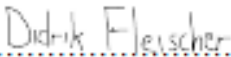




Universitetet
i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Industrial Economics Specialisation: Investment and Finance/Computer Science	Spring / Autumn semester, 2019.. Open/ Confidential
Author: Didrik Nyhus Fleischer	 (signature of author)
Programme coordinator: Supervisor(s): Roy Endre Dahl	
Title of master's thesis: Automated Trading System using Machine Learning	
Credits: 30 sp	
Keywords: - Automated Trading Systems - Backtesting - Machine Learning - Financial Machine Learning - Equity Risk Premium Prediction - Dataset Development	Number of pages: 83 + supplemental material/other: 12 Stavanger, 15. June 2019 date/year

Preface

This thesis concludes my master's degree in Industrial Economics, carried out at the University of Stavanger (UiS). My degree primarily involved studies in investing and finance but also computer science. Much of what drew me towards finance was a fascination for the stock market. I always wanted to learn more about investing and how I could best manage my own money, but the tedious work of studying financial statement and keeping up with the latest news did not appeal to me. This project has convinced me of taking a different approach to investing; a data-driven approach, where machine learning and artificial intelligence is used to automate the trading process.

My experience was limited in machine learning, empirical asset pricing, automated trading systems, backtesting, and Python programming at the onset of the project. It has been a tremendous challenge and memorable learning experience to produce the project's software, results, and report.

I would like to thank my supervisor, associate professor Roy Endre Dahl, who provided me with constructive feedback and useful comments.

Oslo 15.06.2019

Didrik Nyhus Fleischer

Abstract

This thesis investigates how machine learning can be applied in automated trading systems. To this end, an automated trading system driven by machine learning algorithms is developed. The system's design is inspired by techniques presented in "Advances in Financial Machine Learning" by Marco Lopez De Prado (2018). The automated trading system trades based on the predictions made by two random forest classification models. One model to set the side of trades and one to set the size of trades. The automated trading system was tested using a custom backtester, built to simulate real market conditions. The automated trading system was able to outperform the S&P500 index over the 7-year test period from 2012 to 2019 in terms of risk-adjusted return measured by a skew and kurtosis adjusted Sharpe ratio. Compared to a randomly trading model, the system seems to express stock picking ability that significantly exceeds random selection.

Also, machine learning is applied to the canonical problem of equity risk premium prediction. Several popular machine learning algorithms are used to regress 86 predictor variables from the literature on monthly equity risk premiums. The algorithms include principal component regression, random forests, and deep neural networks. Out-of-sample R^2 is used to measure performance. The results indicate that more advanced machine learning algorithms, allowing complex and non-linear interactions among predictors are more successful than linear regressions at modeling equity risk premiums. However, overall, the results are rather unimpressive and fail to demonstrate the added benefit of more complex ML algorithms convincingly. A random forest classifier for equity risk premium signs obtained better results, yielding an accuracy of 53.33% on out-of-sample data, which statistical tests confirm, with high statistical significance, is an accuracy unlikely to be produced by a random model.

All machine learning models were trained on a dataset developed specifically for this project, which includes 86 predictor variables from the literature. The dataset is based on company fundamental data as well as price, volume, and dividend data spanning over 20 years of history from 1998 to 2019 and includes over 14000 U.S. companies.

Table of Contents

Preface.....	i
Abstract	ii
List of Figures.....	vi
List of Tables.....	vii
1 Introduction.....	1
1.1 Objectives and Scope	2
1.2 Methodology	2
1.3 Main Empirical Findings	3
1.4 Thesis overview	4
2 Theory.....	5
2.1 Introduction to Stock Return Prediction	5
2.2 Predictive features	6
2.2.1 Price Momentum features	6
2.2.2 Value-versus-growth Features	7
2.2.3 Investment Features.....	7
2.2.4 Profitability Features	7
2.2.5 Intangibles Features	7
2.2.6 Trading Friction Features.....	8
2.3 Machine Learning.....	8
2.3.1 Classification vs. Regression Algorithms	9
2.3.2 Supervised vs. Unsupervised Learning	9
2.3.3 Online vs. Batch Learning	9
2.3.4 Instance-Based vs. Model-Based Learning.....	10
2.3.5 The General Structure of a Machine Learning Project.....	10
2.3.6 Performance Evaluation for Binary Classifiers	11
2.3.7 Performance Evaluation for Regressions	13
2.3.8 The Bias-Variance Tradeoff	15
2.4 Machine Learning Algorithms	15
2.4.1 Multiple Linear Regression.....	15
2.4.2 Principal Component Analysis and Principal Component Regression.....	16
2.4.3 Decision Trees	17
2.4.4 Random Forests.....	18
2.4.5 Artificial Neural Networks	19
2.5 Financial Machine Learning.....	22
2.5.1 Challenges of applying ML to Financial Data.....	22

2.5.2 Producing a Structured Dataset	24
2.5.3 Labeling Financial Data.....	26
2.5.4 Sample Weights.....	27
2.5.5 Why Ensemble Methods and Random Forests are well suited for Finance.....	27
2.5.6 Cross-Validation in Finance	29
2.6 Why Apply Machine Learning to Stock Return Prediction	31
2.7 Automated Trading Systems	31
2.8 Backtesting	32
2.8.1 Avoiding Overfitting	34
2.8.2 Backtesting Paradigms	34
2.8.3 Risk-Adjusted Performance Measurement	35
2.8.4 Backtest Statistics.....	36
2.8.5 Benchmarks	38
3 Methodology	39
3.1 Task 1: Dataset Development	39
3.1.1 Feature Calculations.....	40
3.1.2 Sampling	41
3.1.3 Labeling	42
3.2 Task 2: Stock return prediction	45
3.2.1 Sample Splitting and Cross-Validation	46
3.2.2 Performance Evaluation	46
3.2.3 Multiple Linear Regression.....	46
3.2.4 Principal Component Regression	47
3.2.5 Random Forest Regression.....	47
3.2.6 Artificial Neural Network Regression	48
3.2.7 Random Forest Classifier for Equity Risk Premiums.....	49
3.2.8 Statistical Significance of Model Accuracy	50
3.3 Task 3: Developing an Automated Trading System.....	51
3.3.1 Developing ML Models.....	52
3.3.2 Signal Generation Algorithm	52
3.3.3 Order Generation Algorithm	53
3.4 Task 4: Backtesting	55
3.4.1 Statistical Significance of ATS Outperformance relative to the S&P500 Index.....	60
4 Results and Discussion	61
4.1 An Empirical Study of US Equity Returns	61
4.1.1 In-Sample Results	61

4.1.2 Out-of-Sample Regression Model Results.....	64
4.1.3 Random Forest ERP Classification and Statistical Significance of Model Accuracy	65
4.1.4 Would re-fitting the model over the testing period yield better results?	66
4.1.4 Note on Feature Importance – RF ERP Regressor and Classifier.....	66
4.2 Automated Trading System Performance	69
4.2.1 Side and Certainty Classifier Results	69
4.2.2 Backtest Results of the Machine Learning Driven ATS.....	70
4.2.3 Backtest Results of a Randomly Trading ATS	75
4.2.4 Note on the Primary Research Question.....	78
4.2.5 Note on Risk and Risk Adjusted Performance	78
4.2.6 Note on Statistical Significance of ATS Outperformance	79
5 Conclusion	80
6 Further Work	81
7 References	82
8 Appendices	i
8.1 Appendix A – Dataset Features	i

List of Figures

Figure 1: Basic structure and notation of a deep neural network	20
Figure 2: Triple Barrier Method for Labeling Financial Data	27
Figure 3: Overlap and embargo for splitting training and testing data (López de Prado, 2018, chapter 7)	30
Figure 4: Sampling Scheme Illustration	42
Figure 5: Triple Barrier Method (López de Prado, 2018, chapter 3)	43
Figure 6: ATS ML model training procedure using the triple barrier method.....	44
Figure 7: Automated Trading System Architecture.....	51
Figure 8: Backtester components and order of operations	56
Figure 9: Trade Exit Policy Visualized	59
Figure 10: Relative Feature Importance - Random Forest ERP Regression Model	67
Figure 11: Relative Feature Importance - Random Forest ERP Sign Classification Model.....	68
Figure 12: Monthly Returns of ML Strategy and S&P500 Index (Configuration A).....	72
Figure 13: Total Return Relative to Initial Value of ML-driven ATS (Portfolio) and S&P500 Index (Configuration A)	72
Figure 14: Total Return Relative to Initial Value of ML-driven ATS (Portfolio) and S&P500 Index (Configuration B)	73
Figure 15: Total Return Relative to Initial Value of ML-driven ATS (Portfolio) and S&P500 Index (Configuration C)	73
Figure 16: Allocation of value between different accounts – ML-driven ATS under configuration A (end of day values)	74
Figure 17: Histogram of Monthly Returns for ML-dirven ATS and S&P500 Index (Configuration A)....	74
Figure 18: Example of Fund Allocation Between Concurrent Trades – ML-driven ATS (Configuration A)	74
Figure 19: Allocation of value between different accounts - Randomly Trading ATS under configuration A (end of day values)	76
Figure 20: Total Return Relative to Initial Value of Randomly Trading ATS (Portfolio) and S&P500 Index (Configuration A)	76
Figure 21: Total Return Relative to Initial Value of Randomly Trading ATS (Portfolio) and S&P500 Index (Configuration B)	76
Figure 22: Total Return Relative to Initial Value of Randomly Trading ATS (Portfolio) and S&P500 Index (Configuration C)	77
Figure 23: Histogram of Monthly Returns for the Randomly Trading ATS and S&P500 Index (Configuration A).....	77
Figure 24: Monthly Returns of Randomly Trading ATS and S&P500 Index (Configuration A).....	77

List of Tables

Table 1: Types of information relevant to financial modeling (López de Prado, 2018)	24
Table 2: Out-of-sample regression results for the historical and zero mean forecast models:.....	61
Table 3: Principal Components - Variance Ratios	61
Table 4: In-sample results for multiple linear regression and principal component regression	61
Table 5: Random Forest Regression Model - Best Hyperparameters	62
Table 6: Random Forest Regression Model - Cross-validation results given best hyperparameters ...	62
Table 7: Deep Neural Net Regression Model - Best Hyper Parameters.....	63
Table 8: Deep Neural Net Regression Model - Cross-validation results given best hyperparameters .	63
Table 9: Out-of-Sample ERP Regression Results	64
Table 10: Random Forest ERP Classifier - Best Hyperparameters	65
Table 11: Random Forest ERP Classifier - Cross-validation results given best hyperparameters.....	65
Table 12: Out-of-Sample ERP Sign Classification Results	65
Table 13: T-tests of Random Forest ERP Sign Classifier Accuracy.....	66
Table 14: Out-of-Sample Results of RF Trained on Extended Training Set	66
Table 15: Side Classifier - Best Hyperparameters	69
Table 16: Side Classifier - Cross-validation results given best hyperparameters.....	69
Table 17: Out-of-Sample Side and Certainty Classifier Results	69
Table 18: Automated Trading System Configurations	70
Table 19: Backtest Results for ML driven Automated Trading System	70
Table 20: Backtest Results for Random Automated Trading System.....	75
Table 21: Information about Dataset Features	ii

1 Introduction

Machine learning has experienced a substantial rise in popularity in recent years. ML algorithms are today capable of performing many tasks that only expert humans were capable of a few years back. Substantial advances have been made in computer vision, natural language processing, and many other areas through the application of ML algorithms. With large opensource projects like TensorFlow and Scikit-Learn, advances in hardware technologies and the commoditization of computational resources through public clouds, ML technologies are more available than ever before. As it relates to finance, ML will transform how we invest, issue loans, detect fraud, and much more.

This thesis explores how machine learning can be applied in automated trading systems to the end of generating risk-adjusted returns. Automated trading systems are computer programs that autonomously creates and submits orders to market exchanges. They use algorithms to carry out an investment strategy on behalf of the investor. This type of investing has seen a rise in recent years for both investment firms and retail investors. There are many reasons for this trend; accessible online brokers, reduced trading costs, increased data availability through online platforms, and open source technology.

Automated trading systems have historically been used a lot in the space of technical analysis. Where the system conducts trades based on detection of trading signals, e.g., moving averages or Bollinger band breakouts, other systems use financial models to make predictions on price developments, which later are used to make trading decisions.

This is where machine learning comes into the picture. ML algorithms are especially suited for prediction tasks. The problem is that financial markets are very complex systems, and random and unpredictable events largely drive their development. Price prediction is therefore notoriously tricky. The best functional form for a predictive model and what parameters to use are largely unknown. Further, the relevant parameter space is likely vast. The academic literature has documented the predictive power of several hundred firm-specific variables and tens of macroeconomic variables. Also, it is likely essential for modeling to allow predictors to interact in complex ways, which further increase the space of possible model specifications. Applying ML can be a solution as it contains a diverse set of algorithms, which enables exploration of an extensive set of functional forms using high dimensional data.

Machine learning is the scientific study of algorithms that can learn automatically from data to perform tasks without using explicit instructions. An ML algorithm can detect deep patterns in high-dimensional data and use this insight to make predictions on data it has not seen before. But machine learning techniques often do not translate directly when applied to financial datasets. One can argue that financial machine learning is best conceptualized as a distinct subject, different from standard ML. A few examples of what makes financial ML different are:

- Financial data has high levels of noise making it easy to produce overfit models
- Problems in finance often involve predicting future developments from historical data, but exact historic conditions are unlikely to repeat themselves.
- Financial data often have variable information arrival rates. Causing some data samples to be more informative than others.
- The economy exhibits regime switches, which makes modeling difficult due to model and variable instability

- Financial data often exhibit bad statistical properties, like non-stationarity with long memory, serial and cross-sectional dependence, and observations are often non-IID and non-normal.

1.1 Objectives and Scope

This project investigates how financial ML can be used in the development of automated trading systems and how such systems should be backtested. The project's research question is:

“How can machine learning be applied in automated trading systems to generate risk-adjusted returns?”

To answer this question, the project aims to develop an automated trading system, where ML algorithms to inform trading decisions. The automated trading system's performance will be measured by a backtester, which is built to mimic real market conditions by accounting for fees, commissions, slippage, bankruptcies, dividends, and more.

A secondary aim of the project is to see if advanced ML algorithms can outperform traditional tools like linear regressions on the canonical problem of equity risk premium prediction. To this end, regressions are performed on monthly equity risk premiums of individual stocks using several popular ML algorithms.

Secondary research questions this thesis addresses are:

- How to develop datasets from unstructured financial data for ML model training?
- What ML algorithms and techniques are well suited for problems in finance?
- What challenges are involved in the development of automated trading systems?
- What challenges are involved in the development and execution of backtests?
- How well do the regression models perform in terms of out-of-sample R^2 compared to a historical average forecast model?

The project has a strong practical tilt, where most of the presented techniques have also been implemented in software. The experienced academic might miss some depth in the descriptions of the economic theory underpinning the project, but practitioners will be happy to see rich descriptions of the system's implementation and discussions around its design.

1.2 Methodology

To answer the research questions, the project's methodology is divided into four tasks: dataset development, equity risk premium prediction, development of an ATS, and backtesting.

Task 1: Dataset Development

Price, dividend, and volume data, as well as company fundamentals spanning 20 years and 14,000 companies, are used to develop a dataset of 86 predictive features from the stock factor literature. The data spans the period from January 1998 to February 2019. The dataset is divided into a training and testing set. The training set spans the period 1998-01-01 to 2012-01-01, and the testing set spans the period 2012-03-01 to 2019-02-01.

Task 2: Equity Risk Premium Prediction

Equity risk premium predictions are performed on monthly risk premiums of individual stocks using multiple linear regression, principal component regression, random forests, and artificial neural networks. The regression problem is defined such that a single model is developed for the entire panel of equities, independent of time. The out-of-sample predictive power of the regression models are measured using out-of-sample R^2 , mean squared error, and mean absolute error.

In addition to the regression models, a random forest classifier for the sign of monthly equity risk

premiums is developed. T-tests are conducted to see if the accuracy of the classifier is higher than that of a random model with an accuracy of 0.5 at a significance level of 5%.

Task 3: Development of an ATS

The ATS developed for the project use predictions made by ML models to decide what stocks to buy/sell and how much capital to allocate to each trade. In short, two random forest classification models are built. The first model is trained to predict the direction of future stock returns, and the second model is trained to predict when the first model is correct in its prediction. The first model is used to set the side of trades, and the second model is used to derive the size of trades. In addition to having trading decisions informed by ML models, the ATS's behavior is restricted in various ways to mitigate excessive risk-taking and trading costs. The goal of the ATS is to extract as much predictive power as possible from the ML models and translate this into risk-adjusted returns.

Task 4: Backtesting

A backtester is developed from scratch to measure the performance of the ATS. The backtester's primary job is to simulate the execution of orders, manage active trades, and keep track of returns and costs. Once an order is successfully processed, the backtester keeps track of price developments and automatically exits the position if the trade's stop-loss, take-profit or timeout limits are reached. Further, the backtester is built to handle delistings, bankruptcies, dividend payments, and interests. The backtester also supports short trades, which involve maintenance of a margin account. After completing a simulation, the backtester calculates several statistics to give insight into how the ATS performed over the backtested period. To assess risk-adjusted performance, a skew and kurtosis adjusted Sharpe ratio is calculated. Further, t-tests are conducted to assess if the ATS's monthly excess returns relative to the S&P500 index have a mean that exceeds zero.

1.3 Main Empirical Findings

The following are the main empirical findings:

- More advanced ML models, like random forests and deep neural nets, seem to be better suited for modeling equity risk premium compared to linear regression and principal component regression.
- Regressing predictor variables on monthly equity risk premiums fail to clearly demonstrate the added benefit of more complex ML algorithms compared to a historical mean forecast or linear models.
- Random forest classification model for the sign of monthly equity risk premiums yielded an accuracy that exceeded that of a random model with high statistical significance.
- The project's machine learning driven automated trading system was able to outperform the S&P500 index in terms of return and adjusted Sharpe ratio on out-of-sample data.
- Compared to a randomly trading ATS, the machine learning driven ATS showed stock picking ability that greatly exceeded random selection.

1.4 Thesis overview

Chapter 1: Introduction

This chapter introduces the thesis topics, outlines the scope of the project, summarizes the main empirical findings, and gives an overview of the report.

Chapter 2: Theory

This chapter goes through the theory on which the project relies — starting with a short introduction to stock return prediction and taxonomy of predictor variables. Then, continuing with an introduction of machine learning and some of the field's fundamental concepts and terminology. After introducing machine learning, comes a description of each machine learning algorithm used in this thesis. Ensuing is a particularly important chapter on machine learning applied to problems in finance. Here the challenges of applying ML in finance is discussed together with how the many pitfalls can be avoided. The chapter ends with a short description of automated trading systems and how to backtest such systems.

Chapter 3: Methodology

This chapter goes through the project's methodology, which is divided into four main tasks: dataset development, equity risk premium prediction, development of an ATS, and backtesting. This chapter includes implementation details regarding how the dataset, ML models, automated trading system, and backtester was developed.

Chapter 4: Results and Discussion

This chapter goes over the results from:

- Running regressions on monthly equity risk premium using various ML models
- Testing a classifier for equity risk premium signs
- Testing the ML models used by the project's automated trading system
- Backtesting the project's ML driven ATS
- Backtesting a randomly trading ATS

The chapter also contains interpretations and discussions of the results in the context of the project's objectives and scope.

Chapter 5: Conclusion

This chapter stats concluding remarks.

Chapter 6: Further Work

This chapter goes over some directions the project could take going forward.

2 Theory

2.1 Introduction to Stock Return Prediction

This paper is concerned with predicting returns of individual stocks and the application of these forecasts in automated trading systems. To this end, literature related to the prediction of equity risk premiums is of interest.

For a long time, the popular opinion was that stock returns were entirely unpredictable. The canonical random walk model was used extensively by many modelers and was popularized to the general public by Malkiel in his book “A Random Walk Down Wall Street” (1973). Towards the end of the 20th-century economists started to take the view that aggregate stock returns do contain a predictable component. There are considerable amounts of in-sample evidence in support of predictability of aggregate US stock market returns using several economic variables. There is, however, much less support for out-of-sample predictability. Goyal and Welch (2003, 2008) found the predictive power of several popular economic variables do not hold up in out-of-sample tests. (Campbell and Thompson, 2008)

Over the years, much research has been devoted to finding variables with predictive or explanatory power over asset prices. Hundreds of predictive variables have been published in academic journals. However, the wealth of factors is likely not only a product of sound scientific research. A sizable proportion is probably attributable to data mining. An important debate in the return anomalies literature is whether the reported abnormal returns are compensation for systematic risk, evidence of market inefficiency, or merely a result of extensive data mining (Yan and Zheng, 2017). Many researchers use largely overlapping data, and the more scrutiny a set of data is subjected to, the more likely it is that spurious patterns will emerge.

Despite the rich literature on return anomalies, Goyal and Welch (2008) and others argue that the historical average excess return forecasts’ future stock returns better than regressions on predictor variables. However large the predictable component of excess stock returns is, it is likely very small. (Campbell and Thompson, 2008)

The literature on stock return prediction comes in two basic strands. The first models’ differences in expected returns across different stocks as a function of firm-level characteristics. These types of models are exemplified by Fama and French (2008) and Lewellen (2015). The general approach here is to run cross-sectional regression on a small number of lagged stock characteristics/features. The other strand of models forecast the time-series of returns. Here the general approach is to conduct time-series regressions of broad portfolio returns on a few macroeconomic variables (Gu, Kelly and Xiu, 2018). These traditional strategies are potentially severely limited. One such limitation is the use of linear regressions, which is not suited when applying many predictors in the model. With documented predictors numbering in the hundreds, more advanced statistical techniques used in machine learning is likely better tools for the job.

Stock return prediction is a very difficult problem. There are many reasons for this, most notably; the problem contains a large unpredictable component, meaning that even the best forecasting models will explain only a small part of stock returns. The efficient market hypothesis (EMH) is often brought up as an argument against price predictability. The EMH postulates that our financial markets are efficient in the sense that all available information is already incorporated into prices. As a result, only unpredictable components govern future price developments. The famous result of this rationale is that stock prices are best modeled as Brownian motion.

According to popular theoretical models, such as the discounted cash flow model or the dividend discount model, stock prices are directly influenced by expected future cash flows and the risks involved in obtaining these cash flows. Thus, theoretically, equity returns must be a function of the characteristics of the real economy, where businesses and customers meet to exchange goods and services for payment. Even with this insight, the problem is no more tractable. The economy displays significant business-cycle fluctuations, erratic and unpredictable developments, and different variables affect the different segments of the economy differently. There is no single model to accurately explain returns for all assets. Further, parameter importance can change over time.

