The Pennsylvania State University
The Graduate School

EMPIRICAL EVALUATION OF THE EFFICIENT MARKET HYPOTHESIS: A MACHINE LEARNING APPROACH

A Thesis in
Statistics
by
Isaac Dametris Wright

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2019

The thesis of Isaac Dametris Wright was reviewed and approved* by the following:

Matthew L. Reimherr
Associate Professor of Statistics
Thesis Advisor

Jia Li
Professor of Statistics

John C. Liechty
Professor of Marketing

Ephraim Hanks
Associate Professor of Statistics
Chair of Graduate Studies

*Signatures are on file in the Graduate School.

Abstract

In classic financial theory, financial markets are thought to be efficient. Therefore, stocks are assured to have any information already about them already used in the price discovery process which means models cannot forecast the future price of stocks. Empirical test of this hypothesis frequently use linear models to assess whether the hypothesis is true. However, there is no reason to believe that stock prices are linearly related to the past prices or other information like volume. In this thesis, we conduct an empirical test of market efficiency using models which do not necessarily assume that the price is linearly related to past prices. Specifically, our goal is to classify a future stock price as either a positive return or a negative return. We find that even after allowing for a nonlinear structure, the returns are indeed still unforecastable and the efficient market hypothesis holds.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would first like to thank our heavenly father for choosing me to walk this path and providing me with everything I need to be successful. I know that I am here by grace alone. Second, I would like to thank my wife for supporting me during the rough times and the good times. You believed in me during times I did not believe in myself. Third, I would like to thank my family for supporting me and encouraging me to pursue my dreams. Fourth, I would like to thank my advisor, Matthew Reimherr, for advising me on the ins and outs of life in academia. I also want to thank you for never giving up on me when progress would slow down. Finally, I would like to thank Dr.Preston, Dr. Finch, Dr. Johnson, and Mr. McCoullum. Thank you for being my eye opener. You have made me see and provided me with many opportunities. My life has been transformed because of you. Thank you for your guidance.

Dedication

I would like to dedicate this thesis to all those who will come after me and want to see someone who looks like them be successful. Let this thesis stand as motivation and evidence that you can accomplish anything as long as you believe you can.

"As a man thinketh in his heart, so he is...."- Proverbs 23:7

Chapter 1
Background

1.1 Introduction

The desire to forecast stock returns is a natural goal for any investor. There are many books on the topic such as *Machine Learning in Finance by Bob Mather* [16] and *Machine Trading by Ernest Chan* [1]. In addition, a great deal of financial research has been dedicated to this topic and it has been hypothesized that returns are unforecastable [9]. Formally, this hypothesis is called the *Efficient Market Hypothesis* (EMH) [9]. Two common definitions for market efficiency are used in the literature. We state them both below.

Definition 1.1 - A Market is efficient with respect to an information set I_t if it is impossible to make economic profit by trading on the basis of set I_t where t is a time period [9].

Definition 1.2 - A Market is efficient if prices fully reflect all available information [15].

Informally, the EMH says although you have past information on the stock, it provides no insight to what the future price will be. There are many different papers that have focused on EMH using logical arguments for why it holds. For example, [19] argues that if it were possible to make money from forecasting then there would exist a money-making machine that would effectively destroy the system; thus, it must be impossible to predict returns. In addition, [19] also argues that if investors could predict returns then acting on the predictions would push the price up and effectively erase the predicted return. Furthermore, [14] argues

that markets are efficient in reflecting new information so that prices quickly adjust to new news, thus any model would be too slow since the price already reflects the information. In other words, the current price is already conditioned on the information set. Finally, [14] also argues that news itself is unpredictable which means the price change from the news must also be unpredictable since the source is unpredictable.

There have been many empirical evaluations of the efficient market hypothesis that have shown little predictability in stocks and lead to claims that the EMH holds. However, these papers have focused on linear regression models [18]. Much of that is due to the limited computing power when they were published, however, there is little evidence to support using a linear structure [2]. More modern techniques were introduced in [2] for evaluating the EMH. Namely, the use of Neural Networks for forecasting stock returns as well as a machine learning technique for variable selection. Such techniques allow for the removal of the linear assumptions and allows the structure of the data to be explored. Trading strategies constructed from Neural Network models can offer investors higher returns than can buy-and-hold strategies where in both cases the risk exposure is the same [2].

1.2 Forms of the EMH

Early work regarding the topic of the EMH focused on using past prices to predict future prices. This class of information is the easiest to obtain so it is no surprise for this to be the case. Some investors had access to more detailed information such as accounting data and would use these in their models as well [15]. Clearly, one might think having this information would be more valuable in predicting price information, since it gives a more detailed view of the company. Malkiel [15] describes these as distinct information sets and gives formal names to test based on these information sets as well as a test based on the use of private information. The definitions are as follows.

Definition 1.3 Weak Form Test- A test of the efficient market hypothesis in which the information set contains only historical prices and possibly dividends and trading volume.

Definition 1.4 Semi-Strong Test - A test of the efficient market hypothesis in which the information set contains all publicly available information.

Definition 1.5 Strong Test - A test of the efficient market hypothesis in which the information set contains all publicly available information as well as private information.

It is more common to see tests based on weak and semi-strong information sets since this information is more "readily" available whereas a test on a strong information is hard to obtain [19]. Intuitively, it make sense to separate the information sets in this way because they are increasing in difficulty of obtainment. For example, getting past prices of stocks is free through websites such Yahoo Finance, however, one must pay a fee to access accounting data for a company. In the same way, it may be expense for anyone to obtain private information if it is at all measurable.

1.3 Fair Game Models

An earlier theoretical framework for describing how markets are efficient is through the use of "Fair Game Models". In these models, we assume the market is efficient and show this is a sufficient condition for the conditional expected excess returns to be zero [15]. Thus, no strategy has an expected excess gain. We also assume that we can state the market efficiency in terms of being equivalent to the expected return of the market. We detail this more in both view of price change and percentage change below.

First, we define several quantities.

Let $Y_{i,t}$ = Price of stock i at time t.

Let $Y_{i,t+1}$ = Price of stock i at time t+1

Let $R_{i,t}$ = 1-Period percentage return for stock i over t and t-1

Let $R_{i,t+1}$ = 1-Period percentage return for stock i over t and t+1

Let $P_{i,t+1}$ = Excess return for stock i at time t+1

Let $X_{i,t+1}$ = Excess percentage return for stock i at time t+1

Define I_t = information set at time t

Here, t is in index from $1 \dots T$. Then the conditional expectation of the price change and the conditional expectation of the percentage change can be written as the following:

$$E\left(Y_{i,t+1} \middle| I_t\right) = \left[1 + E\left(R_{i,t+1} \middle| I_t\right)\right] Y_{i.t}$$

where $E(Y_{i,t+1}|\, I_t)$ is the expected price at time t+1 and $E(R_{i,t+1}|\, I_t)$ is the expected percentage return from t to t+1 on a market that reflects all available information (from the assumption that the market is efficient and can be written as an expectation).

Now suppose an investor is interested in making economic gains based predicting the excess return. Then the investor is interested in the following quantity:

$$P_{i,t+1} = Y_{i,t+1} - E(Y_{i,t+1}|\, I_t) \tag{1.1}$$

Equation (1.1) represents the difference in the true price and the efficient markets price. Looking at the conditional expected return based on the information set the investor uses we have

$$E(P_{i,t+1}|\, I_t) = E(Y_{i,t+1}|I_t) - E(Y_{i,t+1}|\, I_t) = 0 \tag{1.2}$$

Thus, the expected excess return gain for the investor is 0.
We get a similar result for modeling the excess change in percentage as well. Namely,

$$X_{i,t+1} = R_{i,t+1} - E(R_{i,t+1}|\, I_t) \tag{1.3}$$

Equation (1.3) represents the difference in the true percentage change over a period and the expected change in percentage (again using the assumption of equating market efficiency to an expectation). Taking expectations and conditioning on the information set we have the following:

$$E(X_{i,t+1}|\, I_t) = E(R_{i,t+1}|I_t) - E(R_{i,t+1}|\, I_t) = 0. \tag{1.4}$$

Again, the expected excess gain is zero.

Notice, for all of the expected returns the set of information is arbitrary. This is consistent with the definition 1.1 in which the definition is stated for any information that could be used. Thus, although it is common to describe the form of testing being done in the context of the three different testing environments (weak, semi-strong, and strong) with the idea they may give different empirical results, the theoretical result holds for all information sets.

One interesting example detailed in [15] is a situation where a trader has some system that gives specifications for how to allocate money to investments based on different information. Call this system $\delta(I_t)$. The system can tell an investor how to allocate to n different stock at time t. Thus, we can write $\delta(I_t)$ in the following way.

$$\delta(I_t) = [\delta(I_t)_1, \delta(I_t)_2, ..., \delta(I_t)_n]$$

Let T_{t+1} = the total return based on $\delta(I_t)$. Then,

$$T_{t+1} = \sum_{i}^{n} \delta(I_t) \left(R_{i,t+1} - E\left(R_{i,t+1} | \, I_t\right)\right)$$

But the expected value of the trading system conditional on the information set is zero since

$$E(T_{t+1}|I_t) = \sum_{i}^{n} \delta(I_t) \left(E(R_{i,t+1}|I_t) - E\left(R_{i,t+1} | \, I_t\right)\right) = 0.$$

1.4 Modeling approaches

In this thesis, we will conduct an empirical evaluation of forecasting stock returns to further test the EMH. We will follow the approach from [18] and not assume prices have a linear relationship to past pricing information. In addition, our goal will be to classify out-of-sample returns as either positive returns or negative returns. We focus on whether the returns are predictable, which is different from whether the strategy is actually implementable. That is, we do not consider trading cost or whether an investor could replicate the analysis in real-time to make a purchase. We view these as distinct from whether stock returns are predictable or not, which is the basis of Definition 1.1. In particular, we focus on modeling the returns using tree-based models, discriminant analysis, and binary logistic regression.

