i.e. had actually a positive excess return, wrong all the time. On the other hand, they predicted the stocks that actually did not beat the market, 100% right. This suggests that the models could not perform better than chance.

To verify this result AUC's have been computed. AUC reveals which model that predict the classes best. An AUC of 50% would mean that the model cannot differentiate between the classes. In our case, the AUC was 50% for both models. This result could potentially be explained due to the threshold used to classify the predicted output variable. Predictions were computed in the form of predicted probabilities. As all probabilities were below 50%, classifying the probabilities with a threshold of 50% led to predicting only negative excess

return for all months and all stocks. Lowering the threshold to get better prediction was not applied, as we perceived this to be more of an intuitive and not statistically justified method.

2.2. Generalized Additive Model

The generalized additive model is a supervised machine learning technique, which can be used for regression problems. It provides the possibility of performing logistic regression, which predicts probabilities on a binary dependent variable. The generalized linear model is therefore of relevance to perform the necessary analysis on our dataset. The generalized additive model differs from a linear model, in the sense that it can identify nonlinear relationships by smoothing the predictive variables to fit the data. Hence, the smoothing function allows to create a non-linear relation that will be close to all data points (Hastie et al., 2017).

2.2.1 Explanation and Implementation of Model

Running a generalized additive model is relatively time expensive considering the large numbers of predictors in our dataset. To find the best model to predict on the test set, we decided to perform analysis on different subsets that have been computed throughout our analysis process, namely the PCA dataset, non-zero coefficients features from our LASSO model, and the top 90% of the most important GBM features.

To perform different generalized additive model, we used the R Packages "mgcv" and "gam". Here, the smoothing method needed to be defined. The "mgcv" package offers the possibility of choosing different methods to smooth the function. To evaluate whether different smoothing methods lead to potentially different or improved in results, we decided to train our models using two different smoothing parameter estimation methods, using the restricted maximum likelihood (REML) and the generalized cross validation criteria (GCV). REML is said to perform well in preventing overfitting. GCV is said to ensure low prediction error when the sample size is infinite. Considering, we are dealing with big data, we decided that it would be appropriate to use GCV (Miller D., 2017). Finally, as our dependent variable is binary, we need to perform a logistic regression, i.e. the "family" term needs to be set to binomial.

Once the models were fitted, we proceeded to predict on the validation set. The predictive values were probabilities, and thereby had to be classified in terms of our binary output variable. We decided to set the threshold to 50%, such that probabilities above 50% would be classified as positive excess return, and below or equal to 50% as negative excess return.

To evaluate the performances of the six different models on the validation set, we computed confusion matrices, from which we retrieved the specificity, sensitivity and accuracy of prediction. Furthermore, each AUC was computed to paint the bigger picture and thereby evaluate how well the models performed in terms of differentiating between positive and negative excess return.

2.2.2. Results from Validation Set

The results have been summarized in Table 6. By analysing and comparing the results of each model, for the respective smoothing method, we can observe that in term of specificity the GCV method overall outperformed the REML approach to smoothing.

Table 6. Results from Validation Set.

Predictors Set	**LASSO**		**PCA**		**GBM**	
Smoothing Method	REML	GCV	REML	GCV	REML	GCV
Specificity	62,49%	64,72%	66,01%	71,53%	63,69%	64,05%
Sensitivity	43,53%	41,40%	35,55%	27,79%	42,39%	41,15%
Pos Pred Value	51,82%	51,76%	49,87%	49,04%	51,78%	51,47%
Neg Pred Value	54,44%	54,72%	51,85%	50,12%	54,58%	54,73%
Accuracy	52,87%	52,89%	50,56%	49,34%	52,88%	52,88%
AUC	53,01%	53,06%	50,78%	49,66%	53,04%	53,05%

Specifically, using the PCA method led to the highest specificity percentage. However, when examining the different specificity percentages, one might notice that the LASSO subset and GBM outperform the model that was performed, using PCA components. Nonetheless, REML provided better results predicting true positives, and when looking at the positive predicted values REML outperformed GCV. Moreover, the accuracy paints a more general picture, and depicts the overall accuracy of the prediction of the model. Naturally, the models that had a

smaller difference between specificity and sensitivity, performed in general better, namely the models build on the LASSO and GBM subsets.

We can see that using the LASSO and GBM subsets led to higher AUC scores. With our problem statement in mind, we concluded that we should focus on the positively and negatively predicted value percentage. It helps us to make the final investment decision and is therefore crucial. Hence, as highlighted in the table above, and with a positive result in terms of accuracy and AUC, we decided to use the LASSO and GBM subset with REML.

Conclusively, we decided to exclude the PCA subset from further analysis since it was outperformed on several evaluation dimensions by the LASSO and GBM subsets.

2.2.3. Results from Test Set

The results from the prediction on the test set, are summarized in Table 7.

Table 7. Results from Test Set.

	LASSO	**GBM**
Sensitivity	99,95%	99,97%
Specificity	0,05%	0,05%
Pos Pred Value	45,35%	45,36%
Neg Pred Value	52,63%	68,75%
Accuracy	45,36%	45,37%
AUC	49,99%	50,02%

One can observe that both models performed well when predicting the stocks that actually beat the market, with 99,95% and 99,97% for Lasso and GBM respectively. However, it predicted the stocks that didn't beat the market wrong in 99,95% of the cases. This led to an accuracy rate of below 50% for both models. The AUC shows that the GBM subset led to slightly better results in classifying the output values. However, due to the bad results in terms of accuracy and due to the fact that the positive predicted value is below 50%, we conclude that the additive model is not the best model to use to solve our problem statement. If we were to invest, based on the positively predicted stocks we would lose more than we win, i.e. we would lose 54,65% of the time.

Potential reasons for the lower accuracy and the positive predicted values, could be led back to our approach of splitting the data. Since we are dealing with time-series data, we used a chronological three ways split into train-, validation- and test set. The model has therefore been trained on a dataset from 2010 to 2015. It is commonly known that the year previous to the test set, would be the most influential. Hence, training the model until the data of 2016 might have led to better results, as it would incorporate a continuous trend.

2.3. Classification Trees

Decision trees are frequently used in machine learning. In its essence, a decision tree can be applied to represent decisions and decision making in a graphical and intuitive manner. When applied as a supervised machine learning technique, a decision tree is typically used as a predictive model with the objective of going from observations to conclusions. As our research question concerns a classification problem, our focus is towards classification trees. Classification trees are applicable for predicting qualitative responses, whereof the model predict if each respective observation belong the class of train observations most commonly occurred in its relevant region (Hastie et al., 2017).

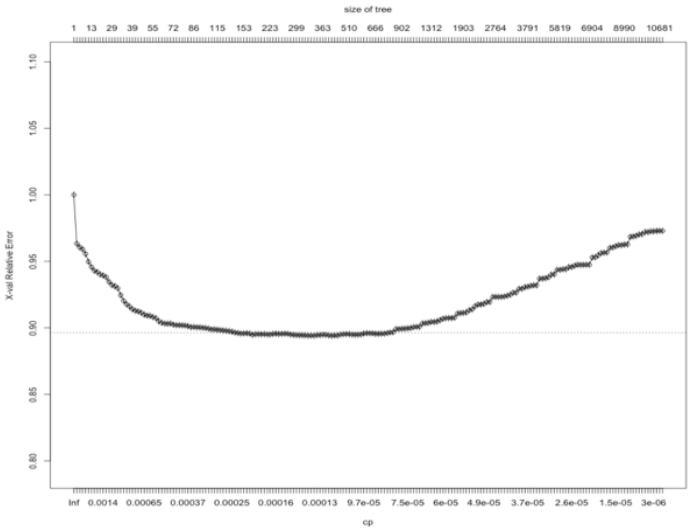
2.3.1. Explanation and Implementation of Model

The model was applied on a rather extensive dataset with a large number of variables, in practise incorporating over 70 features. In the application of classification trees to our data, we have used the R package "rpart". As discussed in the article "Optimal classification trees" (Bertsimas and Dunn, 2017), one of the most important parameters to consider, with respect to classification trees, is maximum depth. The complexity parameter is also introduced in that regard. Further, the article discusses the application of a "growing-then-pruning" approach. Simply put, when growing classification trees, the tree may turn rather complex and overfitted. In this approach, trees are initially grown large and complex, and then pruned down later on, by identifying the subtree that provides the lowest error rate (Hastie et al., 2017).