As demonstrated by Pesaran and Timmerman (1995), model uncertainty and parameter instability are important problems to solve when building stock return models. Model uncertainty refers to the fact that researchers do not know the best model specification or the values of its parameters. Parameter instability means that as the economy changes and business cycle fluctuations manifest, the best model specification often changes. During a recession, some variables might be more predictive than they were during the preceding bubble. In effect, modeling out-of-sample returns consistently well are very hard.

2.2 Predictive features

Stock return prediction relies on the presence of predictive signals. This project primarily uses pricing, volume, and dividend data in addition to company fundamental data from 10-K and 10-Q filings. 10-K and 10-Q filings allow investors and researchers to gain insight into the financial standing of a firm and track developments over time. These types of filings are the products of accounting and are therefore only backward looking. An assumption made when using this type of data in prediction models is that past development has a bearing on future performance. Further, this type of data is well known to professional investors. Many of the features were first introduced many years ago, in some cases; decades. This draws into question what information can be extracted from such data which is not already reflected in market prices shortly after their publication. Mclean and Pontiff did research suggesting that investors learn about mispricing from academic publications and that publication-informed trading yields lower returns (McLean & Pontiff, 2016, p. 1-4). Despite these drawdowns, this project uses this type of data due to its accessibility, the inaccessibility of other types of data (e.g., insider information and market microstructure data, earnings forecasts), and it also allows comparison with other research.

This paper follows the same taxonomy for predictive features as used by Hou, Xue, and Zhang (2017), which divide features into the following categories: momentum, value-versus-growth, investment, profitability, intangibles, and trading frictions. Descriptions of each feature included in the project's dataset are not provided here. The reader is referred to Appendix A, where the full list of the features is given, including references to the original papers, variable descriptions, and implementation formulas.

2.2.1 Price Momentum features

Price momentum features capture past price developments over different time horizons. For example, the one-month momentum is defined as the percentage return over the past month. Returns often have the risk-free rate subtracted, yielding the return in excess of the risk-free rate. One of the rationales behind the use of momentum features is that investors overreact to new information and push prices away from the true values of the underlying assets. This suggests that buying past losers and selling past winners will achieve abnormal returns. Although this strategy has empirical evidence behind it, it is debated what mechanism is at play. Many hold the view that investor overreaction is only part of the explanation. Other hypotheses with regards to momentum

features are that over shorter time scales, prices exhibit mean reversion, while over longer time scales (one year or more) they exhibit momentum behavior. Mean reversion refers to prices tendency to revert towards the average price over time. Momentum behavior refers to rising prices tendency to continue rising, and falling prices tendency to continue falling. The project's dataset contains multiple momentum features with different time horizons (1, 6, 12, and 24 months). In the paper "Empirical Asset Pricing via Machine Learning" (Gu, Kelly and Xiu, 2018) which ran regressions on monthly equity risk premiums with a variety of machine learning algorithms found momentum features (1-month momentum, 12-month momentum) to be some of the most predictive features in their dataset, consisting of over 100 predictors from the literature.

2.2.2 Value-versus-growth Features

Growth and value investing are two fundamental approaches to investing. A growth investor looks for strong earnings growth and high future potential. A value investor looks for companies with solid fundamentals that are undervalued in the current market. There is a continuous debate over which style is superior, but the truth is likely to be a nuanced one, with one style outperforming the other under different market conditions.

Value-versus-growth features capture company characteristics related to its financial strength or growth potential. They are often expressed as a ratio of accounting variables. Examples of value-vs-growth features are book-to-market, quick-ratio, sales-to-receivables, and earnings-to-price. All these features are part of this project's dataset.

Due to the different nature of different industries, it is often more meaningful to compare these types of ratios to other companies in the same industry. Another way to conduct meaningful comparisons is to put companies on a leveled playing field by adjusting the variables by industry mean values.

2.2.3 Investment Features

Investment features capture the level of capital investment and relate this to a variety of other company fundamentals. A high investment level indicates financial strength and availability of profitable investment opportunities. Examples of investment features are capital investment, inventory change, change in PP&E over sales, and change in common equity.

2.2.4 Profitability Features

Profitability features capture information relating to the company's level of profitability as well as its strength and growth in earnings. High returns/earnings and growth in returns/earnings are positive signs and warrant higher stock prices. The earnings/returns can also be related to other company characteristics. For example, return on assets relates returns to the value of the assets needed to generate those returns. Some industries require much less upfront investment than others to generate the same level of returns.

Some examples of profitability features in this project's dataset are cash-flow-to-debt, return on invested capital, revenue surprise, earnings surprise, and sales growth.

2.2.5 Intangibles Features

Not all the valuable assets of a company are physical. A famous brand, patents, proprietary technology developed through in-house R&D, and good corporate governance can hold significant economic value. Intangibles features attempt to quantify the value of the intangible assets of a company. A popular metric to use is research and development expenses and its relation to other firm fundamentals. Other types of intangibles features are advertising expense to market value, firm age, analyst coverage, and pension funding rate.

2.2.6 Trading Friction Features

Trading frictions refer to the costs and risks of trading individual assets or the markets in general. Different liquidity and volatility measures are the most typical features seen in this category. The idea is that low liquidity stocks are more difficult to buy and sell because it is more difficult to find a party to take the opposite side of the trade. This increases the risk associated with entering the trade as it may be difficult to exit in a timely fashion.

High volatility is also associated with risk, because prices may move significantly and unpredictably over short periods. Riskier stocks should have greater potential for higher returns than less risky stocks to compensate the investor. Some examples of trading friction features are the market beta, stock beta, standard deviation of returns, volatility of liquidity, and return over volume.

2.3 Machine Learning

This section introduces fundamental concepts, theory, and terminology of machine learning.

Starting with a definition of machine learning,

A computer is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E .

- (Tom Mitchell, 1997)

This definition captures the fundamental characteristics of ML. ML is to generalize experience from training data and use this generalization to make predictions on data it has not seen before. What makes machine learning different from other optimization problems is that many ML algorithms do not have a well-defined function to be optimized, making the model specification more flexible.

The goal of ML is to be able to make accurate predictions on out-of-sample-data. If it fails to generalize properly, out-of-sample performance will suffer. When a machine learning algorithm fails to generalize, it is often due to overfitting or underfitting. Overfitting means that the ML algorithm ended up learning too much of the idiosyncratic and random patterns of the training set, which does not persist to other samples drawn from the process under study. Underfitting is the opposite of overfitting. The ML algorithms learn too few patterns to make good predictions on either the test or training sets.

ML is a great tool when modeling something where the appropriate model specification is not known. Take the example of recognizing faces in images. Precisely what allows us to recognize that an entity as a face? Is it the roundness of the head? Is it the relationship between the eyes and nose? Even if we knew exactly how faces are recognized, how should the computer be programmed to perform the task? What if the person is upside-down, or turned to the side, will our rules hold up, or must they be adjusted? A task incredibly trivial for us humans are near impossible to express in explicit instructions a computer can follow. There are many other such problems. Stock return prediction is one of them. What features of a company and the markets yields the most accurate model for future stock returns? The answer is; we do not know. Machine learning is an incredibly powerful tool in these cases because it allows the discovery of complex relationships between predictors and target values. Relationships that would often be overlooked if simpler models were applied.

ML is sometimes accused of being black boxes. This is because some types of models (e.g., deep neural networks (DNN)) are quite hard to reason about. Exactly what information the DNN has picked up on during training is often hard to say, especially for more extensive networks. However,

other algorithms like decision trees are much easier to reason about.

There is some truth to the black box analogy, but there are ways to gain strong intuition for how most models work. This makes ML an important research tool which can serve as a catalyst for new insights.

ML algorithms can be categorized in several ways. Understanding what these different classes of algorithms are, gives a broad overview of the field.

2.3.1 Classification vs. Regression Algorithms

One of the most fundamental distinctions made between ML tasks is whether they are a classification or regression task. In classification, the aim is to find an approximate mapping function from the input data to a *discrete* output variable. An example is to classify the direction of future price movements of a stock. The observations are labeled with 1 if the stock increases in value and -1 if the stock decreases in value. In regression, the task is to find an approximate mapping function from the input data to a *continuous* output variable. In contrast to the previous example, to predict the value of monthly stock returns as a continuous percentage value is a regression problem.

This project produced both regression and classification models.

2.3.2 Supervised vs. Unsupervised Learning

This distinction refers to the amount and type of supervision algorithms have during training. Under supervised learning, both the feature data and the associated solutions are provided to the algorithm. During training, the algorithms try to infer the underlying relationship between the input data (features/predictors) and the target values (labels). The labels can be both discrete categories (the case in classification problems) or continuous values (the case in regression problems). Many algorithms can be used on both classification and regression problems, but not all. For example, linear regressions are ill-suited for classification tasks, while decisions trees can be used to solve both classification and regression tasks. (Khanna and Awad, 2015, chapter 1)

Unsupervised learning algorithms are built to discover hidden structures in unlabeled datasets. In other words; the desired output is unknown. The aim in this type of ML is to hypothesize representations of the input data that captures its structure and enables efficient decision making, forecasting, and information filtering. Two classic examples are clustering and dimensionality reduction algorithms. (Khanna and Awad, 2015, chapter 1)

Two other types of learning are semi-supervised learning and reinforcement learning. In semi-supervised learning, the dataset is only partially labeled, and the algorithm typically goes through steps of both unsupervised and supervised learning. In reinforcement learning, the algorithms are trained to learn the best policy for solving a task. The algorithm updates its policy during training by getting punished or rewarded by its environment. DeepMind's AlphaGo program employed reinforcement learning to beat the world champion Ke Jie at the game of Go. (Aurélien Géron, 2017, chapter 1)

This project only employs supervised learning.

2.3.3 Online vs. Batch Learning

This distinction refers to an algorithm's ability to learn incrementally from an ongoing stream of data or if all data must be available before training starts. Batch learning requires all data to be available when trained. This is a time and resource consuming task and is often done offline. Once the model is deployed, the model does not continue to learn. Therefore, it is necessary to train new models from scratch periodically as new data becomes available. This is a time and resource consuming

design, but the process can easily be automated, and if the computational resources are available, it is not a problem. On the other hand, when resources are scarce, online learning can be the solution. This class of algorithms can train incrementally by feeding it new data instances, either one by one or in small groups called mini-batches. This type of learning allows the system to adapt to changing environments in real time. (Aurélien Géron, 2017, chapter 1)

Only batch learning was employed in this project.

2.3.4 Instance-Based vs. Model-Based Learning

This distinction has to do with how the algorithm generalizes. In instance-based learning, the system learns all the examples provided and uses a similarity measure to compare new cases to the learned examples. The learned example that the new observation is most like decides its class/value. In model-based learning, one uses examples to build a model that best represents the data. This model is then used to make predictions on new observations. When running a linear regression, one tries to express the independent variable through a linear combination of the dependent variables. Finding the best fit between the independent and dependent variables is an example of model-based learning. (Aurélien Géron, 2017, chapter 1)

This project only applies model-based learning.

2.3.5 The General Structure of a Machine Learning Project

To further introduce the reader to the field of machine learning and provide context for the project's methodology, the typical steps of an ML project will be explained.

Before even starting the project, the project's participants must gain a good understanding of the problem they want to solve. Properly defining the problem is crucial for successful modeling. Important questions to answer at this early stage are: How to define the problem? How should model performance be measured? What assumptions can be afforded?

The next step is to acquire the data needed for model training, validation, and testing. The data can be anything, but it must be possible to represent it in a digital format as a set of numbers (discrete or continuous). When doing machine learning, the datasets necessarily become very large. This makes it near impossible to gain insights into its structure without building visualizations. Any insights gained through familiarizing oneself with the data will help in the later stages of the process.

Data coming directly from the source is often unsuited to pass to ML algorithms. Therefore, it is common to prepare the data for ML through various data transformations. For example, categorical data represented by strings in the dataset must be transformed into a numerical representation (e.g., using one-hot encoding). Another common transformation is feature scaling. Some algorithms have objective functions that will not work well when the provided data is scaled differently (e.g., clustering through Euclidian distance calculations). For example, stock momentum is expressed as a percentage, while market capitalization is expressed as a large dollar value. This leads to some features being weighted much higher than others during training due to their numeric representation, and not their informational content.

There are multiple techniques to achieve common scale for all features (e.g., standardization and normalization).

Often the data you have at your disposal has errors, missing values, extreme values, and outliers. This can adversely affect model training. Some algorithms do not even accept the presence of a single missing value. It is therefore common to drop examples with too many missing feature values or try to amend the missing data by substituting missing values by the column's mean value or

another statistic. By excluding outliers and extreme values, the model might generalize better as the influence of improbable observations are reduced. Improving data quality is essential to successful modeling.

Once data has been prepared, it is time to select a model and train it. Trying out multiple types of models and multiple configurations of each type are necessary to find the best solution. One of the reasons why we do machine learning is because we do not know what the model should look like. It is through trial and error that the model converges towards a better and better specification. The process of tweaking model parameters is called “hyperparameter tuning.” A hyperparameter might be the depth of a decision tree or the learning rate of an artificial neural network (ANN).

At this point, it is appropriate to introduce the difference between training, validation, and test sets. As mentioned earlier, the goal of machine learning is to generalize the model to perform well on out-of-sample data. This means that we must avoid overfitting the model to the training data. After having trained a model on the training set, we need to measure how the model performs on out-of-sample data. But at this point, we are not yet sure if the model's hyperparameters are optimally configured. In the process of tuning the hyperparameters, we need to perform multiple out-of-sample tests to see if the out-of-sample performance improves as a hyperparameter is changed. If we continually used the test set in this process, we would end up selecting the hyperparameters that optimize performance on the test set, but here is the problem; we don't want to optimize performance on the test set, we want to optimize performance on out-of-sample data in general. By continually using the test set during hyperparameter tuning, one ends up conflating the measure of out-of-sample performance, as the hyperparameters are selected to make the model perform well on the test set. Therefore, we introduce a separate validation set, that does not overlap with either the test or training set. Out-of-sample performance is gauged through testing on the validation set during hyperparameter tuning. When we are happy with the hyperparameters, a true out-of-sample test can be performed using the test set. These results will be representative of what performance can be expected on new data, as neither training or hyperparameter tuning has fitted the model to the test set.

At this point, the model is completed, and it is time to present the final solution. If all parties are satisfied, it is time to deploy. Once the model is deployed, the performance must be monitored, and the overall system maintained. As time goes, more data may become available the optimal model specification may change. It is therefore common to periodically re-train the model to maintain and improve system performance.

2.3.6 Performance Evaluation for Binary Classifiers

Performance evaluation is the process of measuring and assessing model performance. Exactly what the appropriate measurement formula is, depends on the type of model, and the goal of the ML project. Some of the most common statistics used in the ML literature will now be explained.

Accuracy, Precision, Recall, and Confusion Matrixes

When it comes to classification, this project is only concerned with binary classification problems. Therefore, the concepts of precision, recall, and confusion matrixes are only discussed in the context of binary classifiers, although the concepts extend to multi-class problems.

Measuring the performance of classifiers is often more complicated than measuring the performance of a regressor. It requires the developer to assess performance from multiple points of view. For example; if one of the classes has a much higher representation in the dataset (e.g., 85% are of the positive class) and the classifier achieves an accuracy of 85% on the positive class, this contains

no information about how well the classifier generalized. A classifier that blindly predicted the sample to always be of the positive class would also achieve an accuracy of 85%.

Accuracy is the number of correct predictions over the total number of predictions. Where TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives; accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Confusion matrixes are commonly employed to get a more nuanced view of a classifier's performance. A confusion matrix is a specific table layout that allows visualization of a classifier's performance. Each column in the confusion matrix represents the instances in the true classes, and each row represents the instances in the predicted classes. Each cell along the main diagonal contains the number of instances of each class that was predicted correctly. Any other cell will contain the number of misclassified instances. The misclassified instances were supposed to be classified according to the column label but were classified according to the row label. This makes it easy to see which classes are being confused with each other. A confusion matrix for a binary classifier takes the following form:

	Actual values		
Predicted Values		Positive	Negative
	Positive	TP	FP
	Negative	FN	TN

A more concise metric than the confusion matrix is the *precision* of a classifier. The precision is the accuracy of the positive predictions.

$$precision = \frac{TP}{TP + FP}$$

If all the prediction the classifier makes is true positives, the precision will be 1, which is a perfect score. But even if the classifier obtains a perfect precision, this metric will not inform about the positive instances that the classifier failed to recognize as positive. Therefore, the precision of the classifier is often presented together with its recall. Recall is the ratio of positive instances that were correctly detected by the classifier.

$$recall = \frac{TP}{TP + FN}$$

It depends on the problem at hand whether one should aim at high precision or recall. For some problems, it is more important to detect undesired instances than managing to detect all desired instances. When building a classifier for stock prices to make investment decisions, it might be desirable to maximize precision rather than recall, because it is important to mitigate losing trades rather than jump on every available opportunity.

There is a constant tradeoff between high precision and high recall, and the best classifier is often some compromise that optimizes the relationship between both precision and recall.

F₁ Score

By combining precision and recall into a single metric, one can get a more nuanced view of the classifier's performance. The F_1 -score is the harmonic mean of precision and recall. By applying the harmonic mean rather than the regular mean, lower values get much more weight. Therefore, both precision and recall must be high to obtain a high F_1 score.

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 * \frac{precision * recall}{precision + recall} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

(Aurélien Géron, 2017, chapter 3)

2.3.7 Performance Evaluation for Regressions

Mean Squared Error

A common measure is the Mean Square Error (MSE). MSE has been used extensively in the literature to measure the performance of predictive regression models for equity risk premiums.

$$MSE = \frac{1}{m} * \sum_{i=1}^m (h(X_i) - y_i)^2$$

Where m is the number of samples, h is the hypothesis, X_i is the feature vector for the i -th sample and y_i is the true value of the dependent variable the hypothesis is trying to predict. The MSE is the mean of the squares of the errors. MSE typically refers to an in-sample measure but can also be calculated for out-of-sample data. In this case, the MSE is commonly referred to as the mean squared prediction error (MSPE).

An MSE of zero means that the regression model fits the data perfectly, but this is generally not possible to achieve. On the other hand, a large MSE is undesirable as it indicates the regression model does not fit the data well. The MSE captures every deviation between the predicted values and the observed values. By squaring the deviations, both positive and negative deviations contribute equally. A potential downside of MSE is that large errors/residuals have a significant impact on the final value. Therefore, if MSE is used on data with many outliers, the MSE can get inflated and give a wrong impression. To avoid this downside, a popular alternative is the mean absolute error (MAE).

$$MAE = \frac{\sum_{i=1}^n |h(X_i) - y_i|}{n}$$

(Walpole, Myers, Myers and Ye, 2014)

R-Squared

Another popular measure of predictive performance is the R-squared (R^2).

$$R^2 = 1 - \frac{\sum_{i=1}^n (h(X_i) - y_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

Where \bar{y} is the mean of the true values of the dependent variable y , and the other symbols are defined as earlier. The R^2 has many different interpretations, depending on the context of its use. For example, when applied in the context of a simple linear regression, the R^2 is the square of the sample correlation coefficient. If the number of regressors exceeds one, the R^2 is the square of the coefficient of multiple correlation. The R^2 typically takes on values between 0 and 1, but it can also have negative values. Negative values can be interpreted as the mean of the actual observations being a better fit to the data than the fitted model (hypothesis). This can arise when the functional form of the model has been misspecified or inappropriate restrictions have been made on the regressors. The R^2 statistic is commonly used as a goodness of fit statistic. An R^2 of 1 indicates a perfect fit and an R^2 of 0 indicates no linear relationship.

Another way of expressing the R^2 is:

$$SS_{tot} = \sum_{i=1}^n (y_i - \bar{y})^2$$

$$SS_{reg} = \sum_{i=1}^n (h(X_i) - \bar{y})^2$$

$$SS_{res} = \sum_{i=1}^n (h(X_i) - y_i)^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

$$R^2 = 1 - FVU$$

Where SS_{tot} is the total sum of squares, SS_{reg} is the regression sum of squares, SS_{res} is the residual sum of squares and FVU is the fraction of variance unexplained. In the cases where $SS_{res} + SS_{reg} = SS_{tot}$ we have that:

$$R^2 = \frac{SS_{reg}}{SS_{tot}} = \frac{\frac{SS_{reg}}{n}}{\frac{SS_{tot}}{n}}$$

Through this definition, it is easy to see that an interpretation of R^2 is the amount of variance in the data that was explained by the regression model. It is important to note that the condition $SS_{res} + SS_{reg} = SS_{tot}$, does not always hold. It is therefore important not to blindly apply this interpretation. An example of when this condition does hold is when using the ordinary least squares method for fitting a simple/multiple linear regression. The R^2 's use goes beyond linear regressions. It can be used to assess any regression model, although its interpretation may not be as powerful as in the case of linear regressions.

A drawback of R^2 is that its value never decreases as more regressors are added to the model. Even the smallest linear relationship between a new regressor and the dependent variable (which could be a product of chance), will result in a better fit in terms of ordinary least squares and an increase in R^2 . If regressors are continually added to increase R^2 (e.g., using kitchen sink regression), it is advisable to correct for this effect. The adjusted R^2 can be applied in this case. The adjusted R^2 will penalize the addition of additional regressors, and is defined as follows:

$$R_{adj}^2 = 1 - (1 - R^2) \frac{n - 1}{n - p - 1}$$

Where n is the number of samples, and p is the number of regressors in the model.

When building models for forecasting, it is desirable to obtain the highest possible performance on out-of-sample (OOS) data. It is therefore common to use the out-of-sample R^2 , which has the same definition as earlier, but is restricted to only use OOS data.

$$R_{OOS}^2 = 1 - \frac{\sum_{i=1}^q (h(X_i) - y_i)^2}{\sum_{i=1}^q (y_i - \bar{y})^2}, q \text{ is the number of OOS observations}$$

The R^2 statistic is not indicative of the independent variables being causally linked to the dependent variable, the correct regression model being used, or enough data being used to make conclusive statements. (Walpole, Myers, Myers and Ye, 2014)

2.3.8 The Bias-Variance Tradeoff

There are generally three sources of errors when modeling a phenomenon with ML. These are bias, variance, and noise.

Bias refers to the error introduced by inaccurate assumptions. If the independent variables are not linearly related to the dependent variable, applying linear regression would result in high bias. The model would not be able to learn important patterns between features and outcomes. In this situation, the model is underfitting the training data.

Variance refers to the error caused by sensitivity to small variations in the training data. When variance is high, the model is prone to fit random noise in the training set, rather than learning general and persistent patterns. The result is often an overfit model, with poor out-of-sample performance.