1.4.1 Tree-Based Models

Tree-Based models, often referred to as CART (Classification and Regression Tree), are a popular method for modeling data in a non-parametric way. This is due to their ease of implementation since they do not require much tuning as some other methods as well as their natural interpretability. CART draws its name from the particular use of the tree models. If the response is categorical, the tree is being used for classification purposes. If the tree is being used to predict a continuous variable, it is regression.

The idea of CART is to partition the predictor space based on some chosen criteria and then choose a model to fit inside each partition [7]. Often, this is taken to be either the average if the data is continuous or the majority class if the data is discrete [7]. The idea is simple yet powerful and is synonymous with typical human decision making. Often, people make decisions by asking a series of logic/rational questions and the answer to each question leads to another until finally, a decision is made [8]. For example, suppose one is trying to make a decision on whether they should attend a particular graduate school. The top-level question might be does the school offer funding. If the answer is no, the decision is not to attend the school. If the answer is yes, they ask a follow up question such as do students graduate within five years. The process continues until the person decides to attend the school or not. CART follows this same process and so it is easy to follow the decision tree to see how it came to any particular conclusion.

Training a tree-based model is rather intuitive yet it can be computationally intensive. Consider being in a setting where we want to carry out a regression. If we choose to use minimization of the sum of squares as a criterion, then we only need the average in any region [12]. However, the computation required for this may be infeasible. To get an estimate we instead use a more direct, commonly referred to as greedy, algorithm that will instead look at each variable and their possible respective split points and find the one with the minimum sum of least squares of the possible splits. In this way, we simply look at a variable, check all possible split points, and find the best one amongst all the variables.

Figure 1.1 gives an illustration from a decision tree trained on a data set with $y = 5x+\epsilon$, $\epsilon \sim N(0,1)$. We can see although the data set only contains one predictor variable, we get multiple splitting points that divide up the predictor

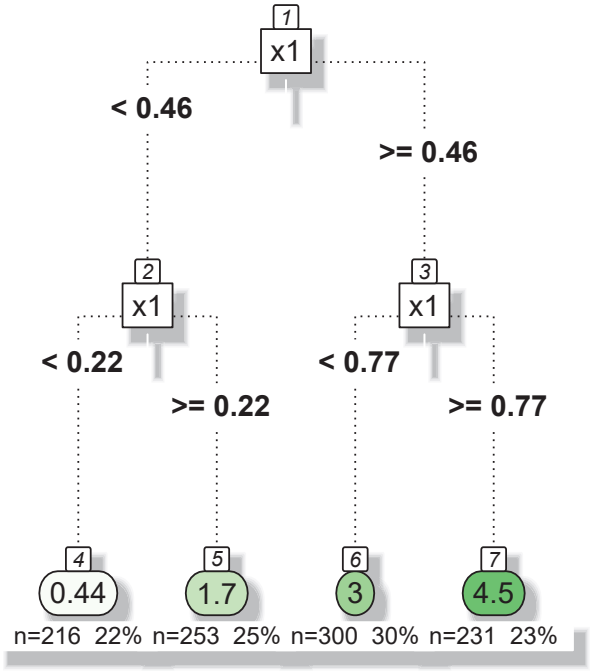

Figure 1.1. Tree fit on a simulated data set with $y = 5x + \epsilon$, $\epsilon \sim N(0, 1)$. The predictor space has been divided up into multiple subregions so any new observation of x_o can easily be used to give a prediction for y. For example, if we observe that $x_o = .60$, then our prediction for the E(Y) is 3. For reproducibility of the simulation, the seed was set to 1 in R.

space. After the division, it is easy to predict a value for a new observation. For example, suppose we observe a new x value such as $x_0 = .60$. Then the decision tree would first ask " Is $x_0 \geq 0.46$?". Since the answer is yes, we would go to the right and then ask "Is $x_0 \geq .77$?" . Since the answer is no, we would conclude that the best prediction based on this tree-model is 3. That is, our prediction of $\mathrm{E}(Y_0)$ at x_0 is 3. Which is sensible since the true model is $y = 5x+\epsilon$, $\epsilon \sim N(0, 1)$ then the true expected value is $\mathrm{E(Y)} = 5(.6) + \mathrm{E}(\epsilon) = 3$.

While this may sound like a reasonable approach, this method ignores the fact that a not so good split at one point may lead to better splits later on. In the reverse case, a good split now, could lead to useless splits later. This is a major drawback of taking the best possible split without regards to what could come after. An additional problem is the instability of the models that is inherent to the idea of looking at the training data to decide on how to split the tree. This is

because partitions now depend heavily on the training data; small changes could actually lead to large changes in the partitions. Leo Breiman introduced newer methods built on tree-based model ideas that attempt to address the problem of instability in [10] and [11]. The core of both methods is to produce multiple trees from bootstrap samples and take an average of their predictions. This takes advantage of the fact that a tree that is grown very deep has very low bias. Thus, if one can deal with the problem that the variance is high, the predictions will be significantly better.

One final issue is how to decide when to stop splitting. There does not exist a theoretical way to decide the number of best splits [8]. It is known that error rates from a training data set tend to be much lower than error from a test set (data that was not used in the training procedures) [8]. This poses a significant problem for choosing when to stop splitting a tree. On one hand, a split may decrease the training error which means it should lead to better predictions on the training set which is supposed to mirror the test set. However, this can lead to overfitting the model since continuous splitting may continue to lead to decreased training error but actually lead to worse predictions on the test set. Therefore, the use of training error as criteria of when to stop fitting the tree can't be used due to this. One might also consider setting a threshold on the sum of squares so that if no split can decrease it by a prespecified amount, then the splitting is stopped. While this may sound like a logical approach, as we discussed in the previous paragraph, the algorithm is greedy. It does not consider later splits and their sum of squares. Therefore, a split may not meet the criteria now but a split later may meet it and make predictions significantly better. Therefore, this method will fair no better than using the training error rate as a criterion for choosing the number of splits.

One potential solution is a technique known as *pruning*. For pruning, a tree is grown as deep as possible then a cost-complexity function is introduced [8]. The idea is to find a tree that is a subset of the full tree that minimizes some cost-complexity function which can be chosen to penalize harshly or penalize lightly. Often, this cost complexity function will penalize for having a larger tree based on some parameter which is usually chosen by using cross-validation techniques. Currently, there is no theoretical choice for choosing an exact value for the penalization parameter [8].

1.4.1.1 Bagging and Random Forest

As discussed earlier, one large problem with using a tree model is they have large variance. By changing one split near the top, the splits below can be different leading to different predictions. Figure 1.1 provides an illustration on how fitting a tree on a data set with two predictors where the data is simulated from y = $5x_1+7x_2 + \epsilon$, $\epsilon \sim N(0,1)$. If we observe $x_1 = .42$ and $x_2 = .42$, then labeling the trees from top to bottom as $T_1, T_2,$ and T_3 we have $T_1 = 3.6$, $T_2 = 6.7$, and $T_3 = 5.5$. Thus, we see the variability in the estimates. [10] sought to address this issue by using bootstrap samples to create a large number of samples to train trees models and take the average of their prediction. If used on this small example, the average of the 3 trees would be 5.27 and the true expected value is $E(Y) = 5(.42) + 7(.42) = 5.04$ which is closer to our estimated average than the three individual estimates. Breiman called this Bootstrap Aggregating (Bagging) .In Breiman's paper [10], he showed that the use of bootstrap samples leads to a reduction of test set error. Notably, he argued that the use of this technique will not work for models that are already stable and will actually degrade the performance.

While Bagging addresses the instability of a single tree by averaging the results of multiple trees through the use of bootstrap samples, the trees used are highly correlated [11]. This is because each tree is constructed from the same set of variables. In addition, if there is any variable with a very strong signal, it may dominate the process and cause the correlation to be higher than expected [8]. To combat this issue, [11] introduces what is called *Random Forest* . The method is similar to Bagging with the use of bootstrap samples to generate multiple samples to get an average estimate, however, the key difference is that Random Forest only considers a subset of the predictor variables at the splits. Therefore, if we have p predictors, then at any split we only consider k of those predictors where k≤p. This improves the estimate since variance of the bootstrap estimate is

$$\rho\sigma^2 + \frac{1-\rho}{B}. \tag{1.5}$$

Where $\rho =$ positive pairwise correlation , $\sigma^2 =$ variance of the individual trees, and B = Number of Bootstrap Samples. Since the last term is divided by B, then by increasing B we can lower the variance. Thus, we are left with $\rho\sigma^2$ as B tends to infinity. Thus, by selecting $k \leq p$ predictors, we reduce the pairwise correlation

of the trees to reduce the variance of our estimate. Naturally, we might think that increasing the number of trees would eventually lead to overfitting which would decrease our prediction accuracy; however, [11] also showed that the error converges. The proof can be found in the appendix of [11].