As outlined above, our strategy was to build a fully-grown tree with the intention of performing pruning later on. The reasoning behind this strategy is as follows: first, we wanted to ensure that our model took key aspects and trends in our dataset into consideration. Second, we wanted to build a rather simple model that could be seen as a benchmark to our tree-based models.

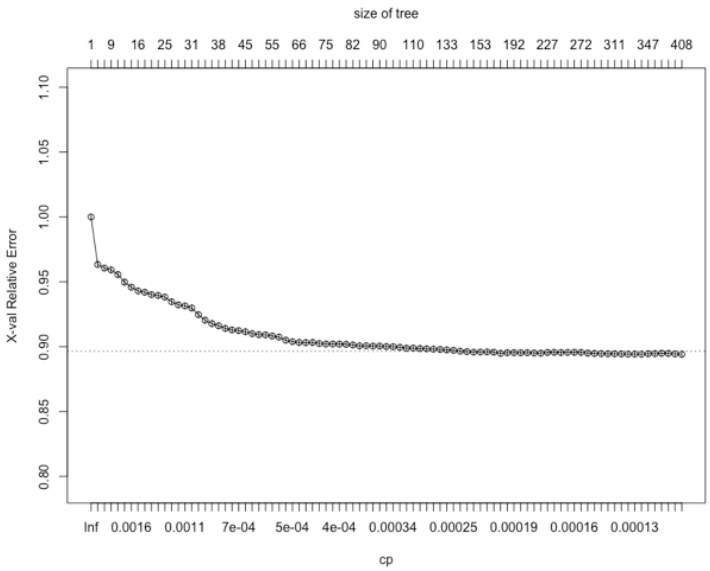
Consequently, we put the cp value to 0, thus allowing the tree to grow fully. Then the cp value yielding the lowest error was identified and applied in the pruning process. As we can observe from the complexity plot in Figure 4 below, a rather complex and large tree was grown initially. As illustrated, the error tends to decrease along the growing size of trees. From approximately 500 trees on the other hand, the error rate appears to increase significantly. A control value of 0.0001187415 yielded the lowest error and was therefore used further to prune the tree.

Figure 4. Complexity Plot for Initial Tree.



In comparison to our initial tree, the new complexity plot in Figure 5 shows that the pruned tree is far less complex. Similarly, the tree provides a lower error rate.

Figure 5. Complexity Plot for Pruned Tree.



It should be noted that classification trees, compared to many other classification methods, are considered to have less predictive accuracy. Further, classification trees can often lack robustness, whereof a minor change in the data may lead to significant changes. Other machine learning techniques such as random forest and boosting can often improve the predictive accuracy of a decision tree (Hastie et al., 2017). Hence, with respect to tree based models, we have focused on building a random forest model and a gradient boosting machine (GBM) model. The application of both these methods will be discussed further later on. However, we will first discuss the results obtained from our decision tree model.

2.3.2. Results from Validation Set

Looking at the validation results in Table 8, we can see that the pruning led to a decrease in the sensitivity rate compared to the initial tree. More precisely, while the classification has a sensitivity just barely above 50.54%, the sensitivity has decreased to just below 45% for the pruned tree. The pruned tree did as such perform worse with respect to actual positives. On the other hand, the pruning led to an increase in the overall accuracy and AUC, yet that is on the cost of a more balanced sensitivity and specificity observed in the classification tree.

Table 8. Results from Validation Set.

	Classification tree	**Pruned classification tree**
Sensitivity	50,54%	44,68%
Specificity	51,30%	61,83%
Pos Pred Value	50,20%	53,20%
Neg Pred Value	51,64%	53,50%
Accuracy	50,93%	53,38%
AUC	50,92%	53,25%

	Classification tree		**Pruned classification tree**	
	0	1	0	1
0	11 227	10 513	13 530	11 759
1	10 657	10 744	8 354	9 498

2.3.3. Results from Test Set

Reviewing the results from the test set, the sensitivity rate of both the initial classification tree and the pruned classification tree has reduced significantly, both having sensitivity rates below 50%. That is especially the case for the pruned classification tree, now having a sensitivity rate of 17.92% compared to 44.68% on the validation set. The sensitivity of the classification tree has similarly decreased, yet less; approximately 9 percent points. The true positive rate is not sufficient neither for the initial classification nor the pruned classification tree. On the other hand, the accuracy of the classification tree is slightly higher on the test set, yet the AUC has decreased while both the accuracy and AUC of the pruned tree are lower. Simply put, we can observe that the pruned classification tree performs significantly worse than the fully-grown tree with respect to actual positives.

Although a higher sensitivity rate among the models are preferred with respect to our research question, one should bear in mind that our classification tree is fully grown. As decision trees tend to overfit, and the best performing model in this particular case is the initial full-grown classification tree, careful considerations should be made with respect to solely relying on this model. Hence, this leads us to experimenting further with other tree-based models.

Table 9. Results from Test Set.

	Classification tree	**Pruned classification tree**
Sensitivity	41,82%	17,92%
Specificity	58,70%	80,71%
Pos Pred Value	45,66%	43,54%
Neg Pred Value	54,86%	54,23%
Accuracy	51,04%	52,23%
AUC	50,26%	49,32%

	Classification tree		**Pruned classification tree**	
	0	**1**	**0**	**1**
0	12 391	10 194	17 037	14 380
1	8718	7 326	4072	3 140

2.4. Random Forest

Random forest is an ensemble-based machine learning technique that in practice construct an ensemble of decision trees. Compared to a traditional decision tree, the randomness added to the forest corrects some of the overfitting problem often seen in decision tree models. The technique is considered an improvement of bagged trees, whereof the method is tweaked to decorrelate the trees. Similar to bagging, the trees are built based on bootstrapped train samples. However, for every split in a tree, the split is chosen from a random sample of predictors. In practice, this adjustment ensures that the algorithm do not take a majority of the available predictors into consideration (Hastie et al., 2017).

2.4.1. Explanation and Implementation of Model

Similar to the classification tree, the model was applied on an extensive dataset and a large number of variables, considering over 70 features. Running a random forest model was a rather time consuming and required a lot of computing power. The R package 'randomForest' was applied in order to introduce this machine learning technique to our dataset. The algorithm allows users to build a random forest model and introduce multiple parameters that can be tuned for potential performance improvement. Some of the most important parameters to consider are the amount of randomly drawn variables ("mtry"), number of observations drawn for training at each tree ("sampsize"), minimum amount of observations in terminal nodes ("nodesize") and number of trees ("ntree") (Probst, Wright and Boulesteix., 2018). These parameters were therefore considered appropriate with respect to the implementation of a random forest model.

The parameter *mtry*, as briefly introduced above, is the amount of randomly drawn variables at each split during the process of growing trees. The parameter constitutes one of the most central random forest parameters, whereof a low mtry value typically leads to trees that are different and less correlated. While this can provide better stability, a low mtry may provide trees which perform worse on average, due being built on a small set of randomly sampled variables. Hence, the balance between stability and accuracy is important with respect to mtry (Probst, Wright and Boulesteix., 2018) . *Sampsize* on the other hand, defines the number of observations that shall be drawn at the training of each tree. More diverse and less correlated trees can be achieved by decreasing the parameter value, while the accuracy similarly tend to go down when

decreasing the value of sampsize, due to less observations used in training. As in mtry, there is a trade-off between stability and accuracy (Probst, Wright and Boulesteix., 2018).

The nodesize parameter sets the minimum amount of observations on the terminal node. Lowering the value leads to trees of larger depth, as more splits are performed (Probst, Wright and Boulesteix., 2018). On the other hand, the number of trees should be set to an appropriately high level, where a high number of trees typically are preferred over a small number of trees. That said, the number of trees required will naturally depend on the dataset. For variable importance in particular, more trees are required in order to get a stable estimate of variable importance, whereof the higher level of trees trained, the better and more stable predictions are yielded (Probst, Wright and Boulesteix., 2018). The same can be said with when it comes to performance.