Noise causes error through variance in the observed values, which again is caused by unpredictable changes in the system under study or measurement error. This error is irreducible and cannot be modeled by any algorithm.

(López de Prado, 2018, chapter 6)

A model's error can be summarized with the following equation:

$$E(h(x) - y) = Var(h(x)) + [Bias(h(x))]^2 + Var(\epsilon)$$

Bias and variance are inversely related to each other. Trying to reduce one will cause an increase in the other. The optimal model strikes the perfect balance between the two. A model with high bias tends to underfit the training data, while a model with high variance tends to overfit. In the case of financial data, variance is very high. Financial machine learning is therefore particularly vulnerable to overfitting. Applying enough regularization is therefore very important when applying ML to problems in finance.

2.4 Machine Learning Algorithms

2.4.1 Multiple Linear Regression

Linear regression is a technique for modeling relationships between a continuous dependent variable and one or more independent variables. When modeling using only a single explanatory variable, it is called simple linear regression, and when modeling using multiple explanatory variables, it is called multiple linear regression. Linear regression can be described as the simplest machine learning algorithm, where each feature is limited to enter the model linearly. This drastically limits the functional form the model can take, and therefore also its use. Compared to random forests or deep neural nets, a linear regression model is much more restricted in its ability to learn patterns from data. A multiple linear regression takes the following form:

$$y = x' * \theta + \epsilon$$

Where x is a vector of explanatory variables, and θ is a vector of parameters. Linear regression models are most often fitted using an ordinary least squares approach, where the parameter vector is adjusted to minimize the following objective function:

$$L(\theta) = \frac{1}{n} * \sum_{i=1}^n (f(x_i, \theta) - y_i)^2$$

In order for linear regressions fitted via OLS to be appropriate to use, multiple conditions should be met. Most notably:

- The dependent variable should be a linear function of the independent variables.
- The error term should have a population mean of zero.
- The independent variables are uncorrelated with the error term.
- Observations of the error term are uncorrelated with each other.
- The error term has constant variance.
- No independent variable is a perfect linear function of the other independent variables.
- The error term is normally distributed.

In essence, residuals should have a mean of zero and constant variance. Further, the residuals should not be correlated with each other. If these assumptions are met, the OLS objective function will produce the best possible estimates, and as the number of data samples increases towards infinity, the model's coefficients will converge on the true population parameters.

(Walpole, Myers, Myers and Ye, p. 444 and 454)

2.4.2 Principal Component Analysis and Principal Component Regression

Principal Component Analysis (PCA) is a dimensionality reduction algorithm. It works by finding the n orthogonal vectors constructed from linear combinations of input variables (features) that best preserve the variance of the original data. Where n is the number of principal components (PC) the dataset should be projected down to. In short, the algorithm goes through the following steps:

- Find the center of the high-dimensional space and move all data points such that the dataset's center is at the origin while preserving the relative distances between all points.
- Find the line through the original high-dimensional space that minimizes the sum of squared distances from each data point to the line. This first best-fit line is called principal component 1 (PC1).
- Find the next best fitting line (PC2) given that it goes through the origin and is orthogonal to PC1.
- Find the third best fitting line (PC3) given that it goes through the origin and is orthogonal to PC1 and PC2.
- Continue this process until the number of principal components is equal to the dimensionality of the dataset or the number of samples in the dataset.
- Project all data points onto the n first principal components, where n is the number of PCs to project the dataset down to.
- The values of the projected points for each principal component is what comprises the final result of the process. That is; a dataset with reduced dimensionality that best preserves the variance of the original dataset.

Principal Component Regression is to first apply PCA to the dataset and then train a linear model on the principal components. This has the advantage of maintaining a lot of the structure of the original dataset while having condensed the data into a few orthogonal features. This largely solves the multicollinearity problem. A caveat of PCA is that the PCs capturing the highest amount of variance are not necessarily the most predictive features, which can affect model performance.

(Pratap Dangeti, 2017, Principal Component Analysis – PCA)

2.4.3 Decision Trees

Decision trees sort instances by a series of if-then rules down a tree-like structure from a root node to a leaf node. There are multiple different decision tree algorithms, like ID3, C4.5, C5.0, and CART. This project used Scikit-learn's implementation of decision trees, which is based on the classification and regression tree (CART) algorithm. CART constructs binary trees, which means that each decision node in the tree only has two branches or leaves. A branch connects a sub-tree containing one or more decision nodes to the rest of the tree, while a leaf is the end of a path in the tree and represents the tree's final output/decision.

The first decision node in the tree is called the root node. The classification or regression process starts here. The CART algorithm performs a greedy search at each decision node for the feature and threshold to use in the further division of the node's data, such that impurity is minimized. How impurity is measured depends on whether the tree is used for classification or regression. In the case of classification, two standard impurity measures used by CART are Gini and Entropy.

Entropy is the classical measure and stems from information theory. According to information theory, entropy expresses the minimum number of bits of information needed to encode the classification of an arbitrary member of the node's data. For example, if it were known that all data samples at the node are of the same class (class A), a random sample would with probability of 1 be a member of class A. Because there is no uncertainty with regards to the class of the drawn sample, no information is needed to encode the class, and entropy is 0 (best score). On the other hand, if there were equally many samples belonging to class A and another class B, each draw would have a 0.5 probability of belonging to the class A and a 0.5 probability of belonging to class B. In this case, one bit is needed for each draw to encode its class. Entropy is in this case; 1 (worst score). Entropy for a binary classifier is calculated by:

$$Entropy(X) = -p_A * \log_2(p_A) - p_B * \log_2(p_B)$$

Where:

- X is the set of data samples belonging to either class A or B.
- p_A is the ratio of samples belonging to class A and the total number of samples.
- p_B is the ratio of samples belonging to class B and the total number of samples.

Gini impurity has a somewhat different interpretation, which will not be discussed, but the trees produced by using entropy and Gini as impurity measures are generally very similar. Gini impurity for a binary classifier is expressed as:

$$Gini(X) = p_A(1 - p_A) - p_B(1 - p_B)$$

(Tom Mitchell, 1997, s. 52-67)

In the case of regression trees, the mean squared error (MSE) is commonly used. See section 2.3.7 for a description.

A mathematic description of the training process of the CART algorithm is:

Given training vectors $x_i \in R^n, i = 1, 2, \dots, I$ and a label vector $y \in R^I$ a decision tree recursively divides the space such that samples with the same labels are grouped together in increasingly purer groups. At node m in the tree, we have the set of data Q . For each possible split $\theta = (j, t_m)$ consisting of feature j and threshold t_m , divide the data into two subsets; $Q_{left}(\theta)$ and $Q_{right}(\theta)$.

$$Q_{left} = (x, y) \mid x_j \leq t_m$$
$$Q_{right}(\theta) = Q \setminus Q_{left}(\theta)$$

The impurity at node m is calculated using impurity function $H()$. $G(Q, \theta)$ is the weighted average impurity of $Q_{left}(\theta)$ and $Q_{right}(\theta)$.

$$G(Q, \theta) = \frac{n_{left}}{N_m} H(Q_{left}(\theta)) + \frac{n_{right}}{N_m} H(Q_{right}(\theta))$$

Parameters θ are selected such that $G(Q, \theta)$ is minimized.

$$\theta^* = \operatorname{argmin}_{\theta} G(Q, \theta)$$

Perform the above steps recursively for subsets $Q_{left}(\theta^*)$ and $Q_{right}(\theta^*)$ until the maximum tree depth or perfect impurity is reached. Perfect impurity means that only one class is present in each leaf node.

(Scikit-Learn, 2019, Decision Trees)

2.4.4 Random Forests

A random forest is a collection of decision trees, where the prediction of the forest is some weighted average of the predictions of the individual trees. Random forests belong to a class of machine learning algorithms called ensemble methods. An ensemble is a group of predictors. Many of the most powerful ML algorithms today are types of ensemble methods. Random forests have shown to provide very good solutions to a wide range of problems, stock return prediction among them (Gu Kelly and Xiu, 2018, p. 15). The intuition behind ensemble methods is that collections of weak learners can be combined to create an increasingly strong learner. A weak learner is an ML model that only performs slightly better than chance.

To illustrate the power of ensemble methods, let us say we have an ensemble of 1000 predictors solving a binary classification problem. Predictors have a probability of 55% of being correct and are independent of each other. Further, we decide that the prediction of the ensemble is the majority vote of the individual predictors. The probability of the ensemble being correct X number of times is binomially distributed. We want to find the probability that the majority ($X \geq 501$) of predictors will be correct when classifying a sample. Via the central limit theorem, we can approximate the binomial distribution with the normal distribution with mean: $\mu = np$ and variance: $\sigma^2 = np(1 - p)$, we get:

$$Z = \frac{X - np}{\sqrt{np(1 - p)}} \xrightarrow{d} N(0, 1)$$

$$P(X \geq 501) = P\left(Z \geq \frac{501 - 1000 * 0.55}{\sqrt{1000 * 0.55 * (1 - 0.55)}}\right)$$

$$P(Z \geq -3.115) = 1 - P(Z \leq -3.115) = 1 - 0.0009 = 0.9991$$

In other words, the probability of correct classification with an ensemble of 1000 learners, each having a probability of 55% of being correct, is 99.91% (assuming learners are independent/uncorrelated). This example does not translate well onto a real-world ensemble model, as learners are generally correlated to some extent because they are often trained using overlapping data and take the same functional form.

Ensemble methods work best when learners are as independent as possible. To achieve this, one could have the predictors built on different ML algorithms. Another strategy is to use the same algorithm for all predictors but train each of them on different random subsamples of the data. When these subsamples are formed with replacement, the process is called bagging (short for bootstrap aggregating). This process generally increases the bias of the individual predictors, but

aggregating the predictions of multiple learners through the aggregation function (typically the mode or majority of predictions) will ultimately reduce both the bias and variance of the ensemble. In the end, an ensemble will generally exhibit the same bias as an individual learner trained on all the data but have less variance. The reduction in variance is directly related to the extent of the correlation between predictors. Lower correlation yields a higher reduction in variance per extra predictor. (López de Prado, 2018, chapter 6)

Another technique to increase independence between predictors and reduce variance is to apply bootstrapping to the feature space. This way, predictors are trained on random subsamples of both the training data and feature space. Random forests are in most cases bagging applied to both the data samples and the feature space in the training of multiple decision trees and finally aggregating the predictions of all the trees by taking the majority vote (also called hard voting).

As overfitting is a major concern when applying ML to financial data, it is important to sufficiently regularize the trees. Several approaches can be taken to this end:

- Constrain the depth of each tree in the forest
- Set a lower limit for the fraction of samples required at each leaf
- Set a lower limit for the fraction of samples required to split an internal node
- Set a lower limit for the decrease in impurity required to split a node
- Stop growing the tree along branches where the impurity of the samples at that branch is less than a given threshold

Each of these techniques will reduce the number of patterns each tree can learn and thus limit the chances of overfitting.

2.4.5 Artificial Neural Networks

Artificial Neural Networks (ANN) are arguably the most potent class of ML algorithms. They are very scalable and can learn very deep and complex patterns. ANNs can solve both regression and classifications problems. The general structure of an ANN can be described as a set of input nodes (artificial neurons) connected to one or more output nodes through zero or more “hidden” layers of nodes. The output layer commonly has one node for each class for classification problems and one output node for regression problems. There are several types of ANNs. This project is only concerned with fully connected feed-forward deep neural networks. In a fully connected network, each node is connected to all nodes in the previous layer. In a feed-forward neural network, connections between nodes do not form a cycle. Connections can strictly form direct chains from the input to the output layer. An ANN is said to be deep when the network has one or more layers between the input and output layers. This enables more interactions among features and learning of more complex patterns. Figure 1 shows a simple deep neural net with one hidden layer.

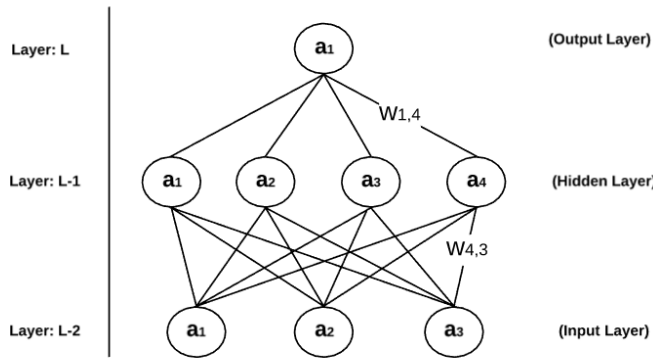


Figure 1: Basic structure and notation of a deep neural network

The figure also shows some common notation. The activation of neuron i in layer $(L-1)$ is noted a_i^{L-1} . The output layer has index L , and the other layer is index relative to the output layer. The weight the activation $a_i^{(L-1)}$ gets in the calculation of the activation $a_k^{(L)}$ is noted $w_{i,k}^{(L)}$. In addition to weights and activations from the previous layers, a constant term called the bias is also involved in activation calculations. The bias associated with node i in layer l is noted $b_i^{(l)}$. The activation of each neuron always gets transformed through an activation function. This aids model learning, but it is not intuitive or easy to explain how or why. Treatment of these questions is therefore omitted. A popular activation function (which networks in this project employ) is called a rectified linear unit (ReLU). It is defined as:

$$ReLU(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

The activation of neuron i in layer l , using ReLU as the activation function, is expressed as:

$$a_i^{(l)} = ReLU\left(\sum_{k=1}^K \left(a_k^{(l-1)} w_{i,k}^{(l)}\right) + b_i^{(l)}\right)$$

From this formula, one can see that an ANN with only an input and output layer is essentially a linear regression model.

Two more elements are needed to train an ANN: a cost/loss function and an optimization algorithm. The cost function quantifies the ANNs error while the optimization algorithm tunes the weights and biases of the network to minimize the cost function. For regression problems, a typical cost function is the Mean Squared Error (MSE) (see section 2.3.7 for a description).

Gradient descent (GD) is a canonical optimization algorithm for ANNs. The calculus will be omitted here, but the idea is to recursively calculate the negative gradient for all weights and basis starting from the output layer and moving backward to the input layer (backpropagation). This process is done for all training samples, resulting in one gradient per weight and bias for each training sample. Finally, the resulting gradients for each weight and bias are averaged across all training samples. The resulting “mean gradients” are used to nudge the weights and biases of the network by multiplying each with a parameter α , called the learning rate. The optimal learning rate varies from problem to problem.

Calculating the negative gradients for all training samples every time parameters are updated is very computationally demanding. Stochastic gradient descent (SGD) is a solution to this problem. By dividing the training set into smaller random subsets (batches) and train on one batch at a time, the

processing times are significantly reduced. The cost of applying SGD over GD is that each change made to model parameters are less accurate. With a too high learning rate, the algorithm may struggle with locating a minimum. On the other hand, a too low learning rate can result in getting stuck at a local minimum. It is therefore very important to configure the learning rate appropriately when using SGD.

Deep neural nets have an amazing ability to learn complex patterns from data. In extreme cases, the resulting network can be described as having learned the entire training set. This complete failure to generalize cause out-of-sample performance to suffer. It is vital to properly regularize DNNs. The techniques employed by this project are learning rate shrinkage, early stopping, ensemble methods, and drop out.

Ideally, the learning rate is changed adaptively during training. As the gradients approach a minimum, it becomes increasingly important to make precise changes to network parameters. Otherwise, the optimizer may overshoot, resulting in a worse model than before. As the gradient approaches zero, the learning rate should also approach zero. Otherwise, noise in the gradient calculations overshadows the directional signal (Gu, Kelly, and Xiu, 2018, p. 20). A popular optimizer which includes a scheme to change the learning rate adaptively is Adaptive Moment Estimation (Adam).

Early stopping refers to terminating model learning before the optimal specification (according to training data) has been found. This reduces the chance of overfitting. One way to detect overfitting is to measure the change in prediction errors on the validation set. When the errors on the validation set start to increase, it indicates that further learning will not improve out-of-sample performance, but rather contribute to overfitting. Algorithms implementing an early stopping scheme commonly allow configuration of the number of epochs over which improvement is required (patience parameter) and how much the optimizer must reduce the loss for it to be considered an improvement (minimum change parameter).

The learning process of a neural network has an element of randomness in that it is common to initialize weights and biases by drawing random values from a probability distribution (e.g., Gaussian). When training the model, the different parameter initiations end up producing different networks and different forecasts. By averaging predictions across multiple networks, prediction variance is reduced.

Dropout is a technique where neurons in the network are randomly ignored (not consider during the forward or backward pass) during the training process. At each training step, the probability of a neuron being kept is p and the probability of it being dropped is $(1-p)$. This forces the network to not rely on any particular set of inputs or patterns, as they may be destroyed in the next iteration of the learning algorithm. The ANN ends up learning many redundant patterns and sort of averaging across all of them to produce each prediction. Dropout leads to some of the same advantages as ensemble learning.

2.5 Financial Machine Learning

Financial machine learning is the topic of applying ML to finance. It is not widely accepted as its own discipline, but it is useful to do so. This is because:

- it often is necessary to apply additional data preparation steps
- high variance and noise in the data makes models prone to overfitting
- model and variable instability complicates model development
- backtest overfitting lead to the adoption of wrong conclusions

Financial ML also differs from econometrics. Econometrics is the application of classical statistical methods to economic and financial series (López de Prado, 2018, chapter 1). A canonical tool of the econometrician is multiple linear regression, a tool with severe limitations. ML, on the other hand, includes a comprehensive set of modeling tools that allow modeling of complex non-linear relationships.

The advanced modeling capabilities of ML should not be seen as a replacement for economic theory. If a predictive model is obtained, it is still necessary to find out what features are predictive and why. Theories can spawn from modeling efforts, but they must still be tested and verified on independent data.

The adoption of ML in finance is a natural development, in line with the algorithmizing that has been ongoing for many years. Most efforts have been centered on automated decision making based on predefined rules. For example, buying a stock based on the presence of some signal, or limiting trading based on some risk metric. The next wave of automation will not be about following rules but making judgment calls (López de Prado, 2018, chapter 1). Human beings are not particularly good at making fact-based decisions. Our information processing bandwidth is limited, and our emotions can come in the way. In situations where data-driven decisions need to be made, investors are better served by computer systems. This rationale applies to virtually every area of finance: granting loans, forecasting inflation, and much more. Another advantage of a computer system is that it always complies with the law when programmed to do so. If the development process of these systems is well managed, unlawful conduct can be mitigated. Further, if some part of the system should fail, it is much easier to trace back how it happened by studying system logs. Algorithms are also easy to update and improve over time. (López de Prado, 2018, chapter 1)

This sub-chapter will present some different aspects of financial machine learning. The following topics are covered:

- Challenges of applying ML to Financial Data
- Producing a Structured Dataset
- Labeling Financial Data
- Oversampling and Sample Weights
- Why Ensemble Methods and Random Forests are well suited for Finance
- Cross-validation in Finance

Backtesting is also a central topic, but this will be covered in a separate section (section 2.8).

2.5.1 Challenges of applying ML to Financial Data

Financial data pose many problems when applying ML algorithms to it. This section will go over some of the most common offenders.

At the highest level, there are only two problems; an insufficient quantity of data and poor quality of data. Having insufficient data is a particularly detrimental problem in finance because of financial data often contain a sizable amount of variance and noise. To extract enough signal for an ML model to generalize well, a lot of data is needed. Generally, if data quality is low, the amount must increase. Unfortunately, financial data can be expensive to acquire, and there is also a limited amount of it. On the one hand, market exchanges generate terabytes of market-microstructural data every day, but at the same time, modelers are limited to a single version of history, only going a few decades back (sometimes centuries). Further, due to the evolving nature of the financial markets, data captured three or four decades ago may be of little utility when trying to understand current or future market dynamics.

Poor quality of data is the other problem. There are multiple sources to the quality problem in financial data. Some of the most important ones are:

- Getting data with a high signal-to-noise ratio is very difficult, as it often is no strong or clear relationships between dependent and independent variables.
- What features are most predictive is often unknown. This results in many irrelevant features being introduced into financial datasets.
- Information arrives at a variable rate, yet sampling is often performed at a fixed rate. This causes some samples to contain more information than others, adversely affecting model training.
- Outcomes/labels span multiple observations, leading to oversampling.
- The observations often have serial and cross-sectional dependence; this can cause information leakage between training and testing sets if not accounted for.
- The markets go through regime switches, and the optimal model and parameters may change over time.
- Only a single path of history is available to us, and the future may not unfold as the past has. Models trained to fit historical market conditions will, in this case, not be suited for use under future market conditions.

Poor data quality, combined with ML algorithms ability to detect complex patterns makes the resulting models prone to overfitting. Overfitting means that models fitted the training data too well and failed to generalize, leading to poor out-of-sample performance.

A related problem is overfitting on backtests. ML models are often used by larger systems. When these systems are backtested, the developer often ends up running multiple tests and tweaking the system between each run. Add in selection bias, and the reported results can end up being grossly inflated.

The following sections will present some solutions to the problems described above.

2.5.2 Producing a Structured Dataset

2.5.2.1 Types of information

There are many types of information that could be relevant when producing investment strategies. Four categories of information are:

Table 1: Types of information relevant to financial modeling (López de Prado, 2018)

Fundamental Data	Market Data	Analysis	Alternative Data
Assets	Volume	Analyst recommendations	Satellite images
Liabilities	Dividends/coupons	Credit ratings	Twitter
Costs/earnings	Quotes/cancellations	News sentiment	Google searches
Macro variables	Price data	Earnings expectations	Metadata

Only creativity limits the investor in what type of information can be applied in the search for alpha. Everything from time series regressions using price data, to analyses of satellite images of parking lots to infer earnings numbers before they are released, are possible angles of attack. Maybe the most applied types of data, at least in academia, is fundamental data and market data. Fundamental data is the type of data found in SEC filings. This type of information is typically released quarterly according to the company's fiscal year. An important nuance of fundamental data is that it is reported with lapse. Another complexity is that firms sometimes make mistakes and correct these by releasing updated versions of their filings. Further, information that was not available at the date of filing may be backfilled, meaning that the missing data is assigned a value. This makes it very important to control the filing's exact time of release to the general public. Modeling using data prior to its release will invalidate results.

Market data refers to data about trading activity on exchanges. Ideally, one has access to a raw feed of all types of unstructured information. With access to Financial Information eXchange (FIX) messages, one could reconstruct the trading book, which contains a wealth of information. Unlike fundamental data, market data is abundant. Multiple terabytes is generated every day. This increases the complexity of processing, but also makes it more interesting for strategy research.

Analytics is information produced by a research firm, investment bank, or other professional entity. Although this type of data can be valuable, it is often costly to acquire.