1.4.1.2 Boosting

Similar to bagging and random forest, boosting is a method of aggregating multiple trees to produce a better prediction than an individual tree. It was originally introduced as a classification tool, but has since been extended to work in a regression setting as well [8]. The distinguishing feature between boosting and random forest is that boosting uses information from other trees to increase its prediction accuracy. When generating a random forest, each tree is fit on a separate bootstrap sample so they are created independently. In contrast, boosting creates the trees sequentially by places weights on observations and trying to improve over the previous tree [4]. It does so by applying initial weights, fitting a tree, then updating the weights based on which observations were misclassified. Those observations that were misclassified are given a larger weight. This forces the next tree to give priority to classifying the missed observations correctly. Ideally, repeating this process produces trees with better predictions than the last. Not only is this method fast computationally, it has high predictive power as well.

We now describe the algorithm for creating the prediction using the tree model from section (2.1.3) except we let Y_i = -1 or Y_i = 1.
Define the following:

$$
\begin{aligned}
c_k &= \text{ weight given to model k} \\
g_k(x) &= \text{model k as a function of x} \\
\omega_i &= \text{ weight given to observation i} \\
G(x) &= \sum_{k=1}^{K} c_k g_k(x)
\end{aligned}
$$

1.) Apply the initial weights $\omega_i = \frac{1}{N}$,i = 1 to N, to the observations in the training set.
2.) For k = 1,...,K

2.1) Grow a tree to the training set.
2.2) Compute

$$\epsilon_k = \frac{\sum_{i=1}^{N} \omega_i (I(\hat{y}_i \neq g_k(x))}{\sum_{i=1}^{N} \omega_i}$$

2.3)Compute

$$c_k = \ln\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$$

2.4) Update the weights by setting

$$\omega_i = \omega_i \exp\left(c_k (I(y_i \neq g_k(x))\right)$$

2.5) Set k = k+1 and repeat unless k = K
3.) Set $\hat{G}(x) = \sum_{k=1}^{K} \hat{c}_k \hat{g}_k(x)$
For a new observation

$$\hat{Y}_i = \begin{cases} 1 & \hat{G}(x_i) > 0 \\ -1 & \text{Otherwise} \end{cases}$$

1.4.2 Logistic Regression

One classical tool for modeling in statistics is normal linear regression, where the response is continuous and we assume the we have i.i.d normally distributed errors. We also assume the mean of the outcome is a linear combination of the predictors. This model does well when the assumptions are met directly from the data. If the assumptions are not met, usually some type of transformation is used such as a natural log or inverse to conform the data to meet the various assumptions [3]. However, forcing the data to conform to a set of rigid assumptions may not always be the best approach. Consider a situation where the data is naturally skewed such as income, then if we want to understand how a particular set of covariates are related, it might be more useful to work with the data on the original scale [3]. Therefore, instead of changing the data to meet the model, we can consider changing the model to meet the data. The use of *generalized linear models* (GLM) can help solve this problem. The model still relies on modeling the response linearly but is more flexible in what kind of response variable (continuous, categorical, etc.) as well as the distribution of the response (normal, binomial, poisson, etc.) they

can handle. While GLMs can be applied in a number of different situations, in this thesis we will only discuss the case where the response is categorical since our application of the model is to a data set with a categorical response. For a more general framework and discussion on GLMs see [3].

Logistic regression is a popular statistical tool for modeling response variables that are categorical. For example, we might have the response variable smoker or non-smoker for each person in a data set and be interested in how they relate to a particular set of covariates like socioeconomic class, education, race, etc. The use of logistic regression in this scenario is called binary logistic regression i.e. the response consists of only two classes. Logistic regression can also handle response variables that have multiple classes.

Still, one might try to model them in the classical regression setting by relaxing the normal assumption since least squares does not depend on the normal assumption. To do this, the regression can be done on an indicator matrix which will have a row for each sample and a column for each class [7]. If we have k classes and N samples then the matrix will be N by K. The linear regression model is fit to each column simultaneously which give us a matrix of coefficients that is (p+1) by K where p is the number of predictors [7]. The estimate of β for the model is

$$\beta = (X^T X)^{-1} X^T Y. \tag{1.6}$$

If we have a new observation x_0 and we let $\hat{g}(x_0)^T = (1, x_0^T)\hat{\beta}$, then we classify the new observation into whichever class has the max entry i.e. max $\hat{g}(x_0)^T$.

While this might sound like a reasonable approach, the predictions might not make much sense in terms of probability. It is possible to get predictions outside of the range of possible values which is restricted to 0 and 1. Switching to logistic regression model can solve the problem of getting predictions that are outside of the [0,1].

With only two classes, we look at the relative odds of class 1 to class 0. The odds is a ratio of the probability of an event occurring divided by the probability the event does not occur [17]. If we define π_i as the probability of being in class 1 in the following way

$$\pi_i = P(Y_i = 1 | X_i = x_i) = 1 - P(Y_i = 0 | X_i = x_i) \tag{1.7}$$

where Y is the response variable and X is a set of predictor variables, then we can use the logit function to get the log odds. Define

$$f(\pi_i) := \log\left(\frac{\pi_i}{1-\pi_i}\right).$$

If we assume that $f(\pi)$ is a linear function of predictors, then we have

$$f(\pi_i) = \log\left(\frac{\pi_i}{1-\pi_i}\right) = \beta_0 + \beta_1 x_{i1} + ... + \beta_k x_{ik}.$$

Notice that if we exponentiate both sides we have

$$\frac{\pi_i}{1-\pi_i} = \exp(\beta_0 + \beta_1 x_{i1} + ... + \beta_k x_{ik}).$$

So with some algebra, we can solve for π_i to be

$$\pi_i = \frac{\exp\left(\beta_0 + \beta_1 x_{i1} + ... + \beta_k x_{ik}\right)}{1 + \exp\left(\beta_0 + \beta_1 x_{i1} + ... + \beta_k x_{ik}\right)} = P(Y_i = 1 | X_i = x_i)$$

Logistic regression can be used as a classifier by placing a threshold on the estimates of the class probabilities [7]. For example, a natural threshold in the setting with only two outcomes might be .5. Therefore, if the probability that an observation falls in class 1 is above .5, we classify as class 1. Consequently, if the probability is less than a .5, it is classified as class 0. One of the more interesting things about this classifier is the threshold can be changed to reflect the particular modeling situation. In some situations, one might desire a more stringent threshold such as .75. It is natural to think this might be the case if a bank is trying to classify someone who applied for a loan as risk or not risky. There, one might make it harder to be classified as not risky.

In addition to be useful as a classification tool, logistic regression can also be used to understand the relationship between the predictors and the response [3]. This can be seen by taking conditional odds ratios which will be discussed in more detail in chapter 2. For now, we just state the resulting interpretations below.

1)$\beta_i > 0, \text{then } \exp(\beta_i) > 1$ which means the odds will increase as β_i increases.

2)$\beta_i < 0, \text{then } \exp(\beta_i) < 1$ which means the odds will decrease as β_i decreases.

$3) \beta_i = 0,$ then $\exp(\beta_i) = 1$ which means the odds is a constant function.

Therefore, for large β_i, unit increases lead to a large positive effect on the odds. If β_i is negative then the odds decrease quickly. Thus, $|\beta_i|$ can tell us how strongly a covariate can affect the odds. This interpretability is useful in many fields since often the goal is to understand what role a set of variables play as well as their influence. For example, in the medical field when researchers are interested in mortality, it might be useful for them to interpret how smoking effects the odds of mortality. This model provides a clear interpretation for those researchers and it can do it with multiple variables.

Thus far, we have not mentioned how to estimate the parameters in a logistic regression. Unlike the case of normal linear regression, this is a complicated task since there isn't a closed form solution for what maximizes the likelihood so an iterative process needs to be taken. A popular method for estimation is the Newton-Raphson Algorithm. It involves iteratively computing the first and second derivative of the likelihood function with respect to β. Although convergence is not guaranteed, it often converges [7]. We exclude the exact details of the algorithm in this thesis but the details can be found in either [3] or [7].

1.4.3 Discriminant Analysis

Discriminant Analysis is a technique used for classifying observations into of 1 of K classes, usually based on some observed data [5]. The setup is like logistic regression in that we are interested in modeling

$$P(Y_i = k | X_i = x_i). \tag{1.8}$$

The difference is that we do this in an indirect way using Bayes theorem to reformulate the problem. That is, we take advantage of the fact that

$$\begin{aligned} P(A|B) &= \frac{P(A,B)}{P(B)} \\ &= \frac{P(B|A)P(A)}{P(B)} \\ &= \frac{P(B|A)P(A)}{\sum_A P(B|A)P(A)}. \end{aligned}$$

This changes the problem from estimating the classification probability directly, to instead looking at the likelihood that the data fall into a particular class given the class information. Typically, the data is assumed to have come from a normal distribution, and we choose to classify observations to whichever group has the highest likelihood [6]. The simplest case of dealing with the normal distribution is to assume the observations come from normal distributions with different means but have an identical covariance matrix. This is known as Linear Discriminant Analysis. A more complex case is to assume they have different means and a different covariance matrix in each class. This is called Quadratic Discriminant Analysis. We discuss both methods below.