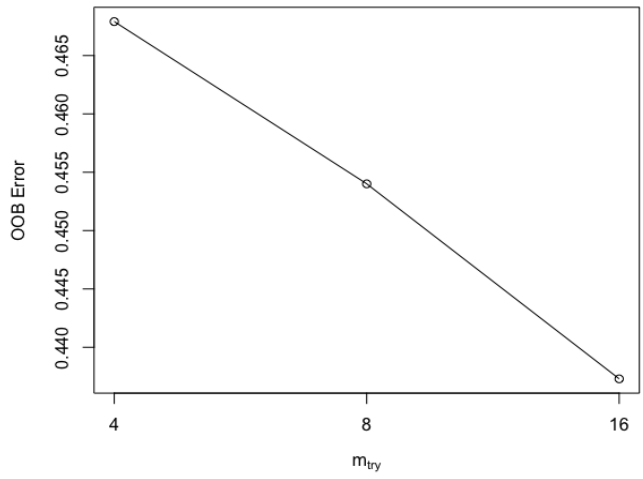
With respect to tuning search strategies, we initially implemented a grid search strategy to our model, applying all of the parameters described above. First, a set of hyperparameter values was defined for the purpose of being applied to our data. We found it important to introduce a somewhat wide range of hyperparameter values in order to capture the performance improvements of our tuning. Second, a loop was created and incorporated into the model. Doing a grid search on our random forest model turned out to be a rather time-consuming task with respect to computing time. As discussed in Probst et al.'s article "Hyperparameters and Tuning Strategies for Random Forest", mtry is the most influential parameter value, supported both by experiments conducted in relation to this article, and by literature. Sample size and node size do on the other hand have minor influence. The number of trees should on the other hand be set high, whereof the higher number of trees, the better results both with respect to performance and variable importance precision (Probst, Wright and Boulesteix., 2018). Further, as stated in the article, random forest is known for working well with default hyperparameter values. In fact, the articles states that random forest is far less tune-able, and that typically, only a small gain in performance is achieved by tuning (Probst, Wright and Boulesteix., 2018). Taking all of this into consideration, we found it reasonable to refine our strategy slightly.

Going forward, our focus was put on trees and mtry. The number of trees was set to 500. This to have a reasonably high number of trees that still could be managed properly with respect to computing time. Further, we incorporated a tuning function included in the randomForest package, hereby TuneRF, in order to tweak the mtry parameter. In short, the TuneRF function

starts off with the default mtry value, and then identifies the mtry value that provides the lowest OOB error, searching both above and below algorithms default mtry until the OOB error reduction obtained is less than a given level. Further, by not including sampsize and nodesize in our model, the parameter values are set to its default values. For sampsize, that is all observations. Nodesize on the other hand, will be set to 1. According to Probst et al., the default value of 1 for classification will in general provide sufficient results in most cases.

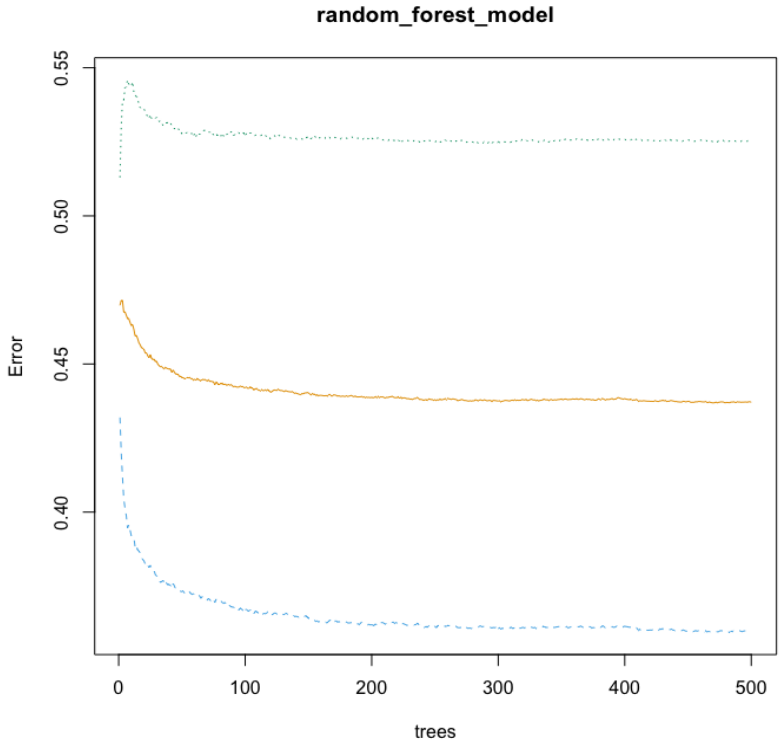
From the Figure 6, we can observe that the OOB error decreases along the increasing level of mtry. The optimal mtry yielded by TuneRF was 16 and gave an OOB error of just below 44%. As described in the section covering the parameter mtry, increasing mtry will typically improve accuracy, but may on the other hand decrease the stability somewhat.

Figure 6. Results from Mtry Tuning



Considering the model in Figure 7, we can observe that the error decreases along the increasing number of trees. The error rate appears to stabilize somewhat towards 500 trees. Although increasing the number of trees may provide a lower error, the biggest performance gain is likely to be within the already existing number of trees. We did as such find the chosen number of trees, that is 500, to be suitable going further.

Figure 7. Error Rate Across Trees.



2.4.2. Results from Validation Set

Looking at the validation results in Table 10, we can observe that the sensitivity rate is 48.22%. Hence, the actual positives are below 50%. The accuracy and AUC are both 52.67% and 52.61% respectively. Comparing this to the validation results on the initial classification tree, we can observe that the sensitivity rate is lower for the random forest model. However, the accuracy and AUC are somewhat higher.

Table 10. Results from Validation Set

	Random forest	**Classification tree**
Sensitivity	48,22%	50,54%
Specificity	56,99%	51,30%
Pos Pred Value	52,13%	50,20%
Neg Pred Value	53,12%	51,64%
Accuracy	52,67%	50,93%
AUC	52,61%	50,92%

	Random forest	
	0	**1**
0	12 471	11 006
1	9413	10 251

2.4.3. Results from Test Set

The results on the test set in Table 11 illustrates that the sensitivity rate has decreased significantly, in practise performing much worse than the classification tree on the test set. More precisely, the random forest model had a sensitivity rate of 29.16% compared to 41.82% observed on the classification tree. The overall accuracy and AUC are however higher for the random forest model. Intuitively we would expect the random forest model to perform better than the classification tree, but as we can observe, that appears to not be the case on our test set.

Table 11. Results from Test Set

	Random forest	**Classification tree**
Sensitivity	29,16%	41,82%
Specificity	70,40%	58,70%
Pos Pred Value	44,98%	45,66%
Neg Pred Value	54,49%	54,86%
Accuracy	51,69%	51,04%
AUC	49,78%	50,26%

	Random forest	
	0	**1**
0	14 861	12 412
1	6248	5 108

2.5. Gradient Boosting Machine

Reference should be made to bagging when introducing gradient boosting machine (GBM). Bagging is a technique that constructs copies of the train data applying bootstraps. A separate decision tree is fitted to each respective copy, and ultimately combined to create one prediction model, whereof each tree is built on bootstrap data independent of all other trees. GBM works in a similar manner as bagging. However, trees are here grown sequentially, based on information available from the previous trees. Unlike bagging, there is no bootstrap sampling. The tree is fitted on a modified version of the original data (Hastie, T. et al. 2017). The idea and motivation behind boosting is to combine outputs of many weak classifiers – that is a classifier with an error rate barely stronger than random guessing – to a powerful ensemble, eventually converting weak classifiers to strong ones (Gareth, Witten and Hastie, T., 2017).