Alternative data can be anything. It is the types of information that does not fit into any of the other categories. For example, before Walmart reported an increase in earnings in their 10-Q filing, their parking lots got more crowded. Information about the increase in earnings was there but had not yet manifested itself as dollars on the income statement. A problematic aspect of alternative data is costs and privacy concerns. The benefit of alternative data is that it has the potential of yielding information no one else has, creating information asymmetries.

2.5.2.2 Building a structured dataset from unstructured financial data

One of the first steps of any ML project is to gather data. As just discussed, data comes in many forms. For ease of use, it is advisable to capture and store data in a structured manner. Datasets are generally arranged as a two-dimensional structure of rows (observations/samples) and columns (features and labels). It is common in the financial industry to call a row in a dataset a bar. There are multiple different methods to form bars. The most common method in the academic literature is "time bars." Time bars are created by sampling data at a fixed time interval. A problem with this approach is that the financial markets do not process information at a constant rate. For example, activity is higher during the open and close and lower in the middle of the day. The result is oversampling when activity is low and under-sampling when activity is high. Time bars also have bad

statistical properties. Specifically, high serial correlation, heteroscedasticity, and non-normality of returns. These characteristics can be mitigated by complicated transformations of the time series or avoided altogether by sampling according to a different scheme. An example is to create “dollar bars.” Dollar bars are created by sampling data every time a pre-defined amount of market value is exchanged. If bars were created based on volume or ticks, the number of samples would vary widely over the years as the price of stocks changes, either through changes in how the market values the stock or through arbitrary corporate actions (splits, reverse splits, dividends, etc.). By sampling according to the dollar amount exchanged, sampling happens as a less volatile rate.

Another approach to sampling is to sample according to the rate of new information being released in the market. The rate of new information is related to market activity. The more unexpected change there is in market activity, the more information seems to have entered the market. An example of how to capture the change in market activity is by detecting unexpected changes in volume or exchanged market value for different instruments. Expectations can, for example, be formed based on an exponentially weighted average of past volume or market value. When the volume or market value for a bar exceeds the expected value, the bar is sampled.

Without access to market microstructural data, building a custom structured dataset from unstructured financial data was not possible to do for this project. It was therefore decided that the best approach was to acquire a license for a dataset developed by a professional vendor of financial data. This data consisted primarily of price, volume and dividend data sampled at a daily interval and 10-K and 10-Q filings, sampled at the filing date. Despite avoiding the problem of generating a structured dataset, much work remained in order to produce predictive features sampled at a monthly interval, suitable for ML.

2.5.2.3 Sampling for Machine Learning

Machine learning works better when the algorithms are trained on informative data. Many algorithms also do not scale well to large datasets. Due to these reasons, it is often necessary to sample down the data in the structured dataset, either to make its size manageable, increase the ratio of informative samples or both. If all data in the structured dataset are of equal relevance, downsampling can be performed by sampling randomly according to a uniform distribution. If some data are of higher relevance than others, we want to train our model on the most relevant samples. Investors typically put on a trade after some event takes place. An event could be the release of a macroeconomic statistic, an earnings announcement, or some market microstructural event. One strategy to build predictive models is to sample when an event occurs and then use machine learning to see if there is some function that is predictive under those circumstances. If this is not the case, what constitutes an event can be redefined, and a new dataset and model can be produced.

The sampling technique used in the development of this project’s dataset was inspired by the concept of event-based sampling. As features of the dataset heavily relied on data from form 10-K and 10-Q filings, sampling was done as close as possible to filing dates of financial statements. The more current the information in each sample, the better. The faster new information is sampled, the more likely it is that the market has not yet properly adjusted to the new information and that this will yield some predictability that ML models can be trained to detect. Although it is highly unlikely that this effect will be strong, this sampling technique is surely superior to sampling at a fixed time interval (for example at the end of each month), where the data for most samples will be unnecessarily outdated. The data behind each sample will, in this case, be up to one month older than necessary.

2.5.3 Labeling Financial Data

How a dataset should be sampled are, of course, entirely dependent on the problem at hand. This thesis is interested in predicting stock returns (or equity risk premiums) with both classification and regression models. By far, the most popular labeling method for stock return prediction is to label observations according to the return over some fixed time horizon. This is often done by labeling the observations;

$$y_t = \begin{cases} 1 & \text{if } r_{t,t+h} > \tau \\ 0 & \text{if } |r_{t,t+h}| \leq \tau \\ -1 & \text{if } r_{t,t+h} < -\tau \end{cases}$$

Where y_t is the label, $r_{t,t+h}$ is the return over fixed time horizon h from time t and τ is a constant threshold.

A disadvantage of this strategy is that the threshold is static. The volatility of different stocks varies widely. Because of this, many low volatility stocks will have their observations labeled with 0, despite their returns being predictable and statistically significant. This is an error commonly made in the literature. A better approach is to dynamically set the threshold according to the prevailing volatility of each stock. The volatility could, for example, be measured by an exponentially weighted moving standard deviation of past returns. However, even this approach has flaws because it does not account for the path followed by prices. All investment strategies have stop-loss limits. To build a strategy that profits from trades that would have been stopped out by the exchange is not realistic. A labeling technique that accounts for the path followed by prices is called “the triple barrier method.” It was developed by Marco López de Prado and presented in “Advances in Financial Machine Learning” (2018). This method was used extensively throughout this project and will be explained in detail under chapter 3: Methodology, in section 3.1.3.2. In short, samples are labeled by the first out of three barriers that are touched. Assuming a long position is taken, the upper barrier represents the take-profit limit, the lower barrier represents the stop-loss limit, and the vertical barrier represents the timeout of the position. If the return exceeds the upper barrier, the sample is labeled 1; if the return goes below the lower barrier, the sample is labeled -1. The width of the upper and lower barrier can be set dynamically based on past price volatility.

How to label samples when the vertical barrier is touched first can be done in multiple ways. One way to do it is to label samples with 1 or -1 if the return of the position is positive or negative, respectively, when the vertical barrier is hit. Another strategy is to label the sample with 0 if the returns do not exceed some predefined limit.

It is important to realize that which barrier represents the stop-loss and take-profit limit depends on whether it is a long or short trade/position. When labeling to learn how to set the side of trades, the trade is always defined to be long. In this case, the upper barrier always represents the take-profit limit, and the lower barrier represents the stop-loss limit. Figure 2 illustrates an example of this. Here the upper barrier (take-profit limit) was hit first, and the resulting label is therefore 1.

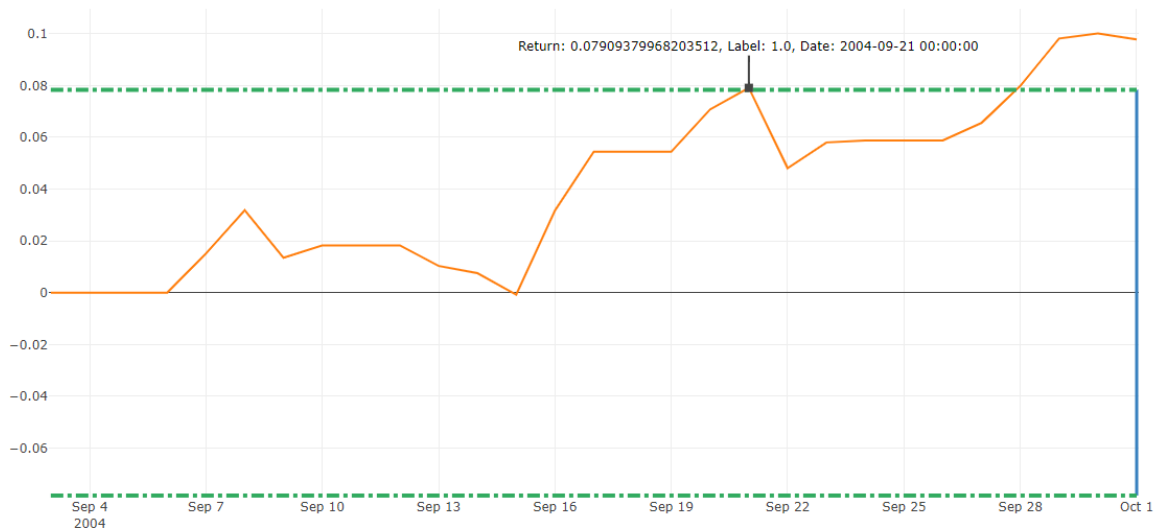


Figure 2: Triple Barrier Method for Labeling Financial Data

A problem that remains unsolved is how to size the trades. The problem of bet sizing is rarely treated in the literature (López de Prado, 2018). In “Advances in Financial Machine Learning”, López de Prado presents a technique called “meta-labeling.” The idea is to train a secondary ML model to use a primary exogenous model. The primary model makes predictions that are used to set the side of the trades (each sample in the dataset being labeled represents a potential trade). Then, triple barrier search is performed a second time, but this time the side of the trades are set by the primary model. The labeling process will end up labeling the samples according to whether following the advice of the primary model gave positive or negative returns. Using these labels, a classification model can be trained to identify when to follow the advice of the primary model and when not to. This is a binary classification problem, and the probability the model assigns to the class representing: “follow the primary model’s advice” can be used in further calculations to derive the dollar amount to allocate to each trade. More details on meta-labeling will be given in chapter 3: Methodology.

Another problem that comes up when labeling data for ML is imbalanced classes. Some ML models do not perform well when trained on data where some classes have higher representation than others. In these cases, it is desirable to drop very rare labels and build a model for the more common outcomes.

2.5.4 Sample Weights

In finance, it is very common that labels overlap in time. When the goal is to sample at a daily frequency, the time horizon each sample’s label could span would be limited to one day if no overlap was allowed. It is often desirable to sample at a frequency greater than the span of each label. The problem of this is that some periods get oversampled and labels are no longer independent. To adjust for this effect, one can introduce weighting schemes to limit the influence of overlapping outcomes. The approach taken in this project was only to allow labels to overlap in rare cases. The sampling scheme used in this project will be explained in section 3.1.2.

2.5.5 Why Ensemble Methods and Random Forests are well suited for Finance

Financial data often have high variance and contain a sizable amount of noise. This makes accurate modeling of most economic phenomena very difficult. Even if variance and noise were handled optimally, the problem of model uncertainty and parameter instability makes it hard to avoid introducing bias into the model. Following this line of thought, it stands to reason that flexible

models, such as random forests and deep neural nets could be of great value to financial modelers. However, with great flexibility comes greater chance of overfitting. With the high levels of variance and noise in financial data, overfitting is one of the greatest challenges to solve. One way to mitigate the problem of bias and/or variance is by using ensemble methods. By using bagging to construct multiple random sub-samples of the dataset and training one model on each sub-sample, the variance in the forecasts can be reduced. The variance of a bagged prediction $\varphi_i[c]$ is a function of the number of bagged estimators (N), the mean variance of a single estimator's prediction ($\bar{\sigma}$), and the mean correlation among their forecasts ($\bar{\rho}$), expressed through the following equation:

$$\begin{aligned} V\left[\frac{1}{N}\sum_{i=1}^N \varphi_i[c]\right] &= \frac{1}{N^2}\sum_{i=1}^N\left(\sum_{j=1}^N \sigma_{i,j}\right) = \frac{1}{N^2}\sum_{i=1}^N\left(\sigma_i^2 + \sum_{j \neq i} \sigma_i \sigma_j \rho_{i,j}\right) \\ &= \frac{1}{N^2}\sum_{i=1}^N\left(\bar{\sigma}^2 + \sum_{j \neq i} \bar{\sigma}^2 \bar{\rho}\right) = \frac{1}{N^2}\sum_{i=1}^N(\bar{\sigma}^2 + (N-1)\bar{\sigma}^2 \bar{\rho}) = \frac{\bar{\sigma}^2 + (N-1)\bar{\sigma}^2 \bar{\rho}}{N} \\ &= \bar{\sigma}^2\left(\bar{\rho} + \frac{1-\bar{\rho}}{N}\right) \end{aligned}$$

Where, $\sigma_{i,j}$ is the covariance of predictions by estimators i and j

$$\begin{aligned} \sum_{i=1}^N \bar{\sigma}^2 &= \sum_{i=1}^N \sigma_i^2 \leftrightarrow \bar{\sigma}^2 = \frac{1}{N}\sum_{i=1}^N \sigma_i^2 \\ \sum_{j \neq i} \bar{\sigma}^2 \bar{\rho} &= \sum_{j \neq i} \sigma_i \sigma_j \rho_{i,j} \leftrightarrow \bar{\rho} = \frac{\sum_{j \neq i} \sigma_i \sigma_j \rho_{i,j}}{\bar{\sigma}^2 N(N-1)} \end{aligned}$$

(López de Prado, 2018, chapter 6)

From this result, it is obvious that bagging is only effective if $\bar{\rho}$ is less than 1 and that variance of the ensemble's predictions are reduced as $\bar{\rho}$ is reduced. Increasing the number of predictors can also reduce prediction variance, provided predictors are sufficiently uncorrelated so as not to have a negative effect on $\bar{\rho}$. To produce independent predictors, it is important to produce training samples that are as independent as possible. One way of doing this is by applying bagging to both the dataset's observations and feature space (which random forests do).

As explained earlier, ensemble methods can also increase the accuracy of a classifier. See section 2.4.4 for an explanation of this phenomenon.

Random Forest

Random forests apply bagging to both the sample and feature space of the training set. In addition to this, random forests apply another level of randomness. When finding the optimal feature and threshold at each node split, only a random subsample (without replacement) of the features will be considered. This helps to further diversify the training process of each decision tree in the forest. Unfortunately, decision trees are quite prone to overfitting, and overfitting increases the variance of predictions. To reduce the problem of overfitting, implementations of random forests often provide many options to regularize the decision trees, like:

- Reducing the maximum number of features available to each tree, to force diversification in training sets.

- Force the training process to stop early by increasing the fraction of the training samples necessary to be at a leaf node.
- Reduce the influence of overlapping samples by limiting the number of samples each tree is trained on.

Random forests can be a very powerful algorithm to apply to problems in finance due to its ability to reduce variance in predictions. This, however, relies on each decision tree in the forest to be as uncorrelated as possible and not overfitted. (López de Prado, 2018, chapter 6)

Boosting vs. Bagging

A powerful technique to use when modeling less noisy data is boosting. Boosting is a technique for training ensembles of predictors, where predictors are trained sequentially. In the beginning, random sampling with equal weighting is performed across the dataset. After training each new predictor, the samples the predictor misclassify have their weights increased. Samples with higher weight have a higher probability of being included in future training sets. This way, new predictors are trained on increasingly more of the samples earlier predictors performed poorly on. There are many boosting algorithms available, each with its nuances. The problem of applying boosting in finance is that it increases the probability of overfitting. Although boosting can reduce both bias and variance in predictions, it often does not cover the cost of increased probability of overfitting. This argument favors bagging over boosting in finance, where the data often have a low signal-to-noise ratio. Another advantage of bagging is that predictors can be trained in parallel, enabling the use of multiprocessing, which can drastically reduce processing times. (López de Prado, 2018, chapter 6)

2.5.6 Cross-Validation in Finance

Cross-validation (CV) is a statistical technique to measure the generalization error of a model. This allows detection of overfitting. If performance in-sample is persistently higher across all iterations of cross-validation compared to the validation fold, the algorithm is likely configured sub-optimally and tend to overfit the training data. Being able to detect overfitting requires the validation folds to be independent of the training folds. In finance, data samples are seldom IID, and detection of overfitting becomes unreliable. Further, CV will even contribute to overfitting through hyperparameter tuning (López de Prado, 2018). This is because hyperparameters will partly be fitted to the training set due to the overlap of information between it and the validation and training folds.

Cross-validation in its simplest form is to split the dataset into a training and test set, which allows the modeler to train the ML algorithm using the training data and then assess out-of-sample performance using the test data. In this simple form, the datasets samples are assumed to be drawn from an IID process. A popular alternative scheme is k-fold CV where the dataset is partitioned into k subsets/folds, and for each fold, an ML algorithm is trained on all folds except the fold currently serving as the validation fold. The output of the process is a set of performance metrics for each validation fold in the dataset. This has the advantage that the entire training set is used to both train and validate the model, giving a more robust impression of out-of-sample performance and how well the model generalizes.

Cross-validation fails when applied in finance because samples are rarely drawn from an IID process. If this is not adjusted for, information will leak between training and validation folds, leading to overfitting and invalid performance metrics. For clarity, consider a serially correlated feature X associated with labels Y which span overlapping data. The serial correlation causes X_t to be approximately equal to X_{t+1} and the overlapping data behind the labels causes Y_t to be approximately equal to Y_{t+1} . When data sampled at time t is part of the training data and data

sampled at time $t+1$ is part of the testing/validation set, information is leaked. When the model is asked to make a prediction based on X_{t+1} the model is more likely to predict Y_{t+1} even if X is an irrelevant feature. In the case where X is a relevant feature, the error may seem less gross, but the out-of-sample performance metric will not be representative for the true out-of-sample predictive power of the model. Note that the case where Y_t and Y_{t+1} are not formed on overlapping information, but X are still serially correlated, is not necessarily a case where information is leaked. For leakage to take place in such a way that training is affected adversely, it must be the case that $(X_t, Y_t) \approx (X_{t+1}, Y_{t+1})$ (López de Prado, 2018, chapter 7).

Two approaches to mitigating information leakage are to drop samples from the training/validation set if they have overlapping labels with the training set and reduce the predictor overfitting. A technique presented in “Advances in Financial Machine Learning” (López de Prado, 2018, chapter 7) to remove label overlapping is called: purged k-fold cross-validation. It works as follows:

A sample, j , has a label that spans the interval $[t_{j,0}, t_{j,1}]$ and a sample i has a label that spans the interval $[t_{i,0}, t_{i,1}]$. The labels are said to be overlapping if:

- $t_{j,0} \leq t_{i,0} \leq t_{j,1}$
- $t_{j,0} \leq t_{i,1} \leq t_{j,1}$
- $t_{i,0} \leq t_{j,0} \leq t_{j,1} \leq t_{i,1}$

As each validation fold is formed, any samples with a label that overlap with one in the other (training) folds, according to the conditions above, will be removed. This solution will sufficiently eliminate information leakage in most cases. Note that purging should be done for overlaps across stocks as well, because a label of one asset may contain overlapping information with another asset (cross-correlation). Another measure to further reduce leakage is to impose an embargo after the validation fold. Note that the embargo does not need to be imposed at the beginning of the validation fold because all information in the training data before the testing period can be assumed to be known information within the test fold. However, test samples that spill information into the subsequent training data are of concern, because it should not be the case that data of the testing period is known during a training period. An embargo can be implemented easily by adding some time threshold s at the end of the test/validation period that cannot be overlapped by any label in the training data. An illustration of label overlap and an embargo is given in figure 3.

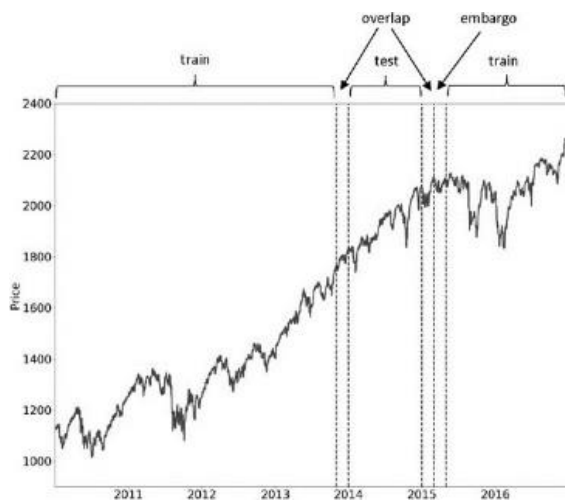


Figure 3: Overlap and embargo for splitting training and testing data (López de Prado, 2018, chapter 7)

2.6 Why Apply Machine Learning to Stock Return Prediction

There are several reasons for why ML can do well on the problem of stock return prediction.

- Machine learning is largely specialized for prediction tasks.
- Stock return prediction is notoriously difficult because the best functional form of the model and what parameters to use are unknown. It might be necessary for modeling to allow predictors to interact in complex ways, this further increase the space of possible model specifications. The field of ML contains a diverse set of algorithms, which enables exploration of a broad set of functional forms.
- There are many predictive variables in the literature. Hundreds of stock level features and tens of macroeconomic features. Many ML algorithms can handle a large set of predictors despite the presence of multicollinearity and missing data, which traditional econometric tools, like linear regressions, are not well suited for.

(Gu, Kelly and Xiu, 2018, p. 3)

2.7 Automated Trading Systems

Automated trading systems (ATS), also known as algorithmic trading systems, are computer programs that produce orders and automatically submits them to an exchange. It is a real-time decision-making system that consumes digital information (financial/market data) which it feeds to various models whose outcomes influence the decisions of the system. This type of system is widely used in the investment industry, and there is an ongoing battle to develop better trading strategies and more profitable systems. Two broad categories of automated trading systems are those based on technical analysis and those based on fundamental analysis. Technical analysis involves transforming data into technical indicators based on price patterns and other statistical features and build trading rules around these indicators. Fundamental analysis uses financial data on individual companies and the economy at large to predict price developments and base order generation on these predictions.

The developer defines the rules by which the ATS will trade securities via mathematical instructions. An ATS can be arbitrarily complex and optimize order generation across multiple dimensions. For example, the ATS may be instructed to maintain a minimum level of investment or limit order size to some percentage of total portfolio value or maintain a fixed ratio between long and short trades. There are no limits on what rules can be implemented.

An ATS can make decisions completely independent of human interaction or require some human input to function.

Some of the most fundamental problems an ATS must solve are:

- What securities to invest in
- How much to invest in each security
- When to enter positions
- When to exit positions

To solve these problems, an ATS is often built on top of models/algorithms making predictions on price developments. The predictive models generate trading signals that the ATS use to produce orders. It is common to set various limitations on order size, direction, number of securities, etc. These limitations are part of the systems risk management strategy.

Many associate automated trading systems with high-frequency trading (HFT), which is a type of algorithmic trading. HFT uses high-speed networking and hardware acceleration to enable systems to trade with very low latency. This project does not involve HFT techniques.

Although an ATS can be developed to trade any financial instrument, this project is only concerned with trading U.S. equities.

The above should give a general idea of what an ATS is. The specifics of the ATS built for this thesis is outlined in section 3.3.

2.8 Backtesting

A backtest is a historical simulation of how an investment strategy would have performed over some past period. It is important to distinguish a backtest from a scientific experiment. A backtest is at best hypothetical. In a physics experiment measuring the gravitational acceleration, the scientist can control her elevation above sea level, make measurements in a vacuum chamber, use an atomic clock and an object with known mass. All aspects of the experiment can be controlled and reproduced. This luxury does not exist when “experimenting” with financial markets. It is not possible to rewind to the early 2000s, ban the issuance of collateralized debt obligations and see how the financial crisis would have unfolded differently. History is just a random path that was realized (López de Prado, 2018), and cannot ever be reproduced. When running a backtest, one also cannot be sure how the market would have reacted to simulated actions. One can attempt to correct for their influence through a realistic slippage model and other techniques, but in the end, one cannot be sure of the results. A backtest does not prove anything.