1.4.3.1 Linear Discriminant Analysis

As stated before, in Discriminant Analysis we look at the likelihood an observation falls into a particular class. Using Bayes thereom, we can write

$$\begin{aligned}\pi_k &= P(Y_i = k|X_i) \\ &= \frac{f(X_i|Y_i = k)P(Y_i = k)}{\sum_k f(X_i|Y_i = k)P(Y_i = k)}\end{aligned}$$

If we assume the conditional distribution of the observations are normal with different means and have an identical variance, then we have LDA. One nice thing about these assumptions is that estimation is simple and straight forward. We only need to estimate $P(Y_i = k)$ and $f(X_i|Y_i = k)$. We can estimate the first by looking at the proportion of samples in each class using the raw data. For the second, we only need to estimate the parameters of the distribution. Since we assume the data is normally distributed, we need to estimate the mean in each class and a common covariance matrix. Once we have done this, we can easily make our classification chose. The following is a derivation of the classifier in the univariate case.

$$\begin{aligned}\pi_k &= P(Y_i = k|X_i) \\ &= \frac{f(X_i|Y_i = k)P(Y_i = k)}{\sum_k f(X_i|Y_i = k)P(Y_i = k)}\end{aligned}$$

$$= \frac{\frac{1}{\sigma_k\sqrt{2\pi}} e^{-(x-\mu_k)^2/2\sigma_k^2} * P(Y_i = k)}{\sum_k \frac{1}{\sigma_k\sqrt{2\pi}} e^{-(x-\mu_k)^2/2\sigma_k^2} * P(Y_i = k)}$$
$$= \frac{\frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu_k)^2/2\sigma^2} * P(Y_i = k)}{\sum_k \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu_k)^2/2\sigma^2} * P(Y_i = k)} \text{ [common variance]}$$

Taking a log we have ,

$$\begin{aligned}
log\left(\pi_k\right) &= log\left(\frac{\frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu_k)^2/2\sigma^2} * P(Y_i = k)}{\sum_k \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu_k)^2/2\sigma^2} * P(Y_i = k)}\right) \\
&= log\left(M * \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu_k)^2/2\sigma^2} P(Y_i = k)\right) \\
&= log(M) + log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) \\
&+ log\left(e^{-(x-\mu_k)^2/2\sigma^2} P(Y_i = k)\right) \\
&= M^* - \left[(x-\mu_k)^2/2\sigma^2\right] + log\left(P(Y_i = k)\right) \\
&= M^* - \frac{1}{2\sigma^2}\left[x^2 - 2x\mu_k + \mu_k^2\right] + log\left(P(Y_i = k)\right) \\
&= M^* - \frac{x^2}{2\sigma^2} + \frac{x\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log\left(P(Y_i = k)\right) \\
&= M^* - \frac{x^2}{2\sigma^2} + D_k(x) \\
&= M^{**} + D_k(x)
\end{aligned}$$

Where M** absorbs all the constant terms (terms that do not depend on k) and $D_k(x)$ has all the terms that depend on k. Notice how simple the classification becomes once we assume the distribution is normal with different means and common covariance matrix. We can estimate μ_k as the mean in each class and σ^2 as the weighted variance across classes. Then the classifier takes in an observation x and classifies it to which ever class has the highest $D_k(x)$. We can extend this to multivariate in the following way. We first note the density of the multivariate normal.

$$f(X) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu)^T\Sigma^{-1}(x-\mu)\right]}$$

Plugging in this density and proceeding as before we have

$$
\begin{aligned}
\pi_k &= P(Y_i = k|X_i) \\
&= \frac{\frac{1}{(2\pi)^{p/2}|\Sigma_k|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right]} P(Y_i = k)}{\sum_k \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right]} P(Y_i = k)} \\
&= \frac{\frac{1}{(2\pi)^{p/2}|\Sigma|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right]} P(Y_i = k)}{\sum_k \frac{1}{(2\pi)^{p/2}|\Sigma|^{-\frac{1}{2}}} e^{\frac{-1}{2}[(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)]} P(Y_i = k)} \text{ [Common Covariance]} \\
&= M * \frac{1}{(2\pi)^{p/2}|\Sigma|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right]} P(Y_i = k)
\end{aligned}
$$

Taking a log, we have

$$
\begin{aligned}
log(\pi_k) &= log\left(M * \frac{1}{(2\pi)^{p/2}|\Sigma|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right]} P(Y_i = k)\right) \\
&= log(M^*) + log\left(\frac{1}{(2\pi)^{p/2}|\Sigma|^{-\frac{1}{2}}}\right) - \frac{1}{2}\left[(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right] \\
&+ log\left(P(Y_i = k)\right) \\
&= M^{**} - \frac{1}{2}\left[(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right] + log\left(P(Y_i = k)\right) \\
&= M^{**} - \frac{1}{2}\left[x^T\Sigma^{-1} - \mu_k^T\Sigma^{-1}\right][x-\mu_k] + log\left(P(Y_i = k)\right) \\
&= M^{**} - \frac{1}{2}\left[x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_k - \mu_k^T\Sigma^{-1}x + \mu_k^T\Sigma^{-1}\mu_k\right] \\
&+ log\left(P(Y_i = k)\right) \\
&= M^{**} - \frac{1}{2}\left[x^T\Sigma^{-1}x + \mu_k^T\Sigma^{-1}\mu_k\right] + x^T\Sigma^{-1}\mu_k \\
&+ log\left(P(Y_i = k)\right) \\
&= M^{***} - \frac{1}{2}\left[\mu_k^T\Sigma^{-1}\mu_k\right] + x^T\Sigma^{-1}\mu_k \\
&+ log\left(P(Y_i = k)\right) \\
&= M^{***} + D_k(x)
\end{aligned}
$$

Again, we need only to maximize $D_k(x)$ to find out which class to assign observations to.

1.4.3.2 Quadratic Discriminant Analysis

Clearly, LDA might not produce as accurate results if the assumption does not hold. If this is the case, we can relax the assumption on the covariance matrix and let each class have a different covariance matrix. This is called Quadratic Discriminant Analysis [7]. The setup for QDA is the same as LDA except now we do not treat the covariance matrix as fixed and it now effects the ending classification function by introducing a quadratic term. Below, we derive the univariate case for QDA.

$$\begin{aligned}
\pi_k &= P(Y_i = k|X_i) \\
&= \frac{f(X_i|Y_i = k)P(Y_i = k)}{\sum_k f(X_i|Y_i = k)P(Y_i = k)} \\
&= \frac{\frac{1}{\sigma_k\sqrt{2\pi}}e^{-(x-\mu_k)^2/2\sigma_k^2} * P(Y_i = k)}{\sum_k \frac{1}{\sigma_k\sqrt{2\pi}}e^{-(x-\mu_k)^2/2\sigma_k^2} * P(Y_i = k)}
\end{aligned}$$

Taking a log we have ,

$$\begin{aligned}
log\,(\pi_k) &= log\left(\frac{\frac{1}{\sigma_k\sqrt{2\pi}}e^{-(x-\mu_k)^2/2\sigma_k^2} * P(Y_i = k)}{\sum_k \frac{1}{\sigma_k\sqrt{2\pi}}e^{-(x-\mu_k)^2/2\sigma_k^2} * P(Y_i = k)}\right) \\
&= log\left(M * \frac{1}{\sigma_k\sqrt{2\pi}}e^{-(x-\mu_k)^2/2\sigma_k^2} * P(Y_i = k)\right) \\
&= log\,(M) + log\left(\frac{1}{\sigma_k\sqrt{2\pi}}e^{-(x-\mu_k)^2/2\sigma_k^2}\right) + log\,(P(Y_i = k)) \\
&= log\,(M) + log\left(\frac{1}{\sigma_k}\right) + log\left(\frac{1}{\sqrt{2\pi}}\right) - (x-\mu_k)^2/2\sigma_k^2 \\
&+ log\,(P(Y_i = k)) \\
&= log\,(M^*) + log\left(\frac{1}{\sigma_k}\right) - (x-\mu_k)^2/2\sigma_k^2 \\
&+ log\,(P(Y_i = k)) \\
&= log\,(M^*) + log\left(\frac{1}{\sigma_k}\right) - \frac{x^2}{2\sigma_k^2} + \frac{2x\mu_k}{2\sigma_k^2} - \frac{\mu_k^2}{2\sigma_k^2}
\end{aligned}$$

$$
\begin{aligned}
&+ log\left(P(Y_i = k)\right) \\
&= log\left(M^*\right) + D_k(x)
\end{aligned}
$$

Again, we need only to maximize

$$
D_k(x) = log\left(\frac{1}{\sigma_k}\right) - \frac{x^2}{2\sigma_k^2} + \frac{2x\mu_k}{2\sigma_k^2} - \frac{\mu_k^2}{2\sigma_k^2} + log\left(P(Y_i = k)\right)
$$

since the other terms do not depend on k. Since σ^2 depends on k, the quadratic term remains in our final function. This is where QDA gets its name [8].We now derive this for the case with multiple predictors.

$$
\begin{aligned}
\pi_k &= P(Y_i = k|X_i) \\
&= \frac{\frac{1}{(2\pi)^{p/2}|\Sigma_k|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right]} P(Y_i = k)}{\sum_k \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right]} P(Y_i = k)} \\
&= M * \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right]} P(Y_i = k)
\end{aligned}
$$