2.5.1 Explanation and Implementation of Model

To introduce GBM to our dataset, the R package "gbm" was applied. The package allows users to setup a boosting model, with the possibility of tuning multiple hyperparameters to potentially improve performance. With reference to Ridgeway, G.'s article "Generalized Boosted Models: A guide to the gbm package", we built a model, with special attention towards the parameters "ntree" and "shrinkage", that is, the number of trees and the learning rate.

We ran two separate models, considering both the out-of-bag method and cross-validation. In addition, we found it important to estimate the optimal number of iterations through early stopping in order to avoid potential overfitting. As our research question concerns a classification issue, we found it appropriate to apply a Bernoulli distribution. This distribution is commonly used and recommended for classification problems (Ridgeway, 2006).

2.5.1.1 Shrinking and trees

The shrinkage, also referred to as the learning rate, controls the impact of the subsequent trees added to a tree ensemble. As a general rule, smaller values of the parameter shrinkage provide an improved prediction performance. However, lowering the shrinkage is rather costly with respect to computation time. A minimum shrinkage rate of 0.01 is suggested in Ridgeway's article. The number of trees on the other hand, should be at a sufficiently high level. With respect to trees, Ridgeway's article suggests 3000 as the minimum number of trees. Hence, the shrinkage rate and number of trees was set to 0.01 and 3000 respectively (Ridgeway, 2006).

It should be noted that the number of trees and shrinkage rate that was used in our model turned out to be a rather time-consuming task with respect to computing time, despite the fact that we are in the lower end of the "tree"- and "shrinkage"-scale suggested in Ridgeway's article. As our models in addition to this were ran both applying the OOB-method and cross-validation, our initial grid search tuning strategy was not reasonable to conduct in practice with respect to computing time. As stated above, the focus was therefore put on the abovementioned parameters and methods.

2.5.1.2 Early Stopping

Early stopping is applied in the process of training and is typically used to avoid overfitting. Simply put, the ideal time to stop a training session is at the point where the error rate has decreased and started to stabilize, and naturally before the error rate starts to increase as a consequence of overfitting. In short, one can identify the optimal number of trees through early stopping. Built in functions within the "gbm" package were applied to perform early stopping, using both an out-of-bag estimation and a v-fold cross-validation. In brief, the out-of-bag estimator tends to underestimate reduction in deviance and is considered conservative with respect to selecting the optimal number of trees. However, it can provide a fairly good performance for a small number of trees without the need of repeated fits. V-fold cross-validation on the other hand, fits a model a given number of times, estimates the corresponding cross validation error and then fits a final model with a number of trees applying all data (Ridgeway, 2006).

Cross-validation can therefore be rather CPU-intensive, especially with a high number of v-folds, but it often gives a performance boost.

2.5.1.2 Variable Importance OOB

Running a GBM model provides variable importance. The variable importance shows which variables influence the results the most by ranking them based on their relative importance. In total, there were 71 predictors of which 60 had non-zero influence. When using the OOB-method, 20 of the predictors accounted for approximately 92% of the variable importance. We note that the four additional variables were among the top five most influential variables.

In addition to the variable importance, the GBM model calculates the optimal number of trees, by using the Bernoulli deviance. The output from the model showed that the optimal number of trees were calculated to be 3000. Here the Bernoulli deviance has decreased significantly and started to stabilize. We note that 3000 trees are the same number as we set it to initially. It

is therefore natural to assume that the optimal number of trees would be higher if we set the number of trees higher. However, this would require even more computing power and was not possible in this project.

2.5.1.3 Variable importance cross-validation

In addition to the OOB-method, we ran GBM using cross-validation. When we used the cross-validation method, the variable importance was slightly different, compared to when we ran it using OOB. There were 71 predictors of which 55 had non-zero influence and 20 of the predictors accounted for approximately 92% of the variable importance. The variable importance showed that the cross-validation method generated the same 20 predictors with an influence above 1%. However, how much each predictor influenced was different, compared to the OOB method.

When we look at the optimal number of trees, we can see that the cross-validation method generates the same number as the OOB-method, 3000. As with the OOB-method, this number is equals to the number of trees we set for the model. Here too it is natural to assume that the optimal number of trees would be higher if we set the number of trees higher, but as mentioned previously, this was not possible due to the increase in processing power needed.

2.5.2 Results from validation set

When we compare the output generated from the GBM model with OOB- and CV-method, we used the area under curve (AUC). The AUCs for the two models are shown in table 12.

Table 12. Comparison of AUC in GBM

	OOB	**Cross-validation**
AUC	55.61%	55.63%

2.5.3 Results from Test Set

When we ran the GBM model on the test set, using the cross-validation method with 5 v-folds and 3000 trees, we got an AUC of 51.05%. By simply using the AUC and the accuracy as a measure, it can seem like the model is better at predicting than chance. In addition, when we look at the confusion matrix for the model, we can see that out of the 1s that the model predicted, 46.6% were correct. This is 0.7 percentage points better than our benchmark. However, the models predicted 97.22% of all the observations to be 0. This led to a specificity of 97.30%, meaning that it predicted more than 97% of the actual 0s correct. On the other hand, the sensitivity was 2.87%, meaning that the model was only able to predict less than 3% of the

observations that were actually 1, correct. This is a difference of 41.73 percentage points, compared to our benchmark. Since our goal is to identify stocks that are able to earn a higher return than the market, we conclude that the GBM model, with the above-mentioned tunings and our dataset, is not useful when trying to predict which stocks that would beat the market.

Table 13. Results from Test Set

	GBM	
	0	**1**
0	22 652	18 570
1	629	549

2.5.4 GBM with PCA

In addition to the above-mentioned GBM models, we ran GBM, using the components from PCA that explained 90% of the cumulative variance in the dataset. When using the PCA components with both the OOB method and the cross-validation method, we got the following results.

Table 14. Results from Validation Set

	OOB	**Cross-validation**
AUC	50.49%	50.80%
Early stopping	2 492	3 000

As the table above shows, the cross-validation method yielded a higher AUC and a higher number of trees, compared to the OOB method. Based on this, we used the cross-validation method when we ran the GBM model on the PCA test-components.
When using the PCA components and the cross-validation method on the test set, the model yielded an AUC of 51.47% and the following confusion matrix.

Table 15. Results from Test Set

	GBM	
	0	**1**
0	13 426	10 267
1	9 855	8 852

As can be seen in the confusion matrix, the model, using PCA components, performed better than our benchmark, both at predicting 1 and 0. In fact, we were able to predict 0s with 57.67%

accuracy, a difference of 3.87 percentage points, and 1s with 46.3% accuracy, a difference of 0.4 percentage points. We also note that the specificity of 57.67% and the sensitivity of 46.3% are both higher than the specificity and sensitivity in our benchmark.

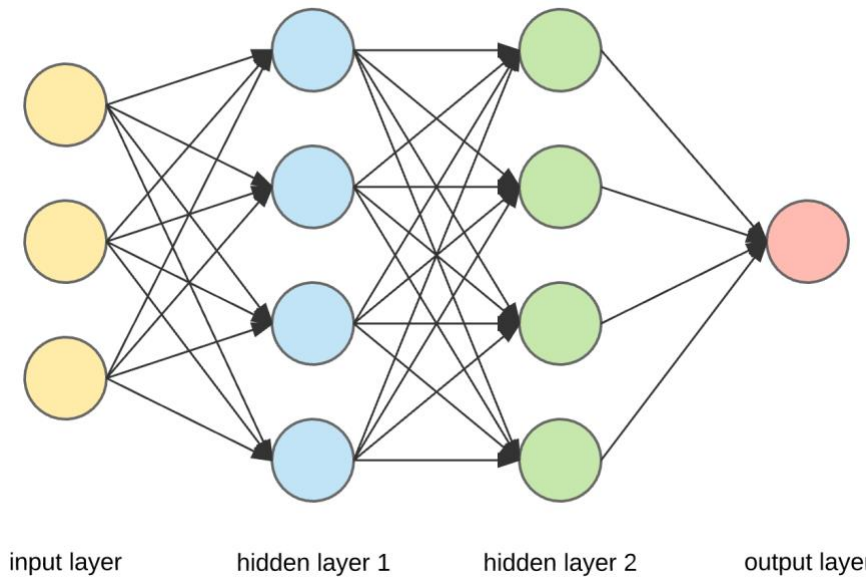
Furthermore, it is interesting to compare the GBM model with and without PCA components. We note that the model that used PCA components to predict, predicted better, and required less computing power, compared to the model that used the original variables in the dataset.