The purpose of a backtest is to check that an investment strategy and its configuration is resilient to different market conditions. The goal is to detect underperforming models so they can be discarded before they end up losing money. A backtest can be used to verify, among other things, that:

- The strategy operates as expected.
- Makes investments that would have been profitable, at least in the past.
- The strategy is resilient to varying costs.
- Turnover is within acceptable limits.

The problem of backtesting is that they are notoriously difficult to conduct. There are several common mistakes made that corrupt results. Some of the usual offenders are:

- Survivorship bias, referring to the use of datasets with an unrepresentative amount of bankruptcies. Backtesting using a dataset only containing winners will produce results that are not representative of the real stock market.
- Look-ahead bias, referring to use of data before it was publicly known. Making investment decisions based on data that would not have been available to investors give the strategies under scrutiny an undeserved advantage.
- Data snooping, referring to a dataset being used multiple times during model selection. After running hundreds or even thousands of backtests, trying different strategies, something is bound to work eventually. But successful strategies may simply profit of a statistical fluke, not a sound investing policy.
- Miscalculating transaction costs; trading strategies being tested with unrealistically low transaction costs will have an unfair advantage. This problem becomes especially severe when the strategy relies on frequent trading. If the strategy is responsible for a large portion of the total traded volume, the market impact of trades become increasingly important and difficult to correct for.

- Relying on outliers; if a strategy relied heavily on a few exceptionally profitable trades in the production of its returns, it is likely that the strategy's performance will be unreliable when applied to new data.
- Not checking if strategy performance is dependent on which day the strategy started trading.
- Not limiting trading appropriately; examples of this is to buy more stock in a company than is available or assume that short trading is always possible.
- Selection bias; referring to the researcher only publishing successful results and not accounting for the number of trials used to achieve them. This gives the wrong impression of the statistical significance of the backtest's results. Transparency into the process that produced the results is important to be able to verify their validity.

How these pitfalls were avoided when developing this project's backtester will be detailed in chapter 3: Methodology.

One of the characteristics of the financial markets that makes it challenging to develop and backtest trading systems is that the financial markets will react dynamically to the trades being made. This poses a unique problem to financial ML. The better the models are at exploiting some anomaly, the more likely it is that it will disappear. Either through the adoption of the same technique by other investors, or that the system alone eliminates the effect of the anomaly. This would be the equivalent of an image classification algorithm trying to detect cats that would change their appearance as the algorithm got better at detecting them. This problem makes it difficult to conduct reliable backtests, particularly for large actors. The bigger you are, the larger the effect you have on the market. Although interesting, this problem does not get much attention in this report, due to the complexity of modeling this effect.

Despite the many complications of backtesting, it is possible to make educated decisions and mitigate errors, but even a flawless backtest is probably wrong due to multiple testing. Even when the most conservative assumptions are made, and everything is properly accounted for, the backtest is probably still wrong, because, in order to produce a perfect backtest, one must run thousands of them. As more backtests are conducted, the probability of false discoveries increase. It is likely that a statistical fluke is at play.

"Professionals may produce flawless backtests, and will still fall for multiple testing, selection bias, or backtest overfitting." (Baily and López de Prado, 2014)

As explained, backtesting is not a scientific experiment. Another thing backtesting is not is a research tool. A backtest gives little insight into why the strategy performed well. It cannot verify economic theories. It is entirely possible that the strategy takes advantage of irrelevant features that by random chance are correlated with profitable investments. A better approach is to study feature importance, as this gives insight into the patterns discovered by the ML algorithms. Even if the strategy was not able to monetize the presence of a predictive feature, it might be able in a different configuration. Detaching feature importance from backtesting results are important when doing strategy research.

It is tempting to tweak the strategy until it performs well on a backtest. But by doing this one runs the risk of backtest overfitting. Backtest overfitting can be defined as selection bias on multiple backtests. To avoid backtest overfitting, one should never backtest before the strategy is fully specified. Once the strategy is fully completed, it can be backtested. If the results are bad, start from scratch. By tweaking the strategy until it performs well, it is highly likely that the strategy becomes overfit to the backtest.

How to address backtest overfitting is arguably one of the most important questions in quantitative finance because if this problem is solved, investment firms would be able to achieve results with certainty, as all they would have to do is to deploy strategies that performed well in backtests. Further, journals could verify the findings reported by researchers by running their own backtests, and we would know with certainty whether a finding was a true discovery.

“Finance could become a true science in the Popperian and Lakatosian sense.”

(López de Prado, 2018)

2.8.1 Avoiding Overfitting

Avoiding backtest overfitting is of prime importance when researching investment strategies. Some general recommendations are:

- Develop strategies for entire asset classes or investment universes, like stock or bonds, rather than for specific securities. An anomaly detected on only one security is likely a false discovery, as random patterns are much more abundant for individual securities than for entire asset classes. Even if the anomaly is a true discovery, it has limited utility as it only applies to one security.
- Do not backtest until all strategy research and formulation are completed. This involves feature engineering, sampling, labeling, bet sizing, ML algorithm training, etc. If backtests are performed during strategy research, it is much more likely to be overfitted to the specific timeframe and investment universe used in the backtest.
- Track all backtests so that performance measurements can be adjusted for the number of backtests performed. See Baily and López de Prado (2014) for an example.
- Apply bagging to avoid overfitting ML models and reduce the variance in forecasting errors.
- Start from scratch if the strategy does not perform well in the backtest.
- Simulate scenarios rather than history (this point will be elaborated on in section 2.8.2).

(López de Prado, 2018, chapter 11)

2.8.2 Backtesting Paradigms

There are several approaches to backtesting. The most common in the literature is the walk-forward approach. This is a historical simulation to see how the strategy would have performed in the past. At any point during the backtest, the strategy is restricted to use data that would have been available at that point in history. The advantages of this approach are that information leakage is easy to prevent, and the results have a clear interpretation. However, the goal of backtesting is to verify that the trading strategy is sufficiently profitable and not excessively risky under various market conditions. With limited data, this may not be achievable with the walk-forward method. Another problem with this method is that only a single scenario is tested, which can lead to overfitting.

An important insight is that it is not necessarily a sin to break the chronological order of the training and testing data. If one is interested in how a strategy would have performed during the 2008 financial crisis, the researcher could train a model on data from 2009 to 2015 and then use this model when backtesting over the period 2007 to 2009. The goal is to detect sound strategies for future economic environments. What these environments will be like is unknown. It is unnecessary to limit strategy testing on the one path history ended up following, as this may not resemble the future more accurately than a rearranged version.

A backtesting paradigm that takes advantage of the above rationale is the cross-validation method. By dividing history into multiple testing sets/folds, future performance can be inferred from multiple out-of-sample scenarios. For each fold, we assume knowledge of everything outside its timeframe

and use this data for model training. Then backtesting is performed on each test fold. Cross-validation in backtesting becomes victim to the same information leakage problems as ML model development. Purging and embargoes are possible solutions to information leakage problems in this context as well.

A third approach to backtesting is to generate synthetic data with similar statistical properties to historical data. The synthetic data can be modified to produce whatever scenario the researcher wants to conduct tests under. This type of backtesting quickly becomes quite complex but provides the researcher with great flexibility and enables testing under scenarios that otherwise would not be available.

This project only employs the walk-forward method due to limited time and resources. More details about how backtesting was conducted are provided in section 3.4.

2.8.3 Risk-Adjusted Performance Measurement

Investment strategy performance measurement is not only concerned with the returns obtained, but also the risks involved in their production. Higher strategy risk should yield compensation in the form of higher returns.

2.8.3.1 Sharpe Ratio

The Sharpe Ratio (SR) is a fundamental risk-adjusted performance measure introduced in relation to the capital asset pricing model (CAPM) in the 1960s. In relation to the CAPM, the Sharpe Ratio is the slope of the capital market line. It gained significant traction in the financial industry and academia because of its ease of use and ease of interpretation. The purpose of the Sharpe Ratio is to evaluate the skill of an investment strategy or investor. The SR makes many assumptions which limit its use. The SR assumes that portfolio returns are normally distributed. Further, the financial markets are assumed to be frictionless (no transaction costs), and the risk-free rate should be constant and identical for both borrowing and lending. The SR is defined as:

$$SR = \frac{E(r_p) - r_f}{\sigma}$$

Where $E(r_p)$ is the expectation of the portfolio's return, r_f is the risk-free rate, and σ is the standard deviation of the portfolio's excess return. $E(r_p)$ is commonly calculated by the mean of past daily/monthly/yearly returns.

(UIS - Compendium for IND640, 2018, p. 256-261)

The SR does not quantify the value added by the investment strategy and is only a ranking criterion. When ranking different portfolios, it is important that all portfolio returns share the same distribution type; normal or student-t with the same degrees of freedom. The SR does not make distinctions between upside or downside risk. Upside risk is much more desirable than downside risk, and lacking the ability to distinguish between these limits the SR's ability to analyze non-normal portfolio returns.

Another drawback is that it is difficult to interpret negative Sharpe Ratios. The SR becomes negative when portfolio returns are lower than the risk-free rate. The normal intuition that the SR is increased by higher returns and lower volatility breaks down for negative values. With a negative SR, a higher volatility portfolio would appear to be superior to a lower volatility portfolio.

Because the mean and variance of portfolio returns are unknown, the true SR cannot be derived with certainty. Both statistics must be estimated, and if the assumptions underlying the SR is broken,

significant estimation errors can occur. Before applying the SR, it is important to check portfolio returns are normally distributed.

2.8.3.2 Adjusting the Sharpe Ratio for Higher Moments

Use of the Sharpe Ratio is limited to investments where returns are normally distributed. Several extensions exist to warrant usage on non-normal returns. Given non-normal returns, an investor would prefer negative skew and lower kurtosis. An adjusted Sharpe ratio that takes skew and kurtosis of the return distribution into account is defined as:

$$ASR = \hat{\lambda} + \left(\frac{\hat{\tau}}{6}\right)\hat{\lambda}^2 - \left(\frac{\hat{\kappa}}{24}\right)\hat{\lambda}^3$$

where $\hat{\lambda}$ is the estimated Sharpe ratio, $\hat{\tau}$ is the estimated skewness and $\hat{\kappa}$ is the estimated kurtosis.

This adjustment will tend to lower the Sharpe ratio if the return distribution has negative skew and positive kurtosis.

(Uis - Compendium for IND640, 2018, p. 256-261)

2.8.4 Backtest Statistics

Backtest statistics are metrics used by investors to assess and compare investment strategies. It is important to assess how the strategy operated under the simulated market conditions to make sure the strategy does not take excessive risks, exhibit unwanted biases, and is able to generate respectable profits. Investors want to weed out bad strategies before they ever get deployed, and backtest statistics will give insights to aid in this process. Backtest statistics can also help the investor improve strategies by helping to identify what the strategies did right and wrong.

Backtest statistics can be categorized into general statistics, performance, runs/drawdowns, implementation shortfall, return/risk efficiency, and classification score.

2.8.4.1 General Characteristics

Time range: The time range of a backtest specifies the start and end dates of the simulation. When deciding what timeframe to run the backtest over, it is important to include as many different market regimes as possible.

Average AUM: The Average dollar value of assets under management (AUM) is the average total market value of the strategy's investments. AUM is defined differently by different investment professionals, although the general interpretation does not change much. In the context of this report, AUM is the sum of the dollar value of all long and short positions, where short positions are considered to be a positive real number.

Maximum dollar position size: is the highest market value of any single trade the strategy held. This can detect if the strategy took any excessively large positions.

Ratio of longs: The proportion of bets that classifies as a long position. If one aim for a market-neutral long-short strategy, this number should be close to 0.5.

Frequency of bets: The number of bets made per year of the backtest. A sequence of positions with the same side in the same asset is considered to be the same bet. The bet ends when all positions in the asset are liquidated or flipped to the other side. The aim of this statistic is to find the number of independent opportunities discovered by the strategy, not to count the number of trades made.

Average holding period: This is the average number of days a bet is held by the strategy. A high-frequency trading strategy might only hold a position for a few milliseconds, while a long-term strategy might hold positions for many years.

Annualized turnover: The ratio of the average dollar amount traded each year to the average annual AUM. If the strategy frequently makes large bets, turnover will increase. Frequent betting or a few large bets alone will not necessarily result in high turnover.

Correlation to underlying: The correlation between the strategy's return and the return of the underlying investment universe (U.S. stock market in the case of this project). If the correlation is close to 1 or -1, the strategy is essentially holding or shorting the investment universe and is not adding much value. The strategy would also not add much value in terms of diversification if the investor is already holding something approximating the market portfolio.

2.8.4.2 Performance related statistics

PnL: The total dollar amount earned over the backtest's time range, including liquidation costs from the terminal position.

PnL from long positions: The dollar amount of total PnL generated from long positions. This measure can be used to assess the strategy's bias towards taking long or short positions.

PnL from short positions: The dollar amount of total PnL generated from short positions. Serves the same purpose as "PnL from long positions."

Annualized rate of return: The equivalent annual return an investor would receive over the backtested period. Presenting returns in this format allows comparison of strategies that operated over different lengths of time.

Hit Ratio: The number of trades that generated a profit over the total number of trades. This measure says something of how consistent the strategy is in picking winning investments.

Average Return from Hits: The average return from trades that generated a profit.

Average return from misses: The average return from trades that generated a loss.

Highest return from hit: The highest percentage return on a single trade. Gives indication of the strategy's maximum potential gain per trade.

Lowest return from miss: The lowest percentage return on a single trade. This gives an indication of the strategy's maximum potential loss per trade.

2.8.4.3 Runs statistics

Returns are generally not IID normal. Return series often exhibit long sequences of bets with returns in the same sign. Such sequences are called runs. To properly assess the downside risk of a strategy, it is common to quantify runs. See López De Prado (2018) for examples.

2.8.4.4 Implementation shortfall statistics

It is very important to properly account for execution costs when backtesting. If costs are underestimated, one ends up giving the strategy advantages it would not enjoy in live trading. The backtest's results will be inflated and unreliable. Some measurements to assess whether the backtest properly accounts for execution costs are:

Broker fees per turnover: The fees paid to the broker (including exchange fees) in order to turn the portfolio over. Turning the portfolio over means to liquidate all current positions and enter new ones.

Return on execution costs: The ratio between net portfolio earnings and total execution costs. The larger this ratio is, the better. A high return on execution costs signifies that the portfolio will still make money despite worse than expected execution costs.

Broker fees per stock: The average dollar amount of fees per stock bought and sold.

Broker fees per dollar: The average dollar amount of fees per dollar involved in entering and exiting trades.

2.8.4.5 Classification Scores

In the context of applying ML in Automated Trading Strategies, it is useful to measure the performance of the ML algorithms in isolation. In this project, the investment decision process is informed by two classification models. Useful statistics to evaluate classification models are accuracy, precision, recall, and F1-score. See section 2.3.6 for a description of these metrics.

2.8.5 Benchmarks

It is common to contrast the performance of a strategy against a benchmark. A benchmark is a standard of performance an investment strategy, fund or money manager can be compared against. Comparing portfolio returns to a benchmark is a way to measure the value added by the portfolio manager or strategy. The benchmark most appropriate to compare a strategy against depends on the strategy's asset allocation and the type of securities it trades. If the strategy trades US bonds, the Barclays Capital U.S. Aggregate Bond Index is more appropriate than a Lipper index, which is suited to benchmark mutual funds. Strategies trading U.S. equities are popularly benchmarked against the Standard & Poor's 500 (S&P500) index or the Dow Jones Industrial Average. The S&P500 index is a market capitalization weighted index of the 500 largest publicly traded companies in the U.S. This makes it a good proxy for large-cap U.S. equity market performance. This project uses the S&P500 index to benchmark the developed ATS.

It is also relevant to have benchmarks in the context of return forecasting. A widely used benchmark, in this respect, is the historical average forecast, which assumes constant expected excess returns equal to the historical average market return:

$$\bar{r}_{T+1} = \frac{1}{T} \sum_{t=1}^T r_t$$

Where r_t is the market average return at time t (over the appropriate time period).

3 Methodology

The project's main aim is to investigate how machine learning can be used in automated trading systems by building and backtesting such software. A secondary aim is to run regressions on monthly equity risk premiums. To achieve these goals, the project's methodology is divided into four main tasks: dataset development, stock return prediction, development of an ATS and, backtesting. The following sections will go over each of these four steps.

3.1 Task 1: Dataset Development

The foundation of any machine learning models is the data it is trained and tested on. To gather a broad set of predictive features, several major reviews of the factor literature and replication studies were consulted to gather 86 factors. Most notably:

- The Remarkable Multidimensionality in the Cross-Section of Expected U.S. Stock Returns (Green, Hand and Zhang, 2013)
- Replicating Anomalies (Hou, Xue and Zhang, 2017)
- Fundamental Analysis and the Cross-Section of Stock Returns: A Data-Mining Approach (Yan and Zheng, 2016)

Little effort was invested in discriminating between the predictive power of different features before or during the development of the dataset. The through was to include as many features as time and resources allowed and let the ML models find whatever signal they could. Most ML algorithms are able to ignore the presence of uninformative features, making their adverse effects on model training negligible. If uninformative features should prove to be a problem, they could easily be excluded before model training.

All factors were produced from datasets available in the "Core US Equities Bundle" (2019) delivered by Sharadar Co. These are referred to as "source datasets" going forward. The source datasets included fundamental data from approximately 14,000 companies and spanned 21 years from January 1998 to February 2019. Pricing, volume, and dividend data were available for 16,000 companies, covering the same period. The 14,000 companies included 5000 listed companies and 9000 delisted companies. The datasets were largely void of survivorship bias, making them suitable for backtesting.

In short, the type of data available for feature construction was:

- Fundamental data from 10-K and 10-Q filings
- Some popular financial ratios, like book-to-market, price-to-sales
- Daily price data (open, high, low, close) adjusted for splits (not dividends)
- Dividends paid per share reported on ex-dividend dates
- Daily volume for each stock
- Company related events, such as bankruptcies reported at the date of filing with the SEC.

The sheer size of the datasets being handed and the fact that feature calculations often required data from multiple source datasets, covering different time ranges and sampling frequencies, made dataset development a demanding process. Computation of features relied heavily on multiprocessing. Implementation details are not covered in the report, but the source code can be consulted for this information (Didrik Fleischer, 2019, github.com/DidrikF/automated-trading-system).

A big selling point to use the “Core Us Equities Bundle” from Sharadar was that it also included fundamental data reported in a trailing twelve months (TTM) format. It is generally desirable to always work with the most up to date information. If only data from 10-K reports were used with monthly sampling, up to twelve consecutive samples would use the same data, increasing serial correlation. By using TTM data, fundamentals would be updated every quarter, ensuring that at most 3 samples included the same feature data. Even if it were 10 months since a company last filed a 10-K report, the company may have released up to three 10-Q filings over that period. Not including this new information would cause the dataset’s picture of the market to be severely out of sync with its current state. As a guiding principle, one should strive to construct the most informative features for the phenomenon being modelled. To this end, using outdated data is counterproductive. Therefore, all features based on yearly fundamental data (revenue, assets, short term liabilities, etc.) was calculated using TTM data. Features based on quarterly data was of course calculated using information from the latest 10-Q filings. Features based on pricing, volume, or dividend data was calculated relative to the sampling date. Most of the features in the dataset are based on data points covering one year. Using yearly data is advisable, as noise from seasonal changes are eliminated.

The dataset is divided into a training and test set. The training set spans the period 1998-01-01 to 2012-01-01 and includes 8068 stocks. The test set spans the period 2012-03-01 to 2019-02-01 and includes 6605 stocks.

3.1.1 Feature Calculations

The reader is referred to Appendix A for the complete list of features, their source paper, description, and implementation formula.

Details not covered in Appendix A, but still relevant for how the features were calculated is treated below.

Dividend adjusting prices

Features involving price data is calculated based on split and dividend adjusted prices. The source data from Sharadar’s SEP dataset is adjusted for splits, but not dividends. The price series were therefore adjusted for dividends by the following transformation:

$$F_i = \frac{\text{closePrice}_i + \text{dividend}_{\text{per share},n}}{\text{closePrice}_i}$$

$$\text{compounding}F_n = 1 * \prod_{i=1}^n F_i$$

$$\text{adjustedPrice}_n = \frac{\text{unadjustedPrice}_n}{\text{compounding}F_n}$$

Where F_i is the adjustment factor at the day i days earlier than the last known price for the stock. n is the number of days prior to the day of the last known price for the stock which the adjusted price is being calculated for.

The price series is adjusted backward, meaning that all prices will be relative to the last known price of the particular stock.

Dealing with missing data

Some feature values could not be calculated, using the available data from Sharadar’s core U.S. equities bundle, because of missing data or the data not lending itself to certain calculations (e.g.,

percent change in taxes when the company had not paid any). Approximately 5.25% of the feature values were missing after initial calculations. Rather than dropping rows with missing values, which would have led to a significant loss of samples, the missing values were replaced by synthetically produced values. The approach taken was to fill in as many values as possible based on random draws from a normal distribution with the mean and standard deviation equal to that of companies in the same industry with similar size. If enough data was not available, the normal distribution was constructed to have a mean and standard deviation like companies only of similar size.

Industry Classification

The industry classification scheme used when calculating industry adjusted features is Sharadar's own scheme based on SIC codes in a format which approximates the Global Industry Classification Standard (GICS).

Feature calculation and sampling

All the dataset's features are calculated relative to the sampled dates. For example, the one-month momentum of Apple at the 2012-07-21 sample is the return from 2012-06-21 to 2012-07-21, and the book-to-market ratio is calculated from data released in the 2012-06-21 10-Q filing (which is the latest at 2012-07-21).

Other Remarks

- All features in the dataset require at most two years of data.
- The dataset only includes firm-specific features and a couple of industry-specific characteristics. Macroeconomic variables are not included.
- Industry classification is also not included in the dataset.

3.1.2 Sampling

Another important aspect of how the dataset was produced has to do with how the data was sampled. The models developed for this project were concerned with prediction over a monthly horizon. To avoid excess overlapping between labels and generate as many samples as possible, the sampling process aimed at generating monthly samples for each company but at the same time set an upper limit for the allowed overlap. Remember that features based on pricing, dividend and volume data could be calculated at daily frequency, and features based on fundamental data could, at most, be calculated every fiscal quarter. This leads to another complexity of company fundamentals, the fact that they are reported according to each company's fiscal year. Fiscal years differs from company to company, causing reports to be filed at irregular dates. If sampling should occur close to the release of financial statements, one must account for the individual filing dates of each company. A more common approach in the literature is to sample at standardized dates for all firms, for example, at the end of every month, and use the most up-to-date information available at that time. The problem of this approach is that feature data will be unnecessarily out of date. Another complexity of financial statements is that firms sometimes make mistakes and issue updated reports. It is desirable to use the updated information once it gets released.