Taking a log on both sides,

$$
\begin{aligned}
log\left(\pi_k\right) &= log\left(M * \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right]} P(Y_i = k)\right) \\
&= log\left(M\right) + log\left(\frac{1}{(2\pi)^{p/2}|\Sigma_k|^{-\frac{1}{2}}} e^{\frac{-1}{2}\left[(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right]}\right) \\
&+ log\left(P(Y_i = k)\right) \\
&= log\left(M\right) + log\left(\frac{1}{(2\pi)^{p/2}|\Sigma_k|^{-\frac{1}{2}}}\right) - \frac{1}{2}\left[(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right] \\
&+ log\left(P(Y_i = k)\right) \\
&= log\left(M^*\right) + log\left(\frac{1}{|\Sigma_k|^{-\frac{1}{2}}}\right) \\
&- \frac{1}{2}\left[x^T\Sigma_k^{-1} - \mu_k^T\Sigma_k^{-1}\right]\left[x - \mu_k\right] + log\left(P(Y_i = k)\right)
\end{aligned}
$$

$$
\begin{aligned}
&= log\left(M^*\right) + log\left(\frac{1}{|\Sigma_k|^{-\frac{1}{2}}}\right) \\
&- \frac{1}{2}\left[x^T\Sigma_k^{-1}x - x^T\Sigma_k^{-1}\mu_k - \mu_k^T\Sigma_k^{-1}x + \mu_k^T\Sigma_k^{-1}\mu_k\right] \\
&+ log\left(P(Y_i = k)\right) \\
&= log\left(M^*\right) + log\left(\frac{1}{|\Sigma_k|^{-\frac{1}{2}}}\right) \\
&- \frac{1}{2}\left[x^T\Sigma_k^{-1}x + \mu_k^T\Sigma_k^{-1}\mu_k\right] + x^T\Sigma_k^{-1}\mu_k + log\left(P(Y_i = k)\right) \\
&= log\left(M^*\right) + D_k(x)
\end{aligned}
$$

Again, if we classify our observation based on the max of

$$
\begin{aligned}
D_k(x) = &\frac{1}{2}log\left(|\Sigma_k|\right) \\
&- \frac{1}{2}\left[x^T\Sigma_k^{-1}x + \mu_k^T\Sigma_k^{-1}\mu_k\right] + x^T\Sigma_k^{-1}\mu_k + log\left(P(Y_i = k)\right)
\end{aligned}
$$

Since QDA has a quadratic term, it is more flexible in its decision boundary than LDA so it can sometimes dramatically reduce the bias; however, this flexibility comes with a trade-off on its variance [7]. This is typically the difference in the two methods and their performance. In addition, LDA may underperform if the assumption on the covariance matrix is far off as might be the case with a large data set where it is unlikely the classes have the same covariance structure.

1.5 Implementation

We implement the learning methods above using R packages. Namely, we use "xgboost" for the tree-based models and the default R package "stats" for the logistic regression models.

1.6 Outline

The remainder of this thesis has the following outline. Chapter 2 gives an introduction to the modeling strategies used for our analysis of the monthly return data. Chapter 3 reports the results of our analysis. Chapter 4 summarizes our finding

from the analysis.

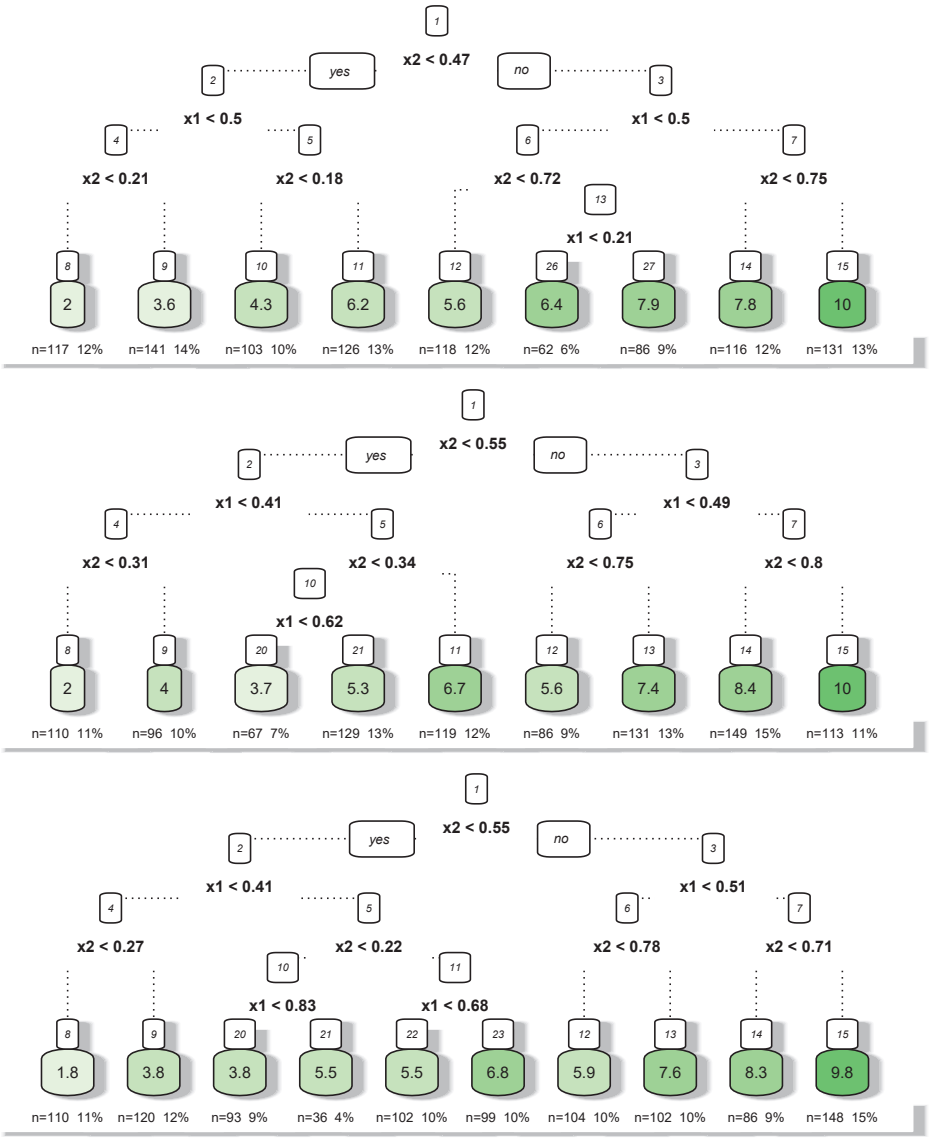

Figure 1.2. Multiple trees fit on a simulated data set with y $= 5x_1 + 7x_2 + \epsilon$, $\epsilon \sim N(0, 1)$. The predictor space has been divided up into multiple subregions for each tree fit. Although the underlying model has not changed, the three trees split at different points. The last two trees also have more splits than the first tree. The high variance of the trees makes their predictions unstable. With more predictors, the variance of the models become more apparent.

Chapter 2
Results

2.1 Data Set

The data set used was obtained from Wharton Research Data Services (https://wrds-web.wharton.upenn.edu/wrds/). It consists of 428 unique stocks tracked monthly from 1970 to 2018. We then attach several columns of lagged variables for monthly returns and monthly volume. We also add a column for whether the monthly return was positive or negative because our approach here is to classify the stock instead of predicting the price directly.

2.2 Prediction Window and Training Process

We will train the models in a rolling window. Choosing the window to train the data set is a subjective process so we will use multiple training windows. We will use 6 , 12, 18 and 24 month training windows. We will also try predicting for one month ahead, two months ahead, and 3 months ahead.

A brief example of how the training window will change is as follows. Suppose that we use a training window of 6 months and want to predict for one month ahead. Let the month we are interested in predicting be the month of July. Then what we would do is take observations from January up until June as our responses. Each month has its previous 12 month returns and volume as its predictors. After training the models and estimating coefficients, we would use the predictors to predict the class of the return for July. Then we would move the window forward. As such, the training data would now consist of the months February through July

as responses and the month of interest is now August. We then repeat training with those months and predict classes for August. We repeat this process until we have made a prediction for all months going forward. If we are interested in predicting for multiple months ahead, we can make those changes in the following way. Suppose we are interested in predicting two months ahead with a training window of 6 months. Suppose the months we are interested in predicting is July and August. Then we would take months January through June as responses for our training set with their previous 12 months of return and volume as predictors. We would then train the models and make predictions for months July and August based on our estimated coefficients. We would then update the training data to consist of observations from March until August. We would train the data on these months and make predictions for September and October. We would repeat this process until we have predictions for all of the months.

One critical reason for this approach is that in a practical scenario, an investor needs to know the future direction of returns. Thus, testing out-of-sample performance gives a more accurate picture of how well an investor would have done using the given forecasting methods. An additional bonus is that we know that our predictions results are not due to overfitting as the out-of-sample observations were not used in the training process. Both of these together mean that given good results, an investor could expect these methods to work in real-time.