2.6. Deep Learning

We will assume that the reader has knowledge of what deep learning and neural networks are. Therefore, we will not spend a lot of time on explaining what a neural network is and what it does. Instead, we will briefly go through the basic structure of the network, so that our step-by-step process of building, training and running the model becomes clear.

For the past few years, using different kinds of neural networks has become increasingly popular because of its proved efficiency in pattern recognition. In Figure 9 we can see the structure of a basic neural network, showing an input layer, 2 hidden layers, and an output layer. This is not a replication of the model we built, but rather a brief example of the structure we used as a base when building a model more fit for our problem statement.

Figure 8. Structure of Basic Neural Network. (Dertat, 2017)



The input layer can be described as the information we feed the model in order to deduct patterns from the data. To deduct these patterns, each node in each hidden layer is assigned a weight depending on the input it got from the previous layer. It then signals to the next layer in the network if the given node should be assigned any weight depending on the information the previous node contains. Since these weight-values do not exist in any visible dataset, they are referred to as hidden layers.

One thing we found very intriguing with neural networks was the models alleged ability to learn. Throughout the training process, it continuously calculates error rates by using a process called backpropagation. It goes backwards and tries to adjust the weight associated with each node in each hidden layer to reduce the error rate. The ultimate goal is that the neural network should learn from its mistakes, by updating the assigned weight to each node properly after several runs. This makes it possible to deduct patterns and distinguish between different classes of information.

2.6.1. Building a Neural Network Model

When building a neural network there are a lot of functions that can be applied. However, which functions that would be appropriate to use will depend on what kind of network you are trying to build. As a result, you need to understand what each function, and the corresponding arguments do.

2.6.1.1. Defining Model

Since we are trying to predict whether the stock beats the market or not, we need to consider the time-series aspect of the data, when building the model. When doing stock predictions, it is important to consider trends that occur over time. In other words, we need to identify whether we have an increasing, decreasing, or no trend at all. In addition, the data might contain seasonal patterns that might influence our predictions.

When defining the model structure, we tried out a series of different numbers of input layers, hidden layers and nodes within each hidden layer, in order to do the best possible predictions. We tried to introduce both a small and large number of neurons and hidden layers. There is no perfect answer in terms of choosing the right number of nodes and hidden layers, but our initial approach was that the number of hidden layers should be somewhere between the number of

input variables and the number of output variables. We also know that a large network for a simple problem tends to overfit and "overcomplicate" things and vice versa. With regards to tuning parameters, the neural network is very sensitive to what activation, optimizer, loss function and metric we choose. Therefore, tuning and choosing the best parameters is crucial in order to build a good predictive model.

In terms of activation, for the hidden layers we used the activation function "relu", also known as a rectified linear unit. The rectifier has shown to be effective when performing supervised learning on neural networks. Because the model does not require unsupervised pre-training, the training on large and complex neural networks is faster and more effective compared to other activation functions such as "sigmoid" (DanB, 2018). For the output layer we used the "softmax" activation function. "Softmax" is a commonly used function in multiple classification logistic regression. In addition, it is useful when building neural networks with different layer levels, as it assigns probabilities to each class where the target class are given the highest probability (Polamuri, 2017).

The third aspect to consider is regularization. A common problem in machine learning is overfitting, and that goes for neural network models as well. To reduce overfitting, we introduced a regularizer in the model construction with the purpose of getting more accurate results. A regularizer allows us to penalize layer parameters or layer activity during optimization. These penalties get incorporated in the loss function in the optimization process of the network construction (Keras Documentation, No Date). Regularizers tries to keep model complexity low and at the same time fit the data as good as possible. In other words, it is controlling the trade-off between model simplicity and how well it fits the data (Claesen, 2014). For our neural network we used kernel regularizer l2. L2 regularization adds cost in a proportional manner to the square of the value of the weight coefficients (GitHub, 2017).

2.6.1.2. Compiling Model

When compiling the model to configure the learning process, we need to specify what metrics to use for measuring the performance of our model. The metrics function can be described as being quite similar to the loss function. However, the results you get from evaluating the metric is not applied for training the model. We used accuracy as a metric to evaluate the model performance (Keras Documentation, No Date).

Another argument that needs to be specified is what optimizer to use. We used the optimization algorithm "adam" which is a stochastic gradient based optimizer that updates the network weights while iterating through the training data a given number of times (Kingma, 2014).

The third argument we need to specify, is the loss function that should be applied. Since we are dealing with a binary classification problem, we used "binary_crossentropy". Binary cross-entropy is used for one hot encoded vectors, where the loss of the entire vector is the product of the loss of each single binary variable (StackOverflow, 2016).

2.6.1.3. Fitting Model

We fit the model using the input variables we have gathered in the training set, and our dependent categorical variable that we have stored in a separate dataset as the target variable.

When trying to increase the accuracy of the model, we needed to try different numbers of epochs and batch sizes. The number of epochs says something about how many times we will run through the entire dataset, forward and backward through the neural network. One epoch equals one run-through.

The batch size can be described as the number of units per sample that is used when running through the network (Sharma, 2017). If not specified, the batch size in the "keras" package is set to 32. The algorithm would then take a sample from position 1 to 32 and run through the network once. It would do this for the whole dataset. Next, it would take a sample from position 33 to 64 etc. What the optimal batch size would be will vary, depending on the dataset. In general, we can say that a small batch size will give a less accurate gradient because it uses a small sample of the total dataset to estimate. On the other hand, using small batches will make the network learn faster, because it would update the node weights more frequently (Sharma, 2017).

2.6.2. Implementation of Model

To implement the neural network model in R, we have used "keras" package. The keras package in r includes a user-friendly API which makes it easy and quick to prototype deep learning models. Keras is also capable of running on multiple back-ends like Tensorflow, CNTK and Theano (Allaire, J). For our project we are running Tensorflow in the backend.

Tensorflow is a framework created by Google and it is used for building deep learning models to deduct patterns and correlations in a given set of data.

When building the model, we made use of the function "keras_model_sequence" to make sure that the data is specified as being sequential. In addition, we used the "layer_dense" function to make the networks we are creating densely connected.

In terms of features, the final merged dataset has 72 variables. Instead of picking out variables at random, we started out with a subset of the 20 most influential variables from the GBM model. However, after training and evaluating the results we saw that this gave low overall accuracy, resulting in a network that only predicted zeros. We then tried to train the model on a new subset with PCA components accounting for 90% of the cumulative variance in the dataset. Instead of increasing the accuracy of our predictions on the test set, the PCA subset slightly lowered the overall accuracy of predicting 0s and 1s. Moving forward, we tried a new approach to be able to increase the accuracy of the model. We now went back to the initial GBM subset, but instead of choosing the 20 most influential variables, we trained and tested the results losing one variable at the time. Eventually, we found that using 14 of the most influential variables from the GBM model gave the best accuracy of all the feature selections we had done. Therefore, we decided to proceed with this subset.

After tuning the model and defining the optimal network structure we also introduced the macroeconomic variables to the model. This gave us a slightly lower accuracy compared to the previous subset.

2.6.3. Training

So far, we have explained some theory regarding the structure and tuning parameters for the neural network construction, compiling and fitting. The following section will discuss the parameters that have been used to give us the optimal results in order to answer our problem statement.

2.6.3.1. Overfitting

Neural networks have an ability to maintain complex structures of data, with a large net of hidden layers and nodes. If the network becomes too complex, the model will eventually be trained to remember each individual observation in the dataset instead of recognizing and

extracting patterns. Hence, the neural network model tends to overfit if the structure of the data gets too complex.

After running the model the first couple of times, we quickly found that we needed to do something about the overfitting problem. The first few runs gave us the following results:

Figure 9. Results Before Adjusting for Overfitting.