The idea behind the sampling procedure developed for this project came from "event-based sampling" and "information-driven bars" (López de Prado, 2018), where sampling is done when some informative event occurs, or new information is present in the market. There are many ways to measure the presence of new information; in this case, the release of a 10-Q or 10-K seemed appropriate. In short, the sampling scheme works as follows:

Each company has its observations sampled in isolation. Observations are sampled at a monthly frequency as close as possible to the latest filing of a 10-K or 10-Q report. If a new 10-Q is filed, then a new sample is made immediately. If it is less than 20 days since the last sample was made, then discard the last sample. This avoids excessive label overlap. If a new 10-Q is filed and it is more than 20 days since the last sample was made, then keep the last sample. This avoids dropping too much information in the sampling process. After the filing of a new 10-Q or 10-K report, sampling is done at a monthly frequency relative to this latest filing date. Figure 4 illustrates the sampling procedure.

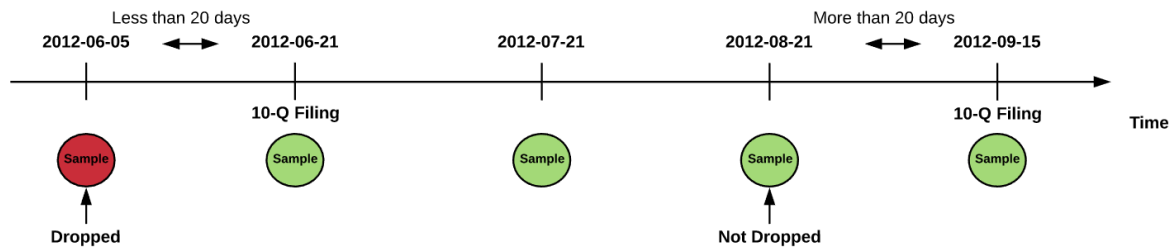


Figure 4: Sampling Scheme Illustration

Allowing labels to overlap is not uncommon but can have some negative effects if not taken into consideration. The problem with overlapping labels is that samples are no longer independent and identically distributed (IID). Not using independent samples causes some periods to be weighed more than others in training, causing the model to be overfitted to these periods. However, this scheme limits overlap considerably and should have limited adverse effects on the final models.

3.1.3 Labeling

Once the data was sampled and features calculated, the dataset was labeled for model training. The labeling procedure depended on what problem the model should solve.

3.1.3.1 Labeling for Regressions on Monthly Equity Risk Premia

When regressing features on monthly equity risk premia, the labels used for training was the excess monthly percentage return over the risk-free rate. The risk-free rate is derived from transforming the three-month treasury rate (secondary market rate, obtained from the Federal Reserve Bank of St. Louis) into a monthly rate, by the following formula:

$$Rf_t^{1m} = \frac{Rf_t^{3m}}{3}$$

Where t is the sampling date, Rf_t^{1m} is the monthly treasury rate and Rf_t^{3m} is the three-month treasury rate.

Over the course of a month, the three-month treasury rate will change, by assuming the rate at the beginning of the sample period is representative for the ensuing 30-days is inaccurate but should not introduce too much error. Further, the monthly rate is assumed not to be compounding; otherwise, the transformation to monthly rate is not correct.

Return is calculated from closing prices that have been adjusted for splits and dividends. An example of a label is 0.035, which means the stock appreciated 3.5% over the month in excess of the risk-free rate after adjustments for splits and dividends. Return labels were calculated by:

$$r_{i,t+1} = \frac{p_{i,t+1}}{p_{i,t}} - Rf_t^{1m}$$

Where $r_{i,t+1}$ is the percentage return of stock i at the end of month $t+1$, $p_{i,t+1}$ is the closing price at the end of month $t+1$, $p_{i,t}$ is the closing price for stock i at the end of month t (see 3.2 for further explanation of the time notation).

3.1.3.2 Labeling for ATS classification Models

As discussed in section 2.5.3, the labeling method most common in the academic literature (fixed time horizon labeling) does not account for the path followed by prices. This means that situations where a position would be stopped out in live trading, either due to reaching a stop-loss or take-profit limit, is not considered when training or testing the algorithm. To address this issue, López de Prado (2018) developed the triple barrier method. The idea is to label the samples according to which out of three barriers is touched by the price series first. Two horizontal and one vertical barrier is set, such that the upper horizontal barrier sets the take-profit limit, the lower horizontal barrier sets the stop-loss limit, and the vertical barrier sets the timeout for the hypothetical trade (each sample is the source of potential trade). The labeling procedure is thus analogous of how a trade would be handled in the real world.

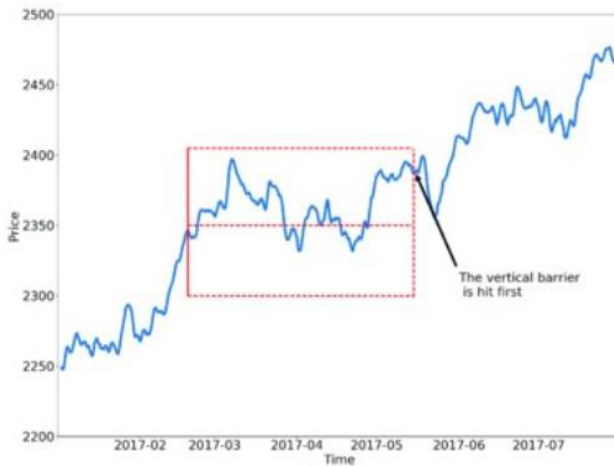


Figure 5: Triple Barrier Method (López de Prado, 2018, chapter 3)

Figure 5 visualizes the triple barrier method. In this case, the vertical barrier is hit first, and because the price has increased since the start date, the sample will be labeled with 1 (in the case of a long trade). In the case where the upper or lower barrier is struck first, the label would be 1 or -1 respectively.

What constitutes the take-profit and stop-loss limit, of course, depends on the direction of the trade. In the case of a long position, a price increase means the trade is in the money, and the upper barrier represents the take-profit limit while the lower barrier represents the stop-loss limit. In the case of a short position, a price increase drives the trade out of the money, and the upper barrier represents the stop-loss limit while the lower barrier represents the take-profit limit. When it is not possible to differentiate between take-profit and stop-loss limits, the horizontal barriers must be configured symmetrically as it is impossible to differentiate them.

Another limitation of fixed time horizon labeling is that it does not consider past volatility. The triple barrier method, on the other hand, use measurements of past volatility to set the width of the horizontal barriers. Labeling in this project used an exponentially weighted standard deviation (EWSD) of monthly returns, the past 24 months, to scale the width of the horizontal barriers.

The triple barrier method is a flexible device and can be used to label datasets to learn both the side and size of trades. The process is somewhat difficult to grasp, but the following figure should be helpful.

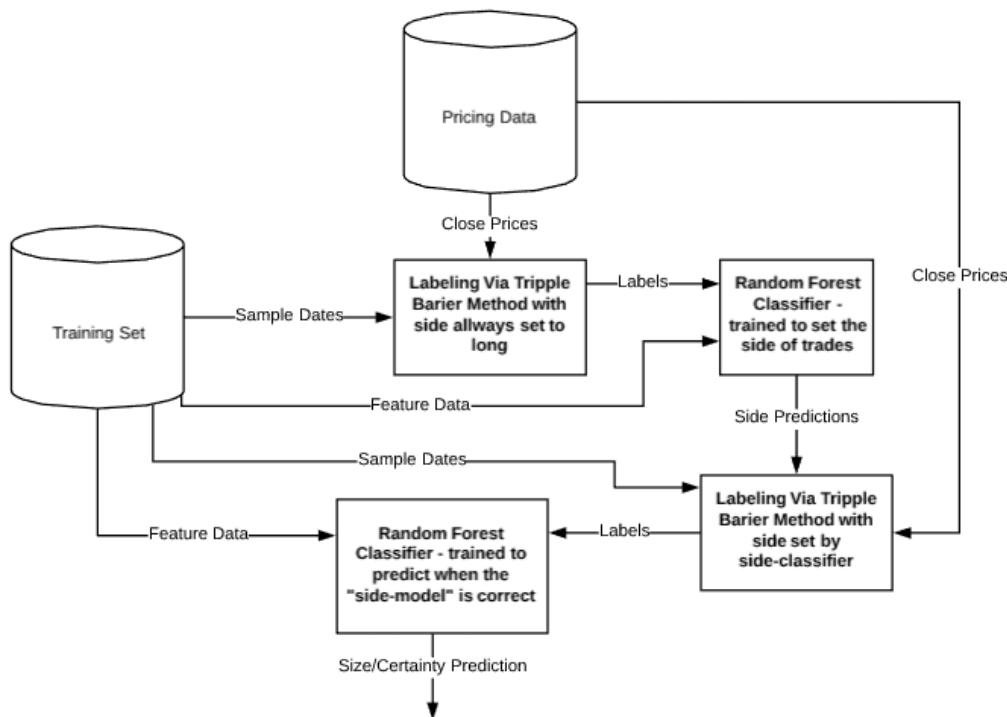


Figure 6: ATS ML model training procedure using the triple barrier method

Figure 4 illustrates how two ML models for this project were trained to learn how to set the side and size of trades. The training set contains features sampled at an approximately monthly interval (see section 3.1.2) for each company. The sample dates represent when a hypothetical trade would have been initiated. Without a model to set the side of trades, labels must be constructed for this purpose. This is done by assuming that all “trades” take long positions and use horizontal barriers with a width equal to 100% of the EWSD of the past 24 monthly returns. The vertical barrier is set one month after the sample date. Close prices are used to derive a series of returns relative to the sampling date. The sign of the return at the time of the first barrier being touched constitutes the label for the sample.

This process is repeated for every sample of every stock. The resulting labels are then used to train a classifier to make predictions to set the side of trades.

Once a model is built to set the side of trades a second sweep over the training samples (with TBM) is done, but this time the side of trades are set by the side-classifier. This process was coined “meta-labeling” by López de Prado (2018), who developed the method. During this second sweep, the horizontal barriers can be set asymmetrically as the side of trades are now informed by the primary model. It is thus possible to make the process more sensitive to price movements in a certain direction. For example, if the investor using the model is very risk averse, he might prefer to set the stop-loss limit closer to the start price than the take-profit limit. Meta-labeling was conducted, in this project, with stop-loss limits set to 80% of the EWSD of the past 24 monthly returns.

Meta-labeling labels the samples such that if following the advice of the primary model would have yielded a profit, the sample is labeled 1. If the advice would have resulted in a loss, the sample is labeled 0. The second model will, as a result, learn which circumstances lead the primary model being correct and which circumstances lead to the primary model being incorrect. The probability the classifier assigns to the 1-class can be used in further calculations to set the size of bets.

3.2 Task 2: Stock return prediction

The problem of equity risk premium prediction can be expressed as best approximating the conditional expectation $E(r_{i,t+1}|X_t)$, where $r_{i,t+1}$ is the return of stock i in excess of the risk-free rate from time t to $t+1$ and X_t is the true and unobservable information set of all market participants (Gu, Kelly and Xiu, 2018). The fact that the best model and regressors for the problem are both unknown, make ML algorithms particularly useful in finding a good solution. Four ML models was investigated; simple linear regression, principal component regression, random forest, and deep neural net. Parameter estimation for all algorithms is done using the basic objective of minimizing mean squared prediction error (MSE). It is worth noting that there are a plethora of machine learning algorithms, and any single algorithm can have many different implementations. The descriptions of all algorithms will, therefore, be quite terse. As all ML algorithms used in this project are implemented by external parties, the reader is referred to the documentation of the respective third-party packages for further reading regarding the technical details of algorithm implementations (see references).

The general form the prediction problem is an additive prediction error model:

$$r_{i,t+1} = E_t(r_{i,t+1}) + \varepsilon_{i,t+1}$$

Where

$$E_t(r_{i,t+1}) = f^*(x_{i,t})$$

The notation followed is:

- Stocks are indexed as $i = 1, \dots, N$.
- Months are indexed by $t = 1, \dots, T$, but note that is a simplification. In reality, each stock has their monthly samples constructed relative to the last form 10-Q/10-K filing and can have some overlap in their labels. This misaligns month 1 of stock i and month 1 of another stock j . Despite this, each sample's return always spans one month. Therefore, another way to interpret the notation is that t is the date of sampling of feature vector $x_{i,t}$ for stock i and $t+1$ is the date one month later.
- $r_{i,t+1}$ is the equity risk premium of stock i from time t to time $t+1$.
- $E_t(r_{i,t+1})$ is the expected value of $r_{i,t+1}$ at time t .
- $\varepsilon_{i,t+1}$ is an error term.
- $x_{i,t}$ is a vector of features for stock i , sampled at time t .
- f^* is a flexible function of feature vector $x_{i,t}$, which is shaped to best approximate the true expectation of the equity risk premium of stock $i = 1, \dots, N$ at time $t = 1, \dots, T$.

Note that f^* is not a function of i or t , meaning that the same function is constructed for all stocks across all time. Also, the function f^* only depends on x through $x_{i,t}$, which means that no information prior to time t for stock i (not captured in a predictor variable in $x_{i,t}$) or samples for any other stock j is used in making a prediction. It is common in the literature to re-estimate the model

multiple times over the testing period as more data becomes available. This approach is not taken here in favor of leveraging information from the entire panel to build a single model and not re-estimate as more data becomes available. This decision has to do with time and resource limitations. Re-estimating the model on, e.g., a yearly basis would very likely lead to better results.

The available equity universe consists of 14000 stocks and spans the timeline from 1998-01-01 to 2019-02-01. The period is split into two. A training and testing period, where the training period spans from 1998-01-01 to 2012-01-01 and the testing period spans from 2012-03-01 to 2019-02-01. The three-month buffer is inserted to eliminate any information leakage. See section 3.1.3 for a description of the labeling procedure.

3.2.1 Sample Splitting and Cross-Validation

When training the random forest regressor and the deep neural net regressor, hyperparameter tuning is performed using purged K-fold cross-validation. As explained in section 2.5.6 on cross-validation in finance, without properly purging the validation set of observations that overlap with the testing set, performance metrics are inflated. Purged k-fold cross-validation is performed by dividing the training set into three equally sized folds. Each validation fold has all samples overlapping with the training folds removed, but no embargo was enforced to further separate the training and validation folds. By employing cross-validation, the temporal ordering of data in the training set was broken, but this is of no consequence if the validation folds are properly purged.

3.2.2 Performance Evaluation

Predictive performance of regression models is measured primarily by the out-of-sample R^2 :

$$R_{OOS}^2 = 1 - \frac{\sum_{(i,t) \in \tau_3} (\hat{r}_{i,t+1} - r_{i,t+1})^2}{\sum_{(i,t) \in \tau_3} r_{i,t+1}^2}$$

Where τ_3 represents the set of months in the test set. The specification of the R^2 above differs from the conventional specification. The denominator is the sum of squared equity risk premiums without subtracting the mean equity risk premium (also interpretable as the mean excess market return). The reason for this is that the historical mean market excess return is so noisy that it lowers the bar good forecasting performance. By benchmarking the R_{OOS}^2 against a mean excess market return of zero, the problem is avoided. In fact, predicting future equity risk premiums based on a historical average forecast typically underperforms a forecast of zero (Gu, Kelly and Xiu, 2018). Note that the above remarks are meant to apply when predicting stock returns for individual firms. Subtracting the mean excess market return in the denominator is more meaningful when predicting market aggregate excess returns. The R_{OOS}^2 formula and rationale for its use were adopted from the paper “Empirical Asset Pricing via Machine Learning” (Gu, Kelly and Xiu, 2018).

Other measures used to assess model performance is the mean squared error (MSE), the mean absolute error (MAE) and the traditional R_{OOS}^2 .

3.2.3 Multiple Linear Regression

The first model description presented in this report is the least complex one; the multiple linear predictive regression model estimated via ordinary least squares (OLS). The motivation for including this over arguably more promising models is that it will illustrate the advantage, if any, of more advanced ML algorithms in that they allow more complex interactions and non-linearities to manifest between predictors.

In vectorized form, a multiple linear regression takes the following form:

$$f^*(x_{i,t}; \theta) = x'_{i,t} * \theta$$

The objective function; OLS, is expressed as:

$$\mathcal{L}(\theta) = \frac{1}{N * T} \sum_{i=1}^{N'} \sum_{t=1}^{T'} (f^*(x_{i,t}; \theta) - r_{i,t+1})^2$$

Where N' and T' is the last stock and month of the training set.

As the objective function is sensitive to the size of the features of $x_{i,t}$, the dataset's features were standardized before model estimation. This process rescales features to exhibit the properties of a standard normal distribution.

Scikit-learn's (v.0.20.3) linear regression implementation was used without any modifications or notable configuration options. The exact class used was `linear_model.LinearRegression`.

3.2.4 Principal Component Regression

The training procedure used was to first standardize the dataset, then condense the dataset down to five principal components and finally to fit a linear regression model to minimize an OLS objective function.

In vectorized form, the regression model is defined as:

$$f^*(k_{i,t}; \theta) = k'_{i,t} * \theta$$

Where $k_{i,t}$ is the vector of principal component values for stock i at time t .

The OLS objective function is express as:

$$\mathcal{L}(\theta) = \frac{1}{N * T} \sum_{i=1}^{N'} \sum_{t=1}^{T'} (f^*(k_{i,t}; \theta) - r_{i,t+1})^2$$

Scikit-learn's (v.0.20.3) linear regression implementation was used without any modifications or notable configuration options. The exact class used was `linear_model.LinearRegression`.

Further, `decomposition.PCA` was used to perform PCA. Only default arguments were used, except `n_components` equal to 5 for PCA.

3.2.5 Random Forest Regression

The random forest regression model was trained by running purged k-fold cross validation where the following hyperparameter space was searched:

- Number of estimators: 200, 500, 1000
- Minimum fraction of samples in leaf nodes: 5%, 10%, and 20%
- Maximum number of features: 5, 10, 15

The proper hyperparameter space to search is difficult to know without experience. To make computation times manageable, while still being able to conduct a fairly wide search, each parameter value was set quite far apart. Some general recommendations made by López de Prado (2018) was followed. When it comes to forest size, López de Prado suggests 1000 trees as a good starting point. It was decided that 200, 500, and 1000 estimators would give a good indication of how performance would vary with forest size. Keeping forest size on the lower side was inspired by preliminary testing, where it was discovered that smaller forests did equally well, if not better than large forests.

The minimum fraction of samples required in leaf nodes (`min_weight_fraction_leaf` in Scikit-Learn's

random forest implementation) is configured quite high, López de Pardo used 5% as an example. It is important this parameter is set quite high to ensure trees are regularized sufficiently and out-of-bag accuracy would converge to out-of-sample (k-fold) accuracy. Early in-sample testing showed a tendency to prefer even higher values for `min_weight_fraction_leaf`, which is why it was decided to go as high as 20%.

The default parameter for the maximum number of features to use when training each tree in Scikit-learn's implementation was the square root of the number of features in the dataset. In the case of 86 features, this would equal 9.3. This was used as the basis for the search. A lower number of features should lead to less correlated trees, while a higher number of features should lead to each tree learning more. Five and fifteen were chosen as alternatives to the default parameter value of approximately ten.

The random forest regressor was further configured to apply bootstrapping in the construction of training samples and use MSE to measure the quality of splits. To eliminate the effects of unbalanced classes, each tree was trained on samples weighted to yield equal class representation ("`class_weight`" decision tree parameter equal to "`balanced_subsample`"). The hyperparameter space was quite restricted to reduce processing times. A wider search would likely obtain better performance.

The best hyperparameters, according to the mean MSE of the model on the validation folds, were then used to train the final model on the whole training set.

Scikit-learn's (v.0.20.3) "`ensemble.RandomForestRegressor`" class was used to build the random forest regression model.

3.2.6 Artificial Neural Network Regression

The developed ANN was a fully connected feed-forward deep neural net, consisting of an input layer of standardized features, one hidden layer of 10 neurons and finally an output layer consisting of a single neuron, yielding a single prediction for the monthly equity risk premiums. It was decided to keep the number of hidden layers low to avoid overfitting. The number of neurons in the hidden layer was based on the geometric pyramid rule, which recommends the hidden layer to have $\sqrt{n * m}$ nodes, where n and m is the number of input and output nodes, respectively.

Regularization techniques employed during training were:

- Learning rate shrinkage (introduced by the Adam optimizer)
- Early stopping
- Ensemble learning
- Dropout

A custom cross-validated grid search class was developed to perform hyper-parameter tuning. This class used purged k-fold cross-validation to split training and testing folds and averaged the MSE score for each validation fold to obtain the final score for each set of hyper-parameters.

The following hyper-parameters were searched:

- Initial learning rate: 0.1 and 0.01
- Lambda parameter of L2-regularization: 0.1, 0.5, 0.7
- Epochs: 100, 500 and 1000
- Patience: 2 and 5, 10
- Dropout: 0.2, 0.4, 0.5

ReLU was used as the activation function, and batch size was set to 100000. The loss function the optimizer was set to minimize was the penalized L2 objective function of prediction errors (mean squared error). The optimizer used was the Adam optimizer.

Note that it is common to train on batches of size 32, 64 or 128, but due to the high variance and noise of the dataset it is likely that adjusting weights based on such a small number of samples would yield too imprecise changes in network parameters, ultimately making it difficult for the loss function to converge on a minimum.

Once the optimal hyper-parameters had been identified an ensemble of ten neural nets were trained with these parameters. The prediction of the ensemble was defined to be the mean of the predictions of the individual neural nets.

The neural nets were built using TensorFlow (v. 1.13.1) and the Keras API.

3.2.7 Random Forest Classifier for Equity Risk Premiums

In addition to the regression models, a classification model for the sign of monthly equity risk premiums was built. The model was trained by running purged k-fold cross-validation, where the following hyperparameter space was searched:

- Number of estimators: 100, 200, 500
- Minimum fraction of samples in leaf nodes: 5%, 10%, and 20%
- Maximum number of features: 3, 5, 7

See section 3.2.5 for the rationale behind the chosen hyperparameter space. The random forest classifier was further configured to apply bootstrapping in the construction of training samples and use entropy to measure the quality of splits. To eliminate the effects of unbalanced classes, each tree was trained on samples weighted to yield equal class representation (“class_weight” equal to “balanced_subsample”).