We also include what we refer to as the naive method in the results. The naive method predicts the month(s) of interest's class as the majority class of the last 12 months. If the previous 12 months had 8 months of positive returns and 4 months of negative returns, we would predict the month of interest class to be positive. Alternatively, if the previous 12 months had 8 months of negative returns and 4 months of positive returns, we would predict the month of interest as negative. If there are an even number of positive and negative months (6 for both classes), the convention will be to classify the month of interest as negative.

2.3 Empirical Results

	PW = 1			PW = 2			PW = 3		
Method	Overall	Negative	Positive	Overall	Negative	positive	Overall	Negative	Positive
GLM	52.29%	47.05%	55.21%	52.32%	47.06%	55.21%	51.63%	46.11%	54.69%
Boosting	51.22%	46.32%	54.95%	51.20%	46.3%	54.92%	50.92%	45.96%	54.68%
RF	52.38%	47.23%	55.35%	52.43%	47.32%	55.40%	51.84%	46.55%	54.97%
LDA	52.43%	47.28%	55.30%	52.34%	47.08%	55.21%	51.78%	46.29%	54.79%
QDA	51.61%	46.75%	55.27%	51.60%	46.74%	55.26%	50.90%	45.97%	54.68%
Naive	52.78%	47.78%	55.66%	52.78%	47.78%	55.66%	52.78%	47.78%	55.66%

Table 2.1. Training Window of 6 months

	PW = 1			PW = 2			PW = 3		
Method	Overall	Negative	Positive	Overall	Negative	Positive	Overall	Negative	Positive
GLM	52.76%	47.51%	55.58%	52.39%	46.97%	55.29%	51.86%	46.19%	54.87%
Boosting	51.26%	46.26%	55.09%	51.14%	46.13%	54.98%	50.84%	45.75%	54.70%
RF	52.77%	47.45%	55.52%	52.58%	47.19%	55.38%	51.99%	46.38%	54.97%
LDA	52.68%	47.36%	55.48%	52.36%	46.88%	55.23%	51.80%	46.07%	54.80%
QDA	51.24%	45.87%	54.75%	51.23%	45.85%	54.73%	50.98%	45.53%	54.52%
Naive	52.83%	47.72%	55.79%	52.83%	47.72%	55.79%	52.83%	47.72%	55.79%

Table 2.2. Training Window of 12 months

	PW = 1			PW = 2			PW = 3		
Method	Overall	Negative	Positive	Overall	Negative	Positive	Overall	Negative	Positive
GLM	52.33%	46.40%	55.02%	52.32%	46.40%	55.02%	52.12%	46.12%	54.89%
Boosting	51.01%	45.85%	54.89%	50.88%	45.69%	54.78%	50.92%	45.72%	54.80%
RF	52.32%	46.48%	55.06%	52.04%	46.06%	54.88%	51.98%	46.01%	54.86%
LDA	52.47%	46.62%	55.12%	52.38%	46.46%	55.04%	52.20%	46.22%	54.94%
QDA	51.47%	45.89%	54.86%	51.34%	45.74%	54.78%	51.32%	45.69%	54.74%
Naive	52.79%	47.56%	55.80%	52.79%	47.56%	55.80%	52.79%	47.56%	55.80%

Table 2.3. Training Window of 18 months

	PW = 1			PW = 2			PW = 3		
Method	Overall	Negative	Positive	Overall	Negative	Positive	Overall	Negative	Positive
GLM	52.12%	46.12%	54.89%	52.65%	46.52%	55.08%	52.42%	46.19%	54.95%
Boosting	50.92%	45.72%	54.80%	51.03%	45.82%	54.94%	50.86%	45.61%	54.79%
RF	51.98%	46.01%	54.86%	52.49%	46.52%	55.12%	52.32%	46.26%	55.01%
LDA	52.20%	46.22%	54.94%	52.56%	46.41%	55.03%	52.41%	46.15%	54.93%
QDA	51.32%	45.69%	54.74%	51.32%	45.64%	54.78%	51.44%	45.76%	54.85%
Naive	52.79%	47.48%	55.82%	52.79%	47.48%	55.82%	52.79%	47.48%	55.82%

Table 2.4. Training Window of 24 months

Chapter 3
Conclusion

3.1 Discussion

Our result might suggest the learning methods used here are slightly useful in practice since they give slightly better predictions for predicting stocks that will give a positive return than what we might expect from random selection (50%). However, looking at the naive method of simply classifying the next month as the majority class of the previous 12 months gives a different impression. This method has an overall classification rate that is slightly higher. A two-sample t-test was used to compare the naive method to the other methods with prediction windows of 1 and we found the naive method was better in all cases with $p < .01$ in each case. This means that most months are positive in general and our models fail to pick up on this trend. Thus, the models give no real long-term advantage in terms of predicting when stocks will give a positive return. Since it takes a lot of computational power to implement, it seems better to stick with the naive method if an investor was going to choose a strategy focusing on choosing those stocks they think will increase.

We also see in Table 3.1-3.4, the overall classification (both positive and negative) is slightly above 50%, however, this is misleading because it seems to be driven mostly by classifying the positive stocks correctly. This is clear since we see that our predictions for stocks that would give a negative return is correct less than 50% of the time. In fact, it is always relatively close to 47% and never reaches 50%. The naive method is also slightly better here at classifying stocks that will have a negative return, but it is also correct less than 50% of the time.

Another interesting result is that the results tend not to change much with the prediction window. That is, predicting for 1 ,2, or 3 months ahead, the results are fairly the same. For example, Table 3.1 uses a training window of 6 months and Table 3.4 uses a training window of 24 but months but the prediction accuracy for both are roughly the same. Unfortunately, the predictions themselves don't give any long-term advantage so this fact can't actually be exploited in any way in practical to save on computation cost. In addition, changing the amount of data used in the training set also does not lead to better prediction accuracy even when we are trying to predict only one month ahead. One might think that having more data is better since it might be easier to find a trend, however, we see this is not the case. The data is noisy regardless of the amount of data we use to train the models and so the models cannot pick up on any useful signal for predictions. Notice that none of the models could beat the naive methods across prediction windows and training window length leading us to conclude the models are of no predictive use.

One of the purposes of this thesis was to examine if non-linear methods would have a better chance at predictions since there was no reason to believe the underlying structure of stock returns were linear. To that end, we thought the tree-based methods might have a good chance at predicting returns. We see that these methods were also not able to predict the returns very well. Non-linear methods tend to do well when there is a strong signal since the data is driving the method, however, the signal in stock returns seem to be weak.

One final point is to look at and compare the prediction accuracy of LDA to QDA. Notice that in almost every case, LDA outperforms QDA.This is surprising given the size of the data set and how strict the decision boundary is for LDA. Typically, the difference in accuracy of these methods can be attributed to the tradeoff between bias and variance. LDA has a stricter decision boundary than QDA so the variance is lower while QDA has lower bias because of its flexibility. Recall from chapter 2, we showed that LDA draws a linear boundary between classes while QDA draws a non-linear boundary. The decision boundary for LDA has the following form:

$$-\frac{1}{2}\left[\mu_k^T\Sigma^{-1}\mu_k\right] + x^T\Sigma^{-1}\mu_k + log\left(P(Y_i = k)\right)$$

The decision boundary for QDA has the following form.

$$\frac{1}{2}log\left(|\Sigma_k|\right) - \frac{1}{2}\left[x^T\Sigma_k^{-1}x + \mu_k^T\Sigma_k^{-1}\mu_k\right] + x^T\Sigma_k^{-1}\mu_k + log\left(P(Y_i = k)\right)$$

Thus, we see that we have a quadratic function of x when using QDA and a linear function of x when using LDA. With a data set so large, we might expect QDA to perform better since it could reduce the bias more than LDA could reduce the variance. This is because QDA is flexible enough to conform to the data while LDA would not have enough flexibility that this data set would need. Clearly, this is not the case and LDA was able to outperform QDA in almost every scenario.

Overall, our empirical test of the definition 1.1 (Efficient market Hypothesis) provides evidence that it holds. However, one should note that the predictor space is limited to past returns and past trading volume. The EMH is supposed to hold for all information sets since there was no restriction in the definition. As mention in [13], the use of information that is easy for any investor to obtain typically does not hold much predictive power in finance. This is because that information becomes "priced in" to the price of the stocks since every investor has the information. Thus, it would be interesting to see how adding accounting data such as inventory, revenue, profit margins,etc would influence our results. One might suspect that these kinds of variables should hold information about the health of a company and give more information on how the company stock will perform (since investors would likely prefer a healthier company) [15].

3.2 Limitations

One major limitation here is the computational cost of fitting the models this many times requires us to tune the models less. In most scenarios, there is one data set which we want to choose a model to predict future data well. We get a chance to manipulate the data and check assumptions to make sure the model is appropriate. We then do cross-validation and other methods to choose the parameters. In this case, we are fitting the model over many different windows so such methods dramatically change the computation time. In addition, we don't get a chance to see what the data looks like for each window to check assumptions and other details. This could have serious impact on our model's prediction accuracy, however, there

is no realistic way to fix this problem.