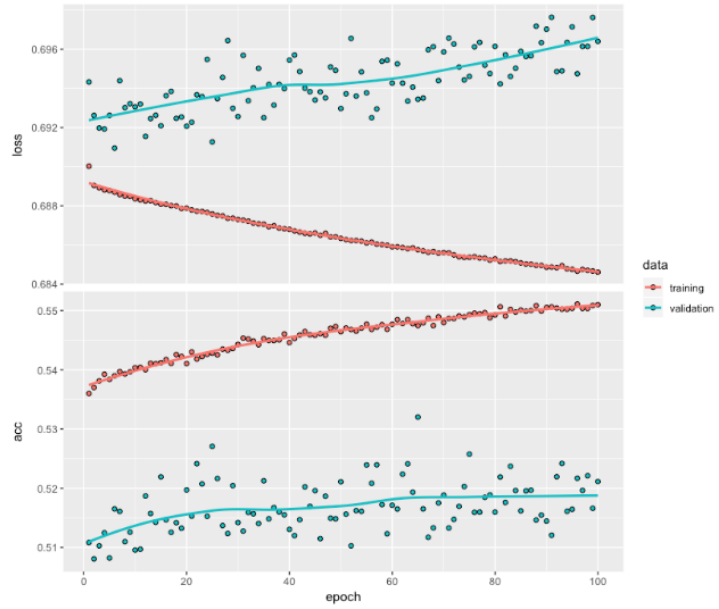
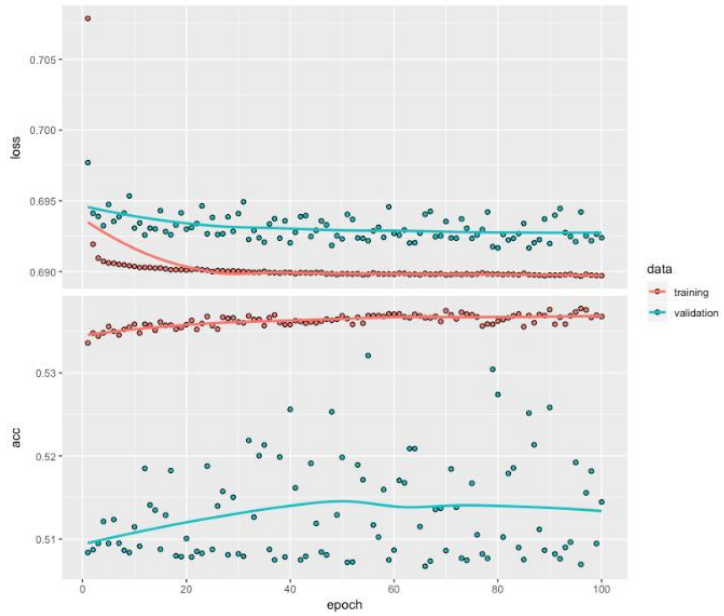


Figure 10. Results After Adjusting for Overfitting.



Before adjusting for overfitting, we had a loss function that showed a decreasing loss for the training set and increasing loss for the validation set. This clearly indicates that the model is overfitting. After introducing a regularizer to the algorithm we got a slow but steady decrease in the loss for the validation set, which indicates that the model is overfitting less. However, the accuracy did not increase, so we needed to do further adjustments to the model structure and tuning parameters.

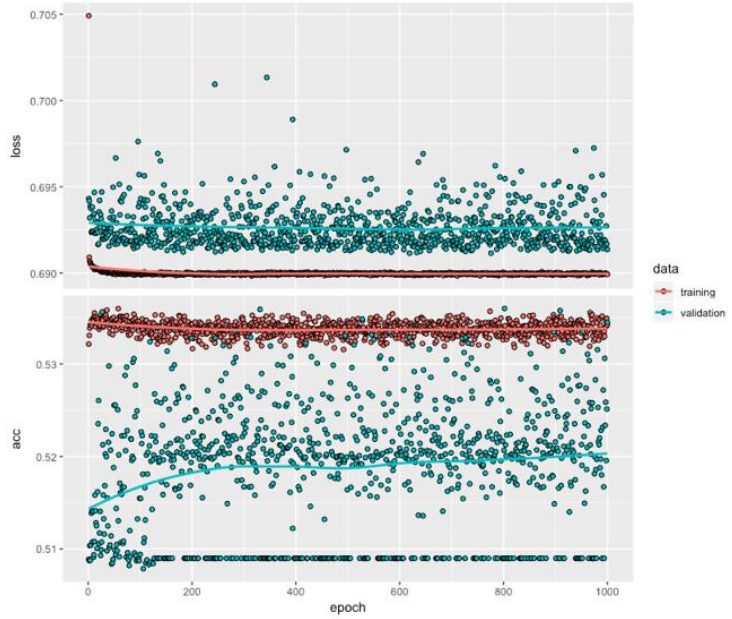
In order to increase the accuracy of our neural network, our approach was trying to train on a less complex network, a large complex network and something in-between, to get a feel of what would work best on our dataset. We then adapted the number of layers and nodes accordingly. The challenge here is to find the perfect balance between a model that is large enough to have a good learning capacity, and at the same time is generalized enough to predict good on new unseen data. After numerous runs with trial and error we ended up with a model with the following structure (Table 17):

Table 16. Structure of Our Neural Network Model

Layers	**Neurons**	**Activation Function**
Input layers	20	relu
Hidden layer 1	40	relu
Hidden layer 2	20	relu
Hidden layer 3	10	relu
Output layer	2	softmax

After implementing the structure shown in the table above, fitting the model on 1000 epochs and a batch size of 50, we got the results shown in Figure 12.

Figure 11. Results on Training Set.



The accuracy for the validation set is now slightly better. This increased the model's predictive power.

2.6.4. Results from Test Set

After our neural network is trained and validated, the model was applied directly on a new unseen and unbiased test set that had not been introduced to the model before. When we estimated the accuracy of the pre-trained and validated model, we got a total accuracy of 51.17%, as shown in Table 18.

Table 17. Accuracy of Neural Network Model on Test Set

Neural Network Accuracy	
Sensitivity	50.2%
Specificity	52.1%
Pos Pred Value	46.6%
Neg Pred Value	55.8%
Total Accuracy	51.2%

Neural Network Confusion Matrix		
	0	**1**
0	11 004	8 720
1	10 105	8 800

By looking at the results of running the model on the test set, we can see that the model is performing almost 2 percent points better than guessing if we look at the total accuracy. We are able to predict 50.23% of the observations that actually beat the market correctly, and 52.1% of the ones that actually did not beat the market correctly. However, this does not mean that it would be profitable for us to invest in the companies we predicted to beat the market. In total, we predicted 18905 observations to beat the market, when only 46.6% of those actually turned out to do so. If we were to invest in all of the stocks for each month, we would lose money in the long run. Still, we can say that the neural network model is performing better than our benchmark of pure guessing.

Comparison and Discussion

After an extensive exploration of different models and their respective test results, we proceed with a generalizing analysis of the final findings, establishing comparisons between the models and the initially developed benchmark, and evaluating the usefulness of machine learning techniques with application to our problem statement.

For ease of comparisons, a table comparing the sensitivity, specificity, positive predicted value, negative predicted value, accuracy and AUC of the final models has been computed.

Table 18. Comparison of Models

Model	**Ridge**	**GAM**		**CT**	**RF**	**GBM**	**DL**
Subsets		**LASSO**	**GBM**			**PCA**	
Sensitivity	0%	99,95%	99,97%	41,82%	29,16%	46,30%	50,23%
Specificity	100%	0,05%	0,05%	58,70%	70,40%	57,67%	52,10%
PPV	0%	45,35%	45,36%	45,66%	44,98%	47,32%	46,60%
NPV	54.65%	52,63%	68,75%	54,86%	54,49%	56,67%	55,80%
Accuracy	54.65%	45,36%	45,37%	51,04%	51,69%	52,54%	51,17%
AUC	50%	49,99%	50,02%	50,26%	49,78%	51,47%	NA

CT = Classification Tree
RF = Random Forest
DL=Deep Learning
PPV = Positive Predicted Value
PGM = Pure Guessing Method
NPV = Negative Predicted Value

We start off with the analysis of the models that performed the worst in terms of sensitivity or specificity, namely the Ridge and GAM models. We can see that these models performed poorly in classifying the dependant variable. The sensitivity and specificity show that the model predicted in general either 1 or 0. This is also illustrated by the AUC values concentrating at approximately 50%. When comparing these models to the benchmark, we can see that the accuracy for specifically Ridge classification is indeed higher than the benchmark PGM and all other models. However, when focusing on the actual problem statement and evaluating the PPV dimension, one can conclude that Ridge classification method completely disappoints on this scale. With a PPV of 0%, this model predicted all positive excess returns wrongly. This would not suggest any kind of incentive to invest. Hence, we can conclude that the benchmark outperforms Ridge classification.