The best hyperparameters, according to the mean F_1 score of the model on the validation folds were then used to train the final model on the whole training set.

Scikit-learn’s (v.0.20.3) “ensemble.RandomForestClassifier” class was used to build the classification model.

The advantage of building this classification model is that it produced easily interpretable results that lend itself to simple statistical testing. An interesting question is whether in the simplest formulation of the problem (binary classification); could an advanced ML algorithm learn from the training set to reliably outperform a random model on the testing set. The idea was to run t-tests on the mean accuracy of the ML model compared to a random model with an accuracy of 0.5. If the null hypothesis,

H_0 : The mean accuracy of the ML model is equal to the mean accuracy of a random model,

was refuted with high statistical significance, it would lead to the conclusion that the ML algorithm had, in fact, learned something from the training data and was able to reliably outperform a random model on out-of-sample data.

3.2.8 Statistical Significance of Model Accuracy

To test whether the equity risk premium classifier was able to outperform a random model, the following test procedure was followed:

The test set consists of approximately 350,000 samples. Each sample consisting of a vector of features and a label equal to the sign of the equity risk premium for the month following the date of sampling. The classifier (trained on the training set) makes predictions based on the vectors of features, unaware of the true outcomes. This produces a series of predictions where each is either 1 (positive excess return over the month following the sampling date) or -1 (negative or zero return over the month following the sampling date). By comparing the series of predictions with the series of the true outcomes, element by element, a binary series was constructed such that 1 indicates correct prediction and 0 indicates wrong prediction. An element in this series is noted as b_k , where k is the index of the element. A sample of ML model accuracy was constructed by taking 200 random samples, each with a size of 3000 elements, from the binary series. The accuracy of sample i is:

$$accuracy_i = \frac{\sum_{k=1}^{3000} b_k}{3000} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

The samples of model accuracy A_1, A_2, \dots, A_n represents a random sample from what is assumed to be a normal distribution with unknown μ and σ^2 . Then the random variable $\sqrt{n} \frac{(\bar{A} - \mu)}{S}$ has a student t-distribution with $n-1$ degrees of freedom. Where S is the sample standard deviation. To test whether the mean accuracy of the ML model is better than a random model, a one-sided single sample t-test was conducted. The null hypothesis is that the mean accuracy is 0.5 (random model). The alternative hypothesis is that the mean accuracy of the ML model is greater than 0.5.

$$H_0: \mu = 0.5$$

$$H_1: \mu > 0.5$$

The hypothesis is tested a significance level (α) of 0.05. The t-statistic is defined as follows:

$$t = \frac{\bar{a} - 0.5}{s/\sqrt{n}}$$

A T-value above the critical value corresponding to $(1 - \alpha) = 0.95$ and degrees of freedom equal to 199, will cause rejection of the null hypothesis in favor of the alternative hypothesis.

3.3 Task 3: Developing an Automated Trading System

This section will outline the implementation of the automated trading system. Put shortly, the job of an ATS is to take information as input and generate orders based on that information to optimize risk-adjusted returns given the investor's preferences. The ATS built for this thesis uses ML model predictions to derive what stocks to invest in and how to allocate funds between them. The ATS aims to extract as much predictive ability from the ML models as possible and translate this into returns while avoiding excessive risks. No effort was made to train different models for different economic regimes, industries, or other categorizations. Further, no efforts were made to model market uncertainty to adjust the portfolios level of investment in the market dynamically. The system is built to continually strive towards being fully invested in the market and hold the maximum allowed number of positions.

The system only supports market orders, which means the broker will fill the order for the specified stock and quantity at the best available price at the time of submission. Further, each order generated by the ATS specifies timeout, stop-loss, and take-profit limits.

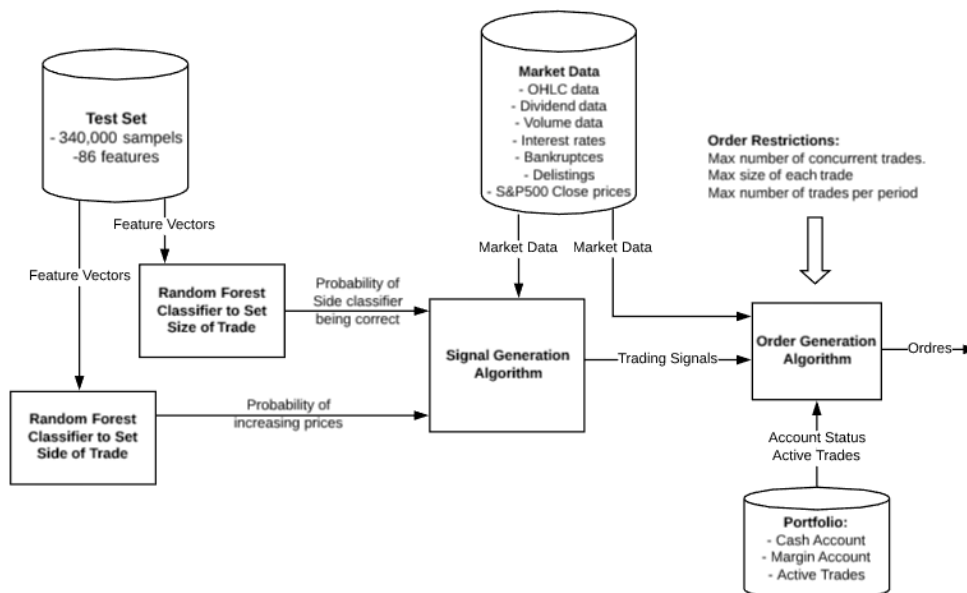


Figure 7: Automated Trading System Architecture

Figure 7 illustrates the overall structure of the system and its major components. The major components of the ATS are:

- A dataset of over 340,000 samples spanning over 8000 companies and 7 years (2012-03-01 to 2019-02-01), each with 86 features.
- A classification model predicting the sign of the return at the first barrier touch (see Triple Barrier Method, section 3.1.4.2). The output of this model is used to set the side of trades. This model is henceforth called the primary model or side classifier.
- A classification model predicting whether following the primary model's advice will yield a positive or negative return. This classification model is henceforth called "the secondary model" or "the certainty model." Further, the probability of positive return is from here on referred to as the level of certainty the secondary model has in a trade's ability to generate positive returns (certainty prediction).
- An algorithm generating trading signals from ML model predictions
- An algorithm using trading signals to produce orders under a set of restrictions

In short, the system operates as follows:

The primary and secondary ML models make predictions based on samples from the test set. The predictions are used by a signal generation algorithm to generate trading signals. The Trading signals are fed to an order generation algorithm, which produces market orders for the best trading signals with the aim of maximizing the level of investment in the market and maintain the maximum allowed number of trades. The order generation algorithm is also subject to other restrictions, which will be explained later. The resulting orders are given to a broker object, which simulates their execution and maintenance.

3.3.1 Developing ML Models

The ATS is built on top of two classification models. One model to set the side of trades and one model to derive the size of trades. See section 3.1.3.2 for a description of the training process of the two classifiers.

Both classifiers are built using random forests. For the side classifier, purged k-fold cross-validation was used to search the following hyperparameter space:

- Number of estimators: 100, 200, 500
- Minimum fraction of samples in leaf nodes: 5%, 10%, and 20%
- Maximum number of features: 5, 10, 15

The hyperparameters yielding the highest F_1 score was selected. The certainty classifier used the best hyperparameters found for the side classifier. Both classifiers were trained on the whole training set once the hyper-parameters were decided.

Both classifiers were further configured to apply bootstrapping in the construction of training samples and use entropy to measure the quality of splits. To eliminate the effects of unbalanced classes, each tree was trained on samples weighted to yield equal class representation (*"class_weight"* equal to *"balanced_subsample"*).

Scikit-learn's (v.0.20.3) *"ensemble.RandomForestClassifier"* class was used to build the side and certainty classifiers.

3.3.2 Signal Generation Algorithm

Predictions made on samples from the dataset form the foundation of trading signals. Trading signals are only produced for the samples for which the secondary model has the greatest level of certainty. This allows the early exclusion of irrelevant samples, which reduces processing needs. A trading signal consists of the predicted side of future return $\{-1, 1\}$, the certainty of the secondary model $[0, 1]$, the EWSD of the past 24 monthly returns, the sampling date and the ticker it concerns. Each signal can inform at most one trade. Once a signal has been acted upon, it cannot be used to generate orders in the future.

3.3.3 Order Generation Algorithm

The order generation algorithm receives multiple signals and produce zero, one or several orders depending on given restrictions and the current state of the portfolio. An order is specified by the following parameters: ticker, amount (number of stocks, where negative values indicate short position), date of creation, take-profit limit, stop-loss limit, and a timeout.

Order generation algorithm works as follows:

1. Determine the number of orders to generate, by:

$$\text{num}_{\text{orders}} = \min(\text{opp}_{\text{max}}, \text{p}_{\text{max}} - \text{num}_{\text{trades}})$$

2. Determine the amount to be invested:

$$\text{availableFunds} = \min(\text{pv} * \text{fipp}_{\text{max}}, \text{cash} - \text{cash}_{\text{min}})$$

3. Get the top $\text{num}_{\text{orders}}$ number of signals to generate orders from. The ranking is done with respect to certainty levels. The resulting signals are referred to as “the top signals”.
4. Determine the allocation of funds between signals:

Given top signals: S_1, S_2, \dots, S_n with certainty levels c_1, c_2, \dots, c_n the amount of funds to allocate to signal x is:

$$\text{orderSize}_x = \text{availableFunds} * \frac{c_x - 0.5}{\sum_{i=1}^n (c_i - 0.5)}$$

Doing this type of weighted average tilt allocations more towards signals with higher certainties compared to a direct weighted average, where 0.5 is not subtracted from c_i . It is known that signals who reach this step have a certainty level above 0.5, if this were not the case, orderSize_x could become negative.

In the case of a short trade, only $(\text{orderSize}_x / (1 + \text{marginRequirement}_{\text{initial}}))$ dollars would be the actual size of the trade as orderSize_x should cover the initial margin requirement as well as the market value of the position.

5. For each top signal, x :
 - a. Calculate the number of stocks to order given the calculated order size for x :

$$\text{orderAmount}_x = \min(\text{roundDown}(\text{volume}_{i,d-1} * \text{volumeLimit}), \text{roundDown}(\text{orderSize}_x * \text{price}_i))$$

- b. Calculate the take-profit, stop-loss and timeout limit for x :

$$\text{takeProfit}_x = \text{EWSD}_{t-24,t,i} * \text{scale}_{\text{tp}}$$

$$\text{stopLoss}_x = \text{EWSD}_{t-24,t,i} * \text{scale}_{\text{sl}}$$

$$\text{timeout}_x = \text{order generation date} + \text{one month}$$

- c. Generate order object for x

6. Return list of order objects

Notation:

Stock i is the stock signal x concerns,

opp_{max} is the maximum number of orders that are allowed per period,

p_{max} is the maximum number of concurrent positions that are allowed,

$\text{num}_{\text{trades}}$ is the current number of active trades,

pv is the current portfolio value (margin account + cash balance + market value of trades),

fipp_{max} is the maximum fraction of the portfolio value that can be invested in a period,

cash is the current cash balance,

cash_{min} is the minimum allowed cash balance after conducting trades,

$marginRequirement_{initial}$ is the initial margin requirement enforced by the broker,
 $price_i$ is the open price of stock i ,
 $volume_{i,d-1}$ is the volume of the previous trading day for stock i ,
 $volumeLimit$ is the percent of the previous day's volume that can be purchased of any stock,
 $EWSD_{t-24,t,i}$ is the exponentially weighted standard deviation of monthly returns the past 24, months for the stock i ,
 $scale_{tp}$ and $scale_{sl}$ are parameters set by the strategy to scale the take-profit and stop-loss limits given $EWSD_{t-24,t,i}$. Examples are $scale_{tp}$ equal to 1 and $scale_{sl}$ equal to 0.5

The order generation algorithm is restricted in the following ways

- Signals need to be younger than a maximum allowed age (e.g., 7 days or less), this ensures the signals are used in accordance with the signal's predictions, which are made for the month following the sampling date (configurable).
- Signals must be at least 1 day old. This eliminates lookahead bias, as samples can depend on information released on the sampling date (e.g., close prices).
- The portfolio can have no more than a maximum number of active trades at any time, e.g., 20 (configurable).
- Trades cannot have a larger size than some percentage of current total portfolio value, e.g., 5% (configurable).
- Only one active trade per ticker allowed at the same time
- Not allowed to hold long and short positions in the same stock at the same time
- A maximum percentage of the total portfolio value can be invested each period, e.g., 33% (configurable)
- A maximum number of trades can be conducted in the same period, e.g., 10 (configurable)
- A minimum of percent of the portfolios original value must be held in cash, e.g., 1% (configurable).
- Orders must have a minimum size, e.g., \$10,000 (configurable).
- The amount available to purchase of any stock at any time is limited to a percent of the previous trading day's volume, e.g., 10% (configurable).

Other Remarks

- Diversification is solely ensured through the number of concurrent trades and trade size. Although it would be simple to ensure further diversification by limiting concurrent investment in the same industry, this was not implemented.
- Although it is not necessarily a contradiction to hold positions with the opposite side in the same stock at the same time (as long as trades span imperfectly overlapping time intervals), this possibility was excluded.
- The limitation on the amount that can be invested in any one period (week/day/month) was put in place to have the system gradually build up its portfolio, and thus limiting the reliance on the signals available in any one period.
- By maintaining a minimum balance, the portfolio would have funds to deal with unexpected future interest payments, transactions costs, or other expenses.
- Note that both the take-profit and stop-loss limits are expressed in terms of a total return percentage, including dividends.
- The decision of timing trades out after one month was made in order to align the usage of the ML models with how they were trained. To train a model to make predictions on monthly return series and then use this model to make predictions on different intervals, e.g., three months, would misalign application and what the models are optimized for.

- The ATS does not dynamically reallocate assets, meaning actively selling out of current positions for the purpose of entering others. All trades are held until completion. This simplifies system design.

3.4 Task 4: Backtesting

The backtester used in this project was custom built from scratch. This allowed customization of all aspects of the backtester and its operation to be tailored for the strategy under testing. It is vital that the backtester assumptions are consistent with the assumptions of the trading strategy. This can be difficult to achieve when using a pre-built system. To avoid the many pitfalls of backtesting, the following measures were taken to address each:

Data snooping: the dataset was split into a training and testing set with a three month period separating the two. Only the training set was used during the development of the strategy and backtester, and only the test set was available to the strategy and backtester when running simulations.

Survivorship Bias: the datasets used for backtesting did not exhibit survivorship bias (contained a representative amount of bankrupt companies). The investment strategy would have no access to bankruptcy data, allowing it to avoid companies with unfavorable futures. If a position is held in a company that declared bankruptcy, it is assumed that no proceeds are received.

Look-ahead bias: all market data and access to it is controlled by a single object. This object is the only source of truth for what date is current. All parts of the software must reference this single state variable when requesting information, keeping the system in sync.

Transaction costs: a popular online broker was referenced to obtain a realistic commission and transaction costs model (Interactive Brokers commission model as of 2019-04-01 was used).

Data quality: to ensure high data quality, a license was purchased for access to datasets developed by a professional vendor of financial data; Sharadar Co.

Capacity constraints: a finite pool of money is made available to the portfolio at the start of the backtest, and this amount is not extended at any point. Further, leveraging is not allowed, and trades are limited to 10% of the previous day's trading volume in any one stock.

Robustness: by having high turnover and limiting how much can be invested in a single trade, the strategy's returns are limited in how much they can rely on a few exceptionally profitable trades.

The backtester is based on daily pricing data (including open, high, low, and close prices), which is less optimal than data of higher frequency, e.g., minute bars. The data frequency and lack of access to market microstructural data limit the backtester's ability to reconstruct the market history accurately. As discussed earlier, all backtests are, at best, hypothetical and do not tell precisely how a strategy would have performed if conducted over the tested period. If a perfect order book was available, the fill and close prices of trades could be calculated more accurately, but without this information, one relies on making assumptions. These assumptions should not introduce any biases that would be favorable to the strategies under testing, as this would inflate performance measurements.

The following assumptions were made in the development of the backtester.

- New trades are only entered at the open.
- Orders are filled at the open price without suffering any slippage when 10% or less of the previous day's volume are being bought or sold.

- If the price declines over the day for a given asset, it is assumed the high is hit before the low (important for calculations around stop-loss and take-profit limits)
- If the price increase over the day for a given asset, it is assumed the low is hit before the high (important for calculations around stop-loss and take-profit limits)
- The initial maintenance requirement is 50%, and the maintenance margin requirement is 30%
- The annual margin interest is 6% and paid daily.
- Proceeds from dividends are received at the end of the ex-dividend date
- Interest payments and reception occurs at the end of each day at the daily risk-free rate
- Dividend payments are received at the end of the ex-dividend date
- Proceeds from exiting a trade are available immediately without delay (regardless, the proceeds would not be used until the next trading day)
- Proceeds from trades are free of taxes, and dividends are taxed at 25%

The major components of the backtester are illustrated in figure 8.

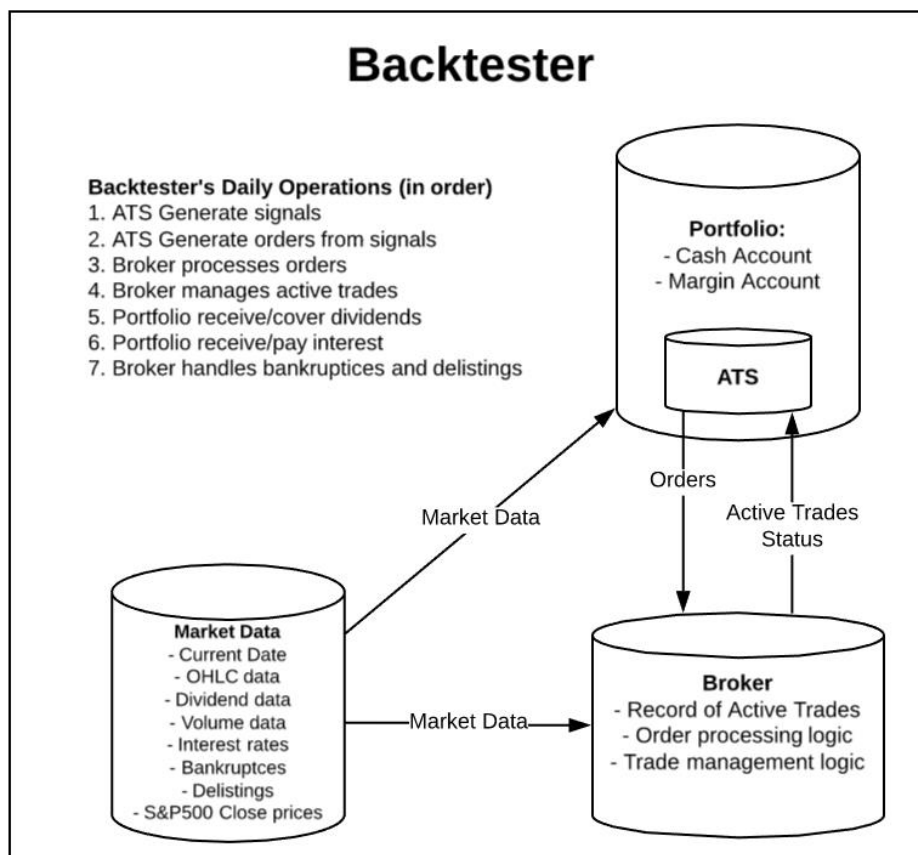


Figure 8: Backtester components and order of operations

The major components of the backtester are:

- A backtester object with an event loop, looping over each day between the backtester's start date and end date.
- A broker object with logic to handle:
 - o Order processing for long and short trades

- Trade management, which involves:
 - Detecting when stop-loss, take-profit or timeout limits are exceeded
 - Calculate close prices and trade returns
 - Give proceeds from closed trades to the portfolio's cash balance
- Logic to handle corporate actions like dividends, bankruptcies and delisting's
- Logic to calculate margin requirements and issue margin calls
- Logic to deal with interest payments on both cash and margin accounts.
- A portfolio object with properties representing the cash and margin account balance, logic to record all in -and outflows from each account and logic to calculate return and cost statistics.
- An object representing market data, which includes price data, interest rates, and corporate actions. This object also has a property which represents the current date of the backtest; all objects reference this property to stay in sync with the rest of the system. Only having a single point of reference for the backtest's current date helps avoid lookahead bias.

The backtester's event loop goes through the following main steps at each iteration, in order:

1. Create market-data-event, initiating the next day of the event loop.
2. If it is a business day:
 1. The portfolio/strategy generate trading signals (see ATS description for details)
 2. The portfolio/strategy generate orders based on the trading signals (see ATS description for details)
 3. The broker processes the orders by calculating commissions, fill price, and performs various checks.
 1. If all succeeds a trade-object is created and added to the broker's blotter
 2. If something fails, a canceled-order-object is created, which then can be handled by the portfolio/strategy if desired
 4. The broker manages the active trades by checking if any exit conditions are met. If they are, the respective trade is closed. Exit conditions include take-profit, stop-loss or timeout being exceeded, the company declaring bankruptcy, or the company being delisted.
 1. If take-profit was exceeded (verified by doing checks against the day's high and low prices), the close price of the trade is set to be the price which would make the total return (including dividends) equal to the take profit limit
 2. If stop-loss was exceeded the close price of the trade is set to be the price which would make the total return (including dividends) equal to the stop-loss limit.
 3. If the timeout was reached, the close price of the trade is set to the close price at the timeout date.
 4. If a company files for bankruptcy, the trade is closed at the end of the day when the bankruptcy was filed. The closing price is set to zero, meaning that the portfolio receives no proceeds from the liquidation.
 5. If the company is delisted, the close price of the trade is set to the close price of the stock on the date of delisting.
 5. The broker pays out any dividends received on long trades and claims payment to cover dividends on short trades.
3. Regardless if it is a business day or not:
 1. The broker pays out interest on margin and cash accounts equal to the risk-free rate (calculated as a daily compounding rate from the 3-month treasury rate)

2. The broker charges interest on negative balances on margin and/or cash accounts equal to the risk-free rate (calculated as a daily compounding rate from the 3-month treasury rate).

Note that the type of strategies the backtester is intended for cannot rely on leverage. Having this piece of logic is simply intended as a fail-safe to address the possibility of negative balances. However, this is highly improbable if a sufficiently large minimum balance is specified.

3. The broker charges interests on cash borrowed to enter short trades. The interest to pay per short trade is calculated by

$$\text{marginInterest} = \text{fillPrice} * |\text{orderAmount}| * r_{f\text{daily}},$$

where $\text{price}_{\text{fill}}$ is the fill price of the short trade and $r_{f\text{daily}}$ is the daily risk-free rate.