Appendix
R code

```
################428 stocks included in the data set

library(doParallel)
library(foreach)
library(snow)
library(gridExtra)
library(dplyr)
library(sqldf)
library(xgboost)
library(e1071)
library(MASS)
######################
###################### Sets up the data for the models
rm(list=ls())
fitlist<-c("glm","QDA")
#fitlist <- c("glm","")
load("lagmsd.Rda")
data_all <- test
dates <-as.Date(data_all$date,"%Y-%m-%d")
nrows<-length(data_all[,1])
data_all<-data_all[order(dates)[1:nrows],]

data<- data_all
```

```
###Use this piece of code if wanting a certain date range
#  Jan 2007 to Dec 2010
Rec <- FALSE
if(Rec)
{
  data <- data[data$date >= "2007-01-01" & data$date <= "2010-12-31",]
}
dates <-as.Date(data$date,"%Y-%m-%d")
days_levels<-unique(dates)
days_n<-length(days_levels)

# Day Index
# This is just to make it easier to cut
# into blocks of days
day_index<- as.numeric(dates - dates[1]+1)
day_index<-as.factor(day_index);levels(day_index)=1:days_n
day_index<-as.numeric(day_index)

rd_names<-c("permno","cusip","date")
data<-data[!(names(data) %in% c(rd_names))]



#Fit windows
fit_window<-12
predict_window<-1

numb_blocks<-floor((days_n - fit_window)/predict_window)
end_day<-fit_window +  numb_blocks*predict_window
###################End of matthew original code

count <- 0
myfun <- function(x)
{
```

```
#Create column names vector but this one will later be dropped
store_names <- c("drop")
### training and testing indices

##########################################
###########This part also from Matthew Code
train_range<-c((1+(ib-1)*predict_window),(fit_window+(ib-1)*predict_window))
train_index<-(day_index>=train_range[1] & day_index<=train_range[2])
test_range<-c((fit_window+1+(ib-1)*predict_window),(fit_window+(ib*predict_wind
test_index<-(day_index>=test_range[1] & day_index<=test_range[2])

#TRAINING & TEST DATA SHOULD CHANGE EACH TIME
train_data <-data[train_index,]
test_data  <- data[test_index,]

#Only keep if you want scaled data
if(FALSE)
{
train_data <- scale(train_data[,1:28])
train_data <-cbind(train_data,data[train_index,29:31])
test_data <- scale(test_data[,1:28])
test_data <- cbind(test_data,data[test_index,29:31])
}
########### End of Matthew code
##########################################

#Set up data for boosting and random forest
if('boost' %in% fitlist | 'rf' %in% fitlist)
{
  #Set up boosting model data
  myvars <- names(train_data) %in% c("id", "idsvm","ret","b","vol")
  want <- names(train_data) %in% c("b")
  boostdata.train <- train_data[!myvars]
  Y.train <- train_data["b"]
```

```
  Y.train <- Y.train[,1]
  boostdata.test <- test_data[!myvars]
  #boostdata.test$sp <- as.numeric(boostdata.test$sp)
  dtrain <- xgb.DMatrix(data = data.matrix(boostdata.train), label = Y.train)
  dtest <- xgb.DMatrix(data=data.matrix(boostdata.test))
}
#Logistic Regression
if('glm'%in%fitlist)
{
  store <-cbind(id = test_data$id)
  store_names <- c(store_names,"id","glm")
  glmfit <- step(glm(id ~ .  -ret -idsvm -b -vol, data = train_data,family = bi
  predglm <- predict(glmfit,newdata = test_data,type = "response")
  predictionglm <- as.numeric(predglm > 0.5)
  store <- cbind(store,predictionglm)
}



#SVM classification
if('svm' %in% fitlist)
{
  store <-cbind(store,idsvm = test_data$idsvm)
  store_names <- c(store_names,"idsvm","svm")
  best <- svm(idsvm ~. -ret -id -b -vol, data = train_data, kernel = "linear",c
  store <- cbind(store,predict(best,newdata = test_data))

}
#Boosting Model
if('boost' %in% fitlist)
{
  store <-cbind(store,tree = test_data$b)
  store_names <- c(store_names,"label","boost")
  tree1 <- xgboost(data = dtrain,max_depth = 200,verbose = 0,
                   eta = 1, nthread = 1, nrounds = 1000, objective = "binary:lo
```

```
  predtree <- predict(tree1,newdata = dtest)
  predictiontree <- as.numeric(predtree > 0.5)
  store <- cbind(store,predictiontree)

}
#PCA Classification
if('pca'%in% fitlist)
{
  store_names <- c(store_names,"pca")
  pca.matrix<-prcomp(~.-ret -id -idsvm -b-vol,rank=5,data=train_data)
  pca.train <- data.frame(id = train_data$id,pca.matrix$x)
  pca.test <- prcomp(~.-ret -id -idsvm -b - vol,rank=5,data=test_data)
  pcaglm <- glm(id ~ . , data = pca.train,family = binomial)
  predpca <- predict(pcaglm,newdata = as.data.frame(pca.test$x),type = "respons
  predictionpca <- as.numeric(predpca > 0.5)
  store <- cbind(store,predictionpca)

}
#Random Forest
if('rf' %in% fitlist)
{
  store_names <- c(store_names,"rf")
  tree2 <- xgboost(data = dtrain,
                   nthread = 1,nrounds = 1,verbose = 0, params = list(max.depth
                                                                      num_paral
                   objective = "binary:logistic")
  predtree2 <- predict(tree2,newdata = dtest)
  predictiontree2 <- as.numeric(predtree2 > 0.5)
  store <- cbind(store,predictiontree2)
}

if('naive' %in% fitlist)
{
store_names <- c(store_names,"nv")
```

```
  naive <- numeric(0)
  lagv <- test_data[ , 3:15]
  for(k in 1:nrow(test_data))
  {
    a <- ifelse(sum(lagv[k,] > 0 ) -  sum(lagv[k,] < 0) > 0,1,0)
    naive <- rbind(naive, a)

  }
  store <- cbind(store,naive)
}
  #LDA
  if('LDA' %in% fitlist)
  {
    store_names <- c(store_names,"LDA")
    ldafit <- lda(id ~ .  -ret -idsvm -b -vol, data = train_data)
    ldapred <- predict(ldafit,newdata = test_data)
    ldaclass <- ldapred$class
    ldaclass <- ifelse(ldaclass == 1,1,0)
    store <- cbind(store,ldaclass)
  }

  #QDA
  if('QDA' %in% fitlist)
  {
    store_names <- c(store_names,"QDA")
    qdafit <- qda(id ~ .  -ret -idsvm -b -vol, data = train_data)
    qdapred <- predict(qdafit,newdata = test_data)
    qdaclass <- qdapred$class
    qdaclass <- ifelse(qdaclass == 1,1,0)
    store <- cbind(store,qdaclass)
  }

  store_names <- store_names[-1]
  colnames(store) <- store_names
```

```
  return(store)

  count <- count + 1
}

#
no_cores <- detectCores()
workers <- makeCluster(no_cores,type="SOCK")
registerDoParallel(workers)
clusterEvalQ(workers,library(MASS))
clusterEvalQ(workers,library(xgboost))
out<-foreach(ib = 1:numb_blocks,.combine="rbind",.errorhandling='stop') %dopar% m
##Stop the cluster when finished
stopCluster(workers)


out <- as.data.frame(out)
#out$svm <- ifelse(out$svm ==2, 1,0)
#out$idsvm <- ifelse(out$idsvm ==2, 1,0)
#out$id <- ifelse(out$id ==2, 1,0)

out[] <- lapply(out, as.numeric)
#out$vote <- ifelse((out$glm + out$svm + out$boost + out$pca + out$rf) > 2,1,0)
#vote.tab <- table(predict = out$vote,true = out$id)
#vote.tab
#load("fit18predict2.Rda")

#pca <- table(predict = out$pca, true = out$label)
glm <- table(predict = out$glm, true = out$id)
#svm <- table(predict = out$svm,true = out$idsvm)
#boost <- table(predict = out$boost,true = out$label)
#rf <-  table(predict = out$rf, true = out$label)
#nv <- table(predict = out$nv, true = out$id)
```

```
#lda <-table(predict = out$LDA, true = out$id)
qda <-table(predict = out$QDA, true = out$id)
save(out,file = "fit12predict1.Rda")



if(FALSE)
{
##########################################
#Overall Correct
100*(glm[1,1] + glm[2,2])/(glm[1,1] + glm[2,1] + glm[1,2]+glm[2,2])
#negative
  100*glm[1,1]/(glm[1,1] + glm[1,2])
#Positive
  100*glm[2,2]/ (glm[2,2] + glm[2,1])
##########################################
##########################################
#Overall Correct
  100*(boost[1,1] + boost[2,2])/(boost[1,1] + boost[2,1] + boost[1,2]+boost[2,2])
#negative
  100*boost[1,1]/(boost[1,1] + boost[1,2])
#Positive
  100*boost[2,2]/ (boost[2,2] + boost[2,1])
##########################################
##########################################
#Overall Correct
  100*(rf[1,1] + rf[2,2])/(rf[1,1] + rf[2,1] + rf[1,2]+rf[2,2])
#negative
  100*rf[1,1]/(rf[1,1] + rf[1,2])
#Positive
  100*rf[2,2]/ (rf[2,2] + rf[2,1])
##########################################
##########################################
#Overall Correct
```

```
  100*(nv[1,1] + nv[2,2])/(nv[1,1] + nv[2,1] + nv[1,2]+nv[2,2])
#negative
  100*nv[1,1]/(nv[1,1] + nv[1,2])
#Positive
  100*nv[2,2]/ (nv[2,2] + nv[2,1])
##########################################

##########################################
#Overall Correct
100*(lda[1,1] + lda[2,2])/(lda[1,1] + lda[2,1] + lda[1,2]+lda[2,2])
#negative
  100*lda[1,1]/(lda[1,1] + lda[1,2])
#Positive
  100*lda[2,2]/ (lda[2,2] + lda[2,1])
##########################################

#Overall Correct
  100*(qda[1,1] + qda[2,2])/(qda[1,1] + qda[2,1] + qda[1,2]+qda[2,2])
#negative
  100*qda[1,1]/(qda[1,1] + qda[1,2])
#Positive
100*qda[2,2]/ (qda[2,2] + qda[2,1])
##########################################
}
```

```
png("GLM.png")
p<-tableGrob(glm)
grid.arrange(top = " GLM",p)
dev.off()
```

```
png("Boost.png")
p<-tableGrob(boost)
grid.arrange(top = " Boost",p)
dev.off()

png("RandomForest.png")
p<-tableGrob(rf)
grid.arrange(top = " RandomForest",p)
dev.off()

png("LDA.png")
p<-tableGrob(lda)
grid.arrange(top = " LDA",p)
dev.off()

png("QDA.png")
p<-tableGrob(qda)
grid.arrange(top = " QDA",p)
dev.off()

png("naive.png")
p<-tableGrob(nv)
grid.arrange(top = " naive",p)
dev.off()
```

```
if(FALSE)
{
  library(RPostgres)
  library(stringr)
  library(dplyr)
  library(getPass)
  library(zoo)
  library(sqldf)
# WRDS # WRDS LOG ON
usr <-   # need to put in your own WRDS id here
pwd <-

#------------------------
# connect to wrds databases
# connect to wrds databases

wrds <- dbConnect(Postgres(),
                  host='wrds-pgdata.wharton.upenn.edu',
                  port=9737,user=usr,password=pwd,  # change user here
                  dbname='wrds',
                  sslmode='require')



################ We will grab these permno between these dates directly from wrds
res <- dbSendQuery(wrds, "select cusip,date,ret,vol
                   from crsp.msf
                   where date between '1970-01-01' and '2018-10-01'")
data <- dbFetch(res, n=-1)
```

```
dbClearResult(res)

u <- unique(newdata$cusip)

# NA + anything gives NA such as NA + 1 = NA
none <- is.na( (data$ret + data$vol) )
n <- data$cusip[none]
u <- unique(n)
### Wow, 30976 stocks have an NA somewhere in the DATA
### Only 439 are complete
newdata <- data[!(data$cusip %in% u),]
m <- unique(newdata$cusip)
save(newdata,file = "trimmsd.Rda")

######################
######################
######
#Lag return
#####
newdata$lag.value <- c(NA, newdata$ret[-nrow(newdata)])
newdata$lag.value[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value1 <- c(NA, newdata$lag.value[-nrow(newdata)])
newdata$lag.value1[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value2 <- c(NA, newdata$lag.value1[-nrow(newdata)])
newdata$lag.value2[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value3 <- c(NA, newdata$lag.value2[-nrow(newdata)])
newdata$lag.value3[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value4 <- c(NA, newdata$lag.value3[-nrow(newdata)])
newdata$lag.value4[which(!duplicated(newdata$cusip))] <- NA
```

```
newdata$lag.value5 <- c(NA, newdata$lag.value4[-nrow(newdata)])
newdata$lag.value5[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value6 <- c(NA, newdata$lag.value5[-nrow(newdata)])
newdata$lag.value6[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value7 <- c(NA, newdata$lag.value6[-nrow(newdata)])
newdata$lag.value7[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value8 <- c(NA, newdata$lag.value7[-nrow(newdata)])
newdata$lag.value8[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value9 <- c(NA, newdata$lag.value8[-nrow(newdata)])
newdata$lag.value9[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value10 <- c(NA, newdata$lag.value9[-nrow(newdata)])
newdata$lag.value10[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value11 <- c(NA, newdata$lag.value10[-nrow(newdata)])
newdata$lag.value11[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.value12 <- c(NA, newdata$lag.value11[-nrow(newdata)])
newdata$lag.value12[which(!duplicated(newdata$cusip))] <- NA


####
#Lag volume
###
newdata$lag.vol <- c(NA, newdata$vol[-nrow(newdata)])
newdata$lag.vol[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol1 <- c(NA, newdata$lag.vol[-nrow(newdata)])
newdata$lag.vol1[which(!duplicated(newdata$cusip))] <- NA
```

```
newdata$lag.vol2 <- c(NA, newdata$lag.vol1[-nrow(newdata)])
newdata$lag.vol2[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol3 <- c(NA, newdata$lag.vol2[-nrow(newdata)])
newdata$lag.vol3[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol4 <- c(NA, newdata$lag.vol3[-nrow(newdata)])
newdata$lag.vol4[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol5 <- c(NA, newdata$lag.vol4[-nrow(newdata)])
newdata$lag.vol5[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol6 <- c(NA, newdata$lag.vol5[-nrow(newdata)])
newdata$lag.vol6[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol7 <- c(NA, newdata$lag.vol6[-nrow(newdata)])
newdata$lag.vol7[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol8 <- c(NA, newdata$lag.vol7[-nrow(newdata)])
newdata$lag.vol8[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol9 <- c(NA, newdata$lag.vol8[-nrow(newdata)])
newdata$lag.vol9[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol10 <- c(NA, newdata$lag.vol9[-nrow(newdata)])
newdata$lag.vol10[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol11 <- c(NA, newdata$lag.vol10[-nrow(newdata)])
newdata$lag.vol11[which(!duplicated(newdata$cusip))] <- NA

newdata$lag.vol12 <- c(NA, newdata$lag.vol11[-nrow(newdata)])
newdata$lag.vol12[which(!duplicated(newdata$cusip))] <- NA
```

```
test <-  newdata[complete.cases(newdata), ]
test$id <- 0
test$id <- ifelse(test$ret > 0 ,1,0)

test$idsvm <- -1
test$idsvm <- ifelse(test$ret > 0,1,-1)
test$idsvm <- as.factor(test$idsvm)



test$b <- 0
test$b <-ifelse(test$ret > 0,1,0)
test$b<- as.numeric(test$b)
save(test,file = "lagmsd.Rda")



data
}

df <- data.frame(a = 1:10, b = c(-3:-1,-4, -15:-10), c = c(-1,-3, 3:10))
#This will return you a boolean vector of which rows have negative values:
pred <- numeric(0)
for(k in 1:nrow(df))
{
  a <- ifelse(sum(df[k,] > 0 ) -  sum(df[k,] < 0) > 0,1,0)
  pred <- rbind(pred, a)

}
```