Moving on to GAM, while keeping in mind that the GAM model predicts almost all observations to be 1 or 0, one can see that the model cannot differentiate between the predictive classes. Furthermore, in terms of accuracy, this model performs worse than our benchmark. While the PPV is slightly better, the NPV is slightly worse with the Lasso subset. At the same time, the GAM model with the GBM subset performs better compared to our benchmark on the same dimensions. Drawing conclusion about the usefulness of GAM is therefore not straightforward using our benchmark. Based on comparison with both the Ridge model and our benchmark, we decided that the low AUC, and accuracy despite higher PPV, does not justify the use of this machine learning method. At this stage, Ridge and GAM are therefore excluded from further comparisons.

Moving on the classification trees, one can see that the total accuracy of prediction was higher than our benchmark PGM. Investigating the differences in terms of PPV, further clarifies that this machine learning technique performs better than our benchmark. In terms of NPV, CT also performed better than our benchmark. With an AUC of slightly above 50%, CT was able to perform better than chance in classifying the response variable. Conclusively, we considered that CT has indeed been worth the time and effort as it provided better results than our benchmark.

Furthermore, the random forest model was able to outperform the results of our benchmark and the classification tree in terms of accuracy. On the other hand, it performed worse than CT in terms of AUC, PPV and NPV. Considering our problem statement, we concluded that despite RF outperforming our benchmark, it was still performing worse than CT. From this point on, we considered the benchmark to be unnecessary for further comparison.

Next, we moved on to comparing the performance of the GBM model with the CT. A quick look at the PPV, NPV, accuracy and AUC, reveals that the GBM model performed better, compared to CT. Especially the increase in PPV, is an important step towards answering our problem statement.

Finally, comparing our results from the deep learning with the performance of the GBM model, we observed that GBM outperformed deep learning on all dimensions. However, it is interesting to note that the deep learning model was our second-best method, as it performed better than CT.

Limitations

The application of machine learning techniques on a rather big data set has been a challenging yet exciting task. An important part of our research question has been to learn about and test as many machine learning techniques as possible. Starting off as a group with no or limited knowledge of machine learning and data cleaning, the learning outcome has been tremendous. On this journey, we soon realized that building and tuning precise prediction models is an art in itself, whereof experience and expertise is key. Further, we have learned that the application of machine learning techniques requires a tremendous amount of computing power. The balance between building good models and models that could be run within a reasonable amount of computing power and time, has somewhat limited our possibilities.

As discussed briefly in the report, we have applied a three-way-split on our dataset. Hence, our train set consisted of observations from the first six years, that is, 2010 - 2015. The observations from 2016 were used as the validation set, while 2017 was used as the test set (except for deep learning). An important note that should be made with respect to applying a split like this, is the fact that we now exclude the 2016 data from the train set. This means that we ignore the most recent data, prior to the test year, in the training process. If we were to include 2016 in our train set, we might have gotten different result.

As we did not perform a lot of feature engineering on our dataset and rather focused on introducing new information to our dataset, we might have missed out on valuable information that could have improved our models' predictive power.

Moreover, the fact that we included a large number of models and had a lot of ambitious ideas, might have come at the expense of time management and further analysis with each of the models.

Additional Work

In terms of additional data, we investigated different types of textual data such as news articles and annual reports that we could incorporate to our data set. Textual data is different from numeric data because the former contains opinions (soft information). The reason for incorporating textual data analysis in this project is inspired by the fact that text analysis has been shown to be important for stock predications. For instance, Lee et al. (2014) show that using text analysis of financial events reported in current reports (called 8-K documents) boosted stock price prediction accuracy.

For the textual analysis that we intended to do for this project, we decided to collect 10-K filings for the time period of 2010 to 2017 for the companies in our data set. 10-K filing is an annual report which is a vital tool for a company to communicate its strategy and financial performance and serves as the foundation for investor valuations (Jønsson and Jakobsen, 2018). We used the EDGAR database of U.S. Securities and Exchange Commission (SEC) to collect the textual data. In order to avoid confusion, it is important to note that Compustat, CRSP and SEC use different identification conventions for companies as shown in Table 19. In order to link 10-K filings to correct companies, we requested CIK (Central Index Key) for the companies in our data set from our lecturer. Our aim was to conduct sentiment analysis on collected 10-K filings.

Table 19. Identification Conventions.

	Compustat	**CRSP**	**SEC**
Unique Identifier	GVKEY	PERMNO	CIK

Sentiment analysis is an analysis of text by classifying a text as positive, neutral or negative (Dupire, 2018). Sentiment analysis tools are often based on bag-of-word models in which words in a text are looked up in a dictionary that has classified words as either positive, negative or neutral. We intended to use SentimentAnalysis package in R to calculate sentiment scores for the annual reports. The sentiment score in this package is defined as the difference between positive words and negative words, divided by all words (Rohrer and Langerfeld, 2018).

In terms of the dictionary in our sentiment analysis, we intended to use Loughran-McDonald dictionary since it is a finance-specific dictionary that suits well for the annual reports used as

our unit of analysis. Research has shown that general language dictionaries tend to misclassify financial words as negative, although in financial context they do not have a negative meaning (Loughran and McDonald, 2011). For instance, liability in a general dictionary is classified as a negative (Loughran and McDonald, 2011). In a financial context, however, liability does not have a negative meaning. Our ambition was to run sentiment analysis on these 10-K filings to give a polarity (e.g. positive, negative, neutral) score to the annual reports for each company in a given year. We wanted to include the sentiment scores to our data set and run our different machine learning models with this textual variable included.

The sentiment score could potentially show how positive or negative the company is about its performance. If a company is negatively discussing its operations and financial results, this has an effect on the expectations and opinions of investors. Thus, our aim with sentiment analysis was to capture a trend in sentiment in 10-Ks and see how this would help us to predict whether or not a stock would beat the market.

We built a code to download the data from EDGAR database as well as codes to clean the annual reports and for sentiment analysis. However, after running the code to gather 10-K filings for the first 2 years, we saw that it would take 60 hours to download this data. This was the data collection time for only 2 years out of 7 years of data needed. Additionally, it would have taken quite a lot of time to also clean these annual reports before running sentiment analysis on them. Thus, we had to take the time constraints into account and we decided to abort this task. However, since we devoted a lot of time for trying to incorporate this textual variable into our data set, we considered it to be important to mention this in this paper.

Conclusion

This project has been a fun trial and error process. For instance, we set an ambitious goal of incorporating textual data to our dataset but realised that within the time constraints we would not be able to incorporate 10-K filings and sentiment scores. One of the major difficulties was handling time-series data. For instance, when incorporating macroeconomic data, we needed to make sure that we are merging the data correctly.

Taking into account the above-elaborated comparison, we can first of all conclude that using different machine learning methods was definitely of use, as our benchmark was outperformed by several models. However, the less flexible models were unfortunately unsuccessful, as these models performed worse than our benchmark. None of the performed models, gave an incentive to consider investing in any of the positively predicted stocks, as neither of the models yielded a PPV and sensitivity above 50%. In a best-case scenario, we would need a model with a PPV and NPV of more than 50% to unravel our problem statement.

With reference to our initial problem statement: “Can we predict whether or not a stock beats the market on monthly basis using different machine learning techniques?”, we have come to the conclusion that the machine learning techniques used in this project are not are useful for predicting whether a stock beats the market or not, with regard to the tuning parameters and dataset we have used for this project.