4. Various broker and portfolio state information are captured, including the currently active trades and the total value of the portfolio.

4. At completion:

1. Data that was generated during the backtest is captured and written to disk
2. Various backtest and performance statistics are calculated and written to disk

Trade Close Price Calculation

To check if a trade exceeded its stop-loss or take-profit limits on any given day, the hypothetical total return the trade would have yielded for the day's high and low prices are calculated.

$$r_{tot} = \left(\frac{\text{price} + \text{dividends}_{\text{per share}}}{\text{fillPrice}} - 1 \right) * \text{direction}$$

Where *direction* is 1 for long trades and -1 for short trades.

If r_{tot} (for the day's high and low prices) exceeds the take-profit or stop-loss limits the close price of the trade is calculated respectively as:

$$\text{closePrice}_x = \text{fillPrice}_x * (1 + \text{takeProfit}_x)$$

$$\text{closePrice}_x = \text{fillPrice}_x * (1 + \text{stopLoss}_x)$$

If the timeout limit is reached the close price of the trade is set to the close price of the stock on the date the trade is timed out.

The trade exit policy outlined above is illustrated in figure 9. The return series of daily high, low, and close prices are graphed. The green dashed line represents the take-profit limit (expressed as a return) at 8.9%, the red dashed line represents the stop-loss limit at approximately 6.5%, and the blue line represents the timeout limit at one month after the trade was initiated. The daily high return series peaked at above 10%, but the close price of the trade is set such that the total return is equal to the take-profit limit at 8.9%.

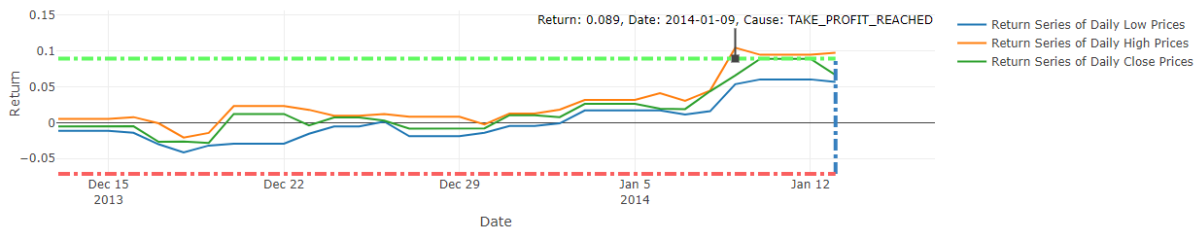


Figure 9: Trade Exit Policy Visualized

Transaction Cost Calculation

Commissions are calculated by the following formula:

$$commission = \min(\max(commission_{min}, commission_{per\ share} * |orderAmount|), \text{percentOfValue}_{max} * |orderAmount| * price)$$

Where $commission_{min}$ is the minimum commission per trade (\$0.35), $commission_{per\ share}$ is the commission charged by the broker per share being bought or sold (\$0.0035), $\text{percentOfValue}_{max}$ is the maximum percentage of total trade value that will be charged (1%) and $price$ is the fill or close price of the trade. This formula is applied to calculate commission for both entering and exiting a trade.

In addition to the commission, the following transaction costs are charged when entering or exiting a trade:

$$US\ clearing\ fees = 0.00020 * |order_{amount}|$$

$$US\ transaction\ Fees = 0.0000207 * |order_{amount}| * price_{fill}$$

$$NYSE\ pass\ through\ fees = 0.000175 * commission$$

$$FINRA\ pass\ through\ fees = 0.00056 * commission$$

$$FINRA\ trading\ activity\ fees = 0.000119 * |order_{amount}|$$

Total transaction fees charged when entering and exiting long and short trades are:

$$transactionFees = commission + usClearingFees + usTransactionFees + nysePassThroughFees + finraPassThroughFees + finraTradingActivityFees$$

This commission model is adopted from Interactive Brokers for North American stocks (tiered pricing structure) as of April 2019.

Margin Requirement Calculation

The margin requirement is calculated for each trade individually and then added together to produce the total required size of the margin account. Margin requirements are only calculated for short trades, following the following scheme:

If trade x is in the money:

$$tradeMarginRequirement_x = shortPosValue_x * (1 + initialMarginRequirement)$$

If trade x is out of the money:

$$tradeMarginRequirement_x = shortPosValue_x * (1 + maintenanceMarginRequirement)$$

Where:

$$\text{shortPosValue}_x = \text{curPrice}_x * |\text{tradeAmount}_x|$$

A trade (x) is in the money if $\text{curPrice}_x \leq \text{fillPrice}_x$

price_{cur} is the current price of x (always the close price), tradeAmount_x is the number of stocks the trade involves.

Performance Measurement and Inspection

The performance of a backtested strategy is measured most notably by its return (total and annualized) and the Sharpe ratio (traditional as well as adjusted for skew and kurtosis). Several other statistics are also calculated, see section 4.2.2 for an exhaustive list.

The results of backtests can be expected in a web-based dashboard. The dashboard includes

- A list of backtest statistics (hit ratio, the standard deviation of monthly returns, mean return from hits and many more)
- Graphs of portfolio value, total return, monthly returns (line chart and histogram) and fund allocation between different tickers for all days of the backtest.
- A tool to inspect details of every trade conducted, including the signal and order behind it.
- Complete sortable lists of all signals, orders, and trades produced over the backtested period.

3.4.1 Statistical Significance of ATS Outperformance relative to the S&P500 Index

To test whether the ATS was able to generate excess monthly returns relative to the S&P500 index, the following test procedure was followed.

Monthly returns were calculated for both the ATS and the S&P500 index. Samples to base the statistical testing on was constructed by taking the difference between the series of ATS returns and S&P500 index returns. The return differences D_1, D_2, \dots, D_n are random samples from a what is assumed to be a normal distribution with unknown μ and σ^2 . If this is the case, the random variable $\sqrt{n} \frac{(\bar{D} - \mu)}{s}$ has a student t-distribution with $n-1$ degrees of freedom. Where S is the sample standard deviation.

A one-sided single sample t-test is conducted with the null hypothesis of monthly excess returns having a mean of zero. The alternative hypothesis is that the monthly excess returns have a mean above zero.

$$H_0: \mu = 0$$

$$H_1: \mu > 0$$

The hypothesis is tested a significance level (α) of 0.05. The t-statistic is defined as follows:

$$t = \frac{\bar{d} - 0.5}{s/\sqrt{n}}$$

A T-value above the critical value will cause rejection of the null hypothesis in favor of the alternative hypothesis.

4 Results and Discussion

4.1 An Empirical Study of US Equity Returns

This section will go over the results from running regressions on monthly equity risk premium for individual stocks using multiple linear regression, principal component regression, random forests, and artificial neural networks. First in-sample results are presented (section 4.1.1) and, in the cases, where cross-validation was used, the results on the validation sets are also presented. This latter gives some indication of the extent the models overfit to the training data and fail to generalize. Secondly, out-of-sample results are presented for all ML algorithms (section 4.1.2).

To give some context for the results obtained by the ML models, results are also calculated for a historical mean forecast and a zero-value forecast. These forecast modes are defined as follows:

$$\bar{r}_{T+n} = \frac{1}{T} \sum_{t=1}^T r_t = \frac{1}{N \cdot T} \sum_{i=1}^{N'} \sum_{t=1}^{T'} r_{i,t} = 0.016327,$$

$$\bar{r}_{T+n} = 0$$

Where n represents the number of months since the beginning of the testing period. N' and T' are the last stock and month included in the training set. The historical mean forecast is not updated as time progresses. All monthly risk premiums are weighted equally across stocks and time.

Table 2: Out-of-sample regression results for the historical and zero mean forecast models:

Model	MSE	MEA	R_{00s}^2	Zero benchmarked R_{00s}^2
Historical Mean Forecast	6.0130	0.1053	-0.0000063%	0.00410%
Zero Mean Forecast	6.0133	0.1041	-0.004.11%	0.0

The historic mean forecast is barely outperforming a forecast value of zero in terms of zero benchmarked R_{00s}^2 . Naturally, the zero mean forecast ends up with a zero benchmarked R_{00s}^2 of zero.

4.1.1 In-Sample Results

4.1.1.1 Multiple Linear Regression and Principal Component Regression

Condensing the dataset of 86 features down to five principal components resulted in the following variance ratios.

Table 3: Principal Components - Variance Ratios

	PC1	PC2	PC3	PC4	PC5
Variance Ratio	0.2783	0.1929	0.1857	0.1802	0.1623

With PC5 capturing 16.23% of the variance it is likely that subsequent principal components would continue to capture significant portions of the variance in the dependent variable.

Table 4: In-sample results for multiple linear regression and principal component regression

Algorithm	MSE	MEA	R^2	Zero benchmarked R^2
Multiple Linear Regression	65.71067	0.1708	0.0317%	0.0321%
Principal Component Regression	65.7313	0.1286	0.00024%	0.000643%

In-sample, the multiple linear regression model significantly out-performs the principal component regression model. This is not surprising as the multiple linear regression model has many more degrees of freedom, increasing the chance of overfitting. Neither value is negative, indicating that both models dominate the naïve forecast of zero. At the same time, both zero benchmarked R^2 values are very low, leading to the conclusion that both models are unsuccessful in modeling in-sample monthly equity risk premiums well.

4.1.1.2 Random Forest Regression

Following are the chosen hyper-parameters from running purged k-fold cross-validation on the training set. Also, the performance metrics on training and validation folds for the chosen hyperparameters are presented.

Table 5: Random Forest Regression Model - Best Hyperparameters

Parameter	Value
Max Features	10
Min Weight Fraction Leaf	0.05
Number of Estimators	200

Table 6: Random Forest Regression Model - Cross-validation results given best hyperparameters

Statistic	Value
Training Folds	
Mean train MSE	0.05841049
Mean train MAE	0.10052616
Mean train R^2	6.2705%
Testing Folds	
Mean test MSE	65.73307004
Mean test MAE	0.12736023
Mean test R^2_{00s}	-0.4736%

The significantly higher mean MSE on the validation folds compared to the testing folds indicate substantial overfitting. Still, the hyperparameters given in table 4 was the best configuration measured by mean MSE on test/validation folds. The zero benchmarked R^2 was not calculated during model training and is therefore not given here. A mean R^2 of 6.27% on the training folds indicate significant explanatory power of the model in-sample. This, however, did not translate to out-of-sample data, as the model is dominated by a mean forecast on the test/validation folds.

4.1.1.3 Deep Neural Net Regression

Following is the chosen hyperparameters from running purged k-fold cross-validation using a deep neural net. Also, the performance metrics on training and validation folds for the chosen hyperparameters are presented.

Table 7: Deep Neural Net Regression Model - Best Hyper Parameters

Parameter	Value
Initial learning rate	0.01
Lambda parameter of L2-regularization	0.3
Epochs	500
Patience	10
Dropout	0.2

Table 8: Deep Neural Net Regression Model - Cross-validation results given best hyperparameters

Statistic	Value
Training Fold	
Mean train MSE	0.082186
Mean train MAE	0.11602275
Mean train R^2_{OOS}	0.2178%
Testing Folds	
Mean test MSE	196.9869
Mean test MAE	0.1489366
Mean test R^2_{OOS}	0.001884%

The mean MSE on the validation folds is much higher than the mean MSE on training fold, indicating significant overfit despite the many efforts to regularize the model. Applying deep neural nets to such a high variance and noisy dataset proved to be a challenging task. It is likely with more time and experimentation the results would improve. It is worth noting that despite allowing 500 epochs, due to early stopping, training would often stop after only 10-20 epochs. Artificial neural nets should have more potential for modeling equity risk premium than what these results indicate. It is more appropriate to conclude that the training procedure used was suboptimal rather than ANNs being ill-suited for equity risk premium prediction.

4.1.2 Out-of-Sample Regression Model Results

The following are the out-of-sample results of the multiple linear regression, principal component regression, random forest, and deep neural net models on equity risk premium (ERP) prediction.

Table 9: Out-of-Sample ERP Regression Results

Algorithm	MSE	MEA	R_{OoS}^2	Zero benchmarked R_{OoS}^2
Historical Mean Forecast	6.0130	0.1053	-0.0000063%	0.00410%
Zero Mean Forecast	6.0133	0.1041	-0.004.11%	0
Multiple Linear Regression	6.4629	0.1638	-7.48%	-7.47800%
Principal Component Regression	6.0130	0.1054	0.0008581%	0.00496%
Random Forest	6.0110	0.1075	0.03848%	0.04259%
Deep Neural Net (1 hidden layer)	6.0122	0.1123	0.01374%	0.01785%

At the highest level, out-of-sample results are rather unimpressive. No model was able to outperform a zero-forecast model clearly. The random forest regressor is most successful out-of-sample with a zero benchmarked R_{OoS}^2 of 0.043%. The least successful model is the multiple linear regression with a zero benchmarked R_{OoS}^2 of -7.48%, entirely dominated by a zero-forecast model. Despite poor performance across the board, it is worth noting that both the random forest and deep neural net models yielded a zero benchmarked R_{OoS}^2 an order of magnitude higher than principal component regression (and much better than the negative value of the multiple linear regression model). This is some indication that more advanced ML algorithms, which allow complex interactions among predictors, are better at modeling equity risk premiums. However, it is challenging to interpret regression results without visual aids, which are out of reach when dealing with 87 dimensions. The low R_{OoS}^2 values obtained further limits the adoption of definite conclusions.

The random forest model's outperformance may be attributed to its application of ensemble methods to increase accuracy and ability to train diverse individual predictors to reduce forecast variance.

Many things could be done to likely improve the results. Most notably:

- Use more data, spanning a longer timeline and broader set of stocks.
- Try to exclude irrelevant samples, for example, those labeled with meager returns.
- Add more weights to more informative samples during training. Information content can, for example, be measured by unexpected size and variance of returns.
- Retrain the models every month or year to use as much training data as possible to inform each prediction (see section 4.1.4).
- Do more extensive feature selection and feature engineering to improve the dataset.
- Do wider hyper-parameter searches.
- Create more specialized models, one to predict negative returns and one to predict positive returns, for example.

Using reason and theory alone, it should be easy to convince anyone of the utility of machine learning to equity risk premium prediction and other problems in finance. Despite this project being

unable to show good performance on out-of-sample equity risk premiums, it is not evidence of ML algorithm’s inability to model equity risk premiums well.

4.1.3 Random Forest ERP Classification and Statistical Significance of Model Accuracy

In addition to training regression models, a classifier was trained to predict the sign of equity risk premiums. Following are the purged k-fold cross-validation results.

Table 10: Random Forest ERP Classifier - Best Hyperparameters

Parameter	Value
Max Features	3
Min Weight Fraction Leaf	0.05
Number of Estimators	200

Hyperparameter tuning via cross-validation showed a preference for smaller forests with trees trained on a small number of features. The reduced number of features considered by each tree should yield less correlated predictors. Despite the chosen “min weight fraction leaf” parameter being the lowest in the hyperparameter space being search, a value of 5% still introduces significant regularization.

Table 11: Random Forest ERP Classifier - Cross-validation results given best hyperparameters

Statistic	Value
Training Folds	
Mean train Accuracy	0.67009763
Mean train Balanced Accuracy	0.6714638
Mean train F_1	0.67707887
Mean train precision	0.64859814
Mean train recall	0.70913802
Test Folds	
Mean test Accuracy	0.50707899
Mean test Balanced Accuracy	0.51702517
Mean test F_1	0.5116706
Mean test precision	0.49448325
Mean test recall	0.5562132

Significantly higher F_1 score on training folds indicates overfitting despite efforts to make the model generalize. The failure to generalize could be attributed to an insufficient amount of data, too many irrelevant samples, insufficient regularization, and too few predictive features.

Table 12: Out-of-Sample ERP Sign Classification Results

Algorithm	Precision	Recall	F_1	Accuracy
RF	0.5337	0.5965	0.5633	0.5333

Recall of 59.65% on out-of-sample data means the model was correctly identifying 59.65% of actual positives (EPR with a positive sign). Precision of 53.37% means that the model predicts correctly 53.37% of the time when predicting a positive class. An accuracy of 53,33% and F_1 score of 56.33% looks like a significant result where the algorithm has been able to learn relevant patterns form the training data. To verify, t-tests were conducted on model accuracy.

4.1.5.1 T-test of RF ERP Classifier Accuracy:

To test whether the classifiers accuracy was superior to a random model with an accuracy of 50%, a single sample one-sided t-test was conducted (see section 3.2.8 for a description). Because the t-test gave such conclusive results with $H_0: \mu = 0.50$, t-test was conducted to compare the ML model's accuracy with accuracy levels from 51-53% as well.

Note that the 200 samples of 3000 observations, drawn with replacement, had a mean accuracy of 53.37% and standard deviation of 0.88%.

Table 13: T-tests of Random Forest ERP Sign Classifier Accuracy

T-Test Nr.	H_0	Alpha	Critical Value	T-statistic	P-value	Conclusion
1	$\mu = 0.50$	0.05	1.6525	54.1524	0.0	Reject H0
2	$\mu = 0.51$	0.05	1.6525	22.0969	0.0	Reject H0
3	$\mu = 0.52$	0.05	1.6525	6.0692	3.20e-09	Reject H0
4	$\mu = 0.53$	0.05	1.6525	-9.9586	1.0	Failed to reject H0

The null hypothesis of a mean accuracy of 50% is rejected at even the most conservative significance levels. Even the null hypothesis of mean accuracy of 52% is rejected in favor of the alternative hypothesis that the mean accuracy is greater than 52%. However, the data contains insufficient evidence to reject the null hypothesis of a mean accuracy equal to 53%.

4.1.4 Would re-fitting the model over the testing period yield better results?

To assess whether re-fitting the model as more data becomes available would yield better out-of-sample results, two models were developed and compared. One model was trained on the training set, spanning the period from 1998-01-01 to 2012-01-01. A second model was trained on an extended training set, spanning the period from 1998-01-01 to 2016-01-01. Both models were tested on out-of-sample data spanning the period from 2016-03-01 to 2019-02-01. The out-of-sample results were as follows:

Table 14: Out-of-Sample Results of RF Trained on Extended Training Set

Algorithm	Precision	Recall	F_1	Accuracy
RF trained on training set	0.5178	0.5786	0.5465	0.5272
RF trained on extended training set	0.5198	0.6367	0.5723	0.5315

Training on the extended training set gave significant performance improvements. It is likely the project would have obtained better performance across the board if ML models were retrained, e.g., every year.

4.1.4 Note on Feature Importance – RF ERP Regressor and Classifier

Feature importance is commonly measured as the reduction in the cost function resulting from the addition of a feature. Due to time and resource limitations, this project did not include measuring feature importance in this way. To still provide some insight to what extent different features contributed to model performance, relative feature importances of the random forest ERP regression and classification models were gathered.

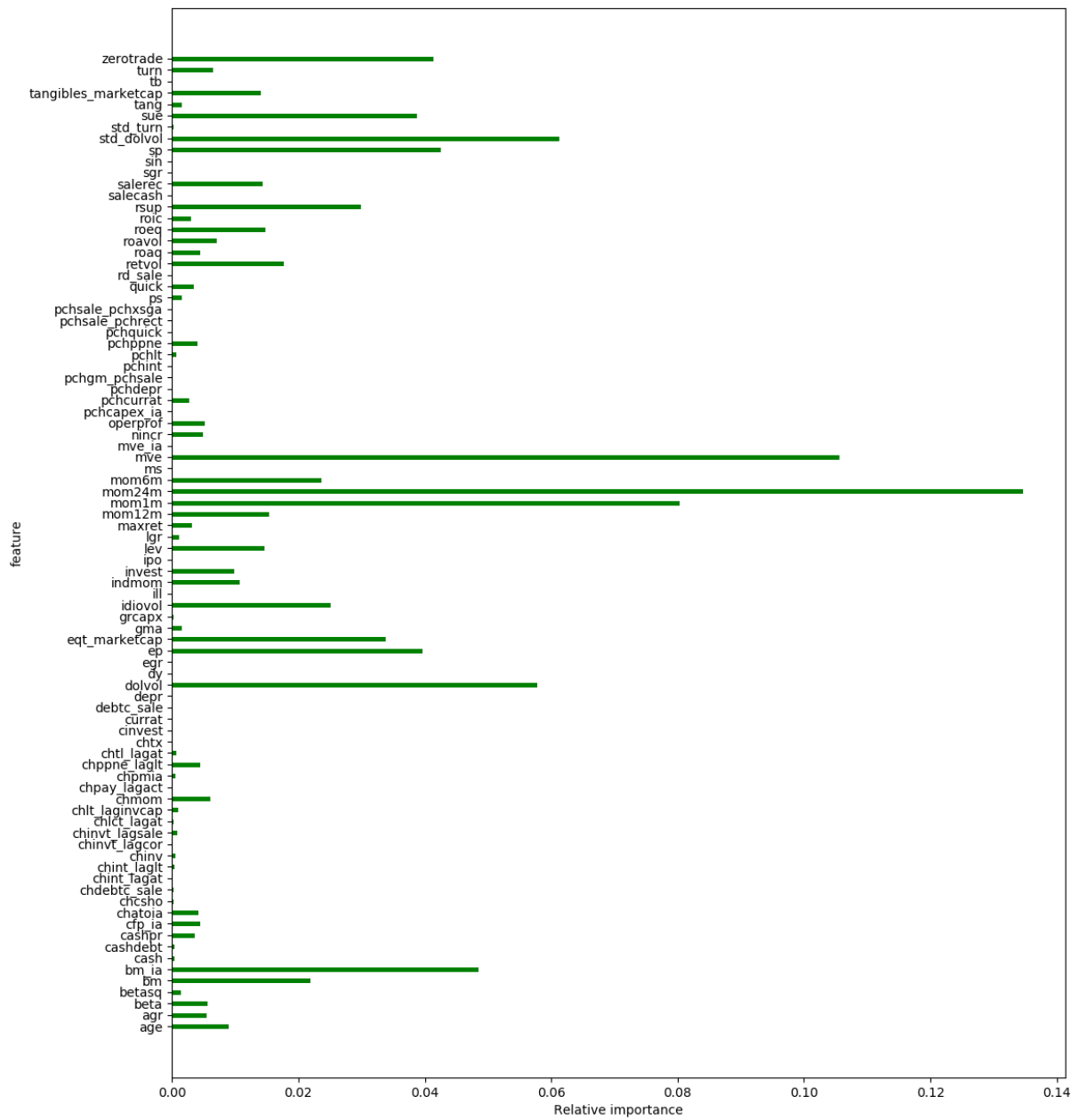


Figure 10: Relative Feature Importance - Random Forest ERP Regression Model