Bibliography

[1] Ernest P. Chan, *Machine trading: Deploying computer algorithms to conquer the markets*, 1 ed., Wiley.

[2] David Enke and Suraphan Thawornwong, *The use of data mining and neural networks for forecasting stock market returns*, Expert Systems with Applications **29**, no. 4, 927–940.

[3] Ludwig Fahrmeir, *Regression: models, methods and applications*, Springer.

[4] Yoav Freund, *Boosting a weak learning algorithm by majority*, Information and Computation **121**, no. 2, 256–285.

[5] Jerome Friedman, *Regularized discriminant analysis*, Journal of the American Statistical Association **84**, no. 405, 165–175.

[6] Trevor Hastie and Robert Tibshirani, *Discriminant analysis by gaussian mixtures*, Journal of the Royal Statistical Society: Series B (Methodological) **58**, no. 1, 155–176.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The elements of statistical learning: Data mining, infernece, and prediction*, 2 ed., Springer US.

[8] Trevor Hastie, Robert Tibshirani, and Jerome. Friedman, *Introduction to statistical learning with applications in r*, Springer US.

[9] Michael C Jensen, *Some anomalous evidence regarding market efficiency*, Journal of Financial Economics **6**, no. 2, 10.

[10] Breiman Leo, *Bagging predictors*, Machine Learning **24**, no. 2, 123–140.

[11] Breiman Leo., *Random forest*, Machine Learning **45**, no. 1, 5–32.

[12] Wei-Yin Loh, *Classification and regression trees*.

[13] Marcos Lopez De Prado, *Advances in financial machine learning*, vol. 1, Wiley.

[14] Burton G Malkiel, *The efficient market hypothesis and its critics*, The Journal of Economic Perspectives **17**, no. 1, 59–82.

[15] Burton G. Malkiel and Eugene F. Fama, *Efficient capital markets:a review of theory and empirical work**, The Journal of Finance **25**, no. 2, 383–417.

[16] Bob Mather, *Machine learning in finance: Use machine learning techniques for day trading and value trading in the stock market*, 1 ed., Abiprod Pty Limited.

[17] Glenn V. Ostir and Tatsuo Uchida, *Logistic regression: A nontechnical review*, American Journal of Physical Medicine & Rehabilitation **79**, no. 6, 565–572.

[18] David E. Rapach and Mark E. Wohar, *In-sample vs. out-of-sample tests of stock return predictability in the context of data mining*, Journal of Empirical Finance **13**, no. 2, 231–247.

[19] Allan Timmermann and Clive W.J. Granger, *Efficient market hypothesis and forecasting*, International Journal of Forecasting **20**, no. 1, 15–27.