List of References

Allaire, J. (2017). "Keras for R". Available at: https://blog.rstudio.com/2017/09/05/keras-for-r/, (Accessed: 5th November 2018).

Amibroker (2018). Walk-forward testing. Available at: https://www.amibroker.com/guide/h_walkforward.html, (Accessed: 6rd November 2018).

Analytics Vidhya (2016). "Practical Guide to Principal Component Analysis (PCA) in R & Python". Available at: https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/, (Accessed: 28th October 2018).

Claesen, M. (2014). "Does overfitting happen in this neural network?" Available at: https://www.researchgate.net/post/Does_over-fitting_happen_in_this_neural_network, (Accessed: 4th November 2018).

Bertsimas D. and Dunn, J. (2017). "Optimal classification trees". Available at: http://www.mit.edu/~dbertsim/papers/Machine%20Learning%20under%20a%20Modern%20Optimization%20Lens/Optimal_classification_trees.pdf, (Accessed: 23nd October 2018).

BLS. (2018a). Unemployment Rate. Available at: https://data.bls.gov/timeseries/LNS14000000, (Accessed: 22nd October 2018).

BLS. (2018b). Consumer Price Index. Available at: https://www.bls.gov/cpi/, (Accessed: 22nd October 2018).

Boyd, J. H., Hu, J. and Jagannathan, R. (2005). "The Stock Market's Reaction to Unemployment News: Why Bad News Is Usually Good for Stocks". The Journal of Finance, 60(2), pp. 649-672.

DanB. (2018). "Rectified Linear Units (ReLu) in Deep Learning". Available at:

https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning, (Accessed: 2nd November 2018).

Dertat, A. (2017). "Applied Deep Learning – Part1: Artificial Neural Networks". Available at:

https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6, (Accessed: 2nd November 2018).

Dupire, B. (2018). "Machine learning: Unlocking the power of unstructured data". Available at: https://www.bloomberg.com/professional/blog/machine-learning-unlocking-power-unstructured-data/, (Accessed: 29th October 2018).

Ferri, R. (2012). 'Any Monkey Can Beat the Market'. Available at: https://www.forbes.com/sites/rickferri/2012/12/20/any-monkey-can-beat-the-market/#570e5e34630a, (Accessed: 20 October 2018).

Financial Times. (2018). "Definition of Vix index". Available at: http://lexicon.ft.com/Term?term=Vix-index, (Accessed: 29th October 2018).

Jønsson, K. R. and Jakobsen, J. B. (2018). "Predicting Stock Performance Using 10-K Filings". Master Thesis at Copenhagen Business School.

Gareth, J., Witten, D. and Hastie, T. (2017). "The Elements of Statistical Learning: Data Mining, Inference, and Prediction". Available at: https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf, (Accessed: 27nd October 2018).

GitHub. (2017). "Deep-learning-with-r-notebooks?". Available at: https://github.com/jjallaire/deep-learning-with-r-notebooks/blob/master/notebooks/4.4-overfitting-and-underfitting.Rmd?fbclid=IwAR3rXzZAE8DsRQg1ik3Rzxh9D6fkZEcN2eiFoHENwBQeCMcpiWl6M3LnQOI, (Accessed: 2nd November 2018).

Hastie, T., Tibshirani, R., Friedman, J. Gareth, J. and Witten, D (2017). "An Introduction to Statistical Learning with Applications in R". Available at: https://www-bcf.usc.edu/~gareth/ISL/ISLR%20Seventh%20Printing.pdf, (Accessed: 25nd October 2018).

Keras Documentation. (No Date). "Usage of regularizers" Available at: https://keras.io/regularizers/. (Accessed: 3 November 2018).

Keras Documentation. (No Date). “Usage of metrics”. Available at: https://keras.io/metrics/, (Accessed: 6th November 2018).

Kingma, D. (2014). “Adam: A Method for Stochastic Optimization”. Available at: https://arxiv.org/abs/1412.6980v8, (Accessed: 3 November 2018).

Kleintop, J. (2018). “What Does GDP Mean For The Stock Market?”. Available at: https://www.schwab.com/resource-center/content/what-does-gdp-mean-stock-market, (Accessed: 22nd October 2018).

Lee, H., Surdeanu, M., MacCartney, B. and Jurafsky, D. (2014). “On the Importance of Text Analysis for Stock Price Prediction”. Stanford University. Available at: https://nlp.stanford.edu/pubs/lrec2014-stock.pdf, (Accessed: 22nd October 2018).

Loughran, T. and McDonald, B. (2011). When Is a Liability Not a Liability? Textual Analysis, Dictionaries, and 10-Ks. *The Journal of Finance*, pp. 35-65.

Miao, J., Wang, P. and Xu, L. (2016). “Stock market bubbles and unemployment”. Economic Theory, 61(2), pp. 273-307.

Miller, D. L. (2017). “Why is the default smoothing method “REML” rather than “GCV.Cp”?. Available at: https://github.com/DistanceDevelopment/dsm/wiki/Why-is-the-default-smoothing-method-%22REML%22-rather-than-%22GCV.Cp%22%3F?fbclid=IwAR0jsE6t9JEzER4yTsULkwWNR1RI94tRw2AWRfOOYV9ceM2b-1W41i-kssY, (Accessed: 22nd October 2018).

OECD. (2018). Quarterly GDP. Available at: https://data.oecd.org/gdp/quarterly-gdp.htm?fbclid=IwAR1OmNUYWRt6x1WRivP10AHxUa3TQUGGOR7sfEoDdgdJWUETN8uoB2X2tMw, (Accessed: 22nd October 2018).

Polamuri, S. (2017). “DIFFERENCE BETWEEN SOFTMAX FUNCTION AND SIGMOID FUNCTION”. Available at: http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/, (Accessed: 5th November 2018).

Probst, P., Wright, M. and Boulesteix, A. (2018). "Hyperparameters and Tuning Strategies for Random Forest". Available at: https://arxiv.org/pdf/1804.03515.pdf, (Accessed: 25nd October 2018).

Ridgeway, G. (2006). "Generalized Boosted Models: A guide to the gbm package". Available at: http://ftp.auckland.ac.nz/software/CRAN/doc/vignettes/gbm/gbm.pdf, (Accessed: 26nd October 2018).

Rohrer, M. and Langerfeld, C. (2018). Lecture Notes for Applied Textual Data Analysis for Business and Finance (BAN432). Norwegian School of Economics.

Shah, T (2017). "Train, Validation and Test Sets". Available at: http://tarangshah.com/blog/2017-12-03/train-validation-and-test-sets/, (Accessed 8th October 2018).

Sharma, S. (2017). "Epoch vs Batch Size vs Iterations". Available at: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9, (Accessed: 5nd November 2018).

Shen, L. (2018). "What's Really Worrying About the Inflation Fear That Hit the Stock Market". Available at: http://fortune.com/2018/02/14/inflation-stock-market-dow-sp-500/, (Accessed: 29th October 2018).

StackOverflow. (2016). "Keras binary_crossentropy vs categorical_crossentropy performance?". Available at: https://stackoverflow.com/questions/42081257/keras-binary-crossentropy-vs-categorical-crossentropy-performance, (Accessed: 3 November 2018).

Trunk, G. V. (1979). "A Problem of Dimensionality: A Simple Example". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 3, pp. 306-307, July 1979.

Wharton Research Data Services (2018). "The Center for Research in Security Prices (CRSP)". Available at: http://www.whartonwrds.com/datasets/crsp/, (Accessed: 3rd November 2018).

Yahoo Finance. (2018). CBOE Volatility Index (^VIX). Available at: https://finance.yahoo.com/quote/%5EVIX/history?period1=1262300400&period2=1541545200&interval=1mo&filter=history&frequency=1mo&fbclid=IwAR0E-

yP4Zyf4qM1Ta5aoY_PvpxutZLfLThX4NIWH7-O2ti5yyrDXDvPIIR8&guccounter=1, (Accessed: 22nd October 2018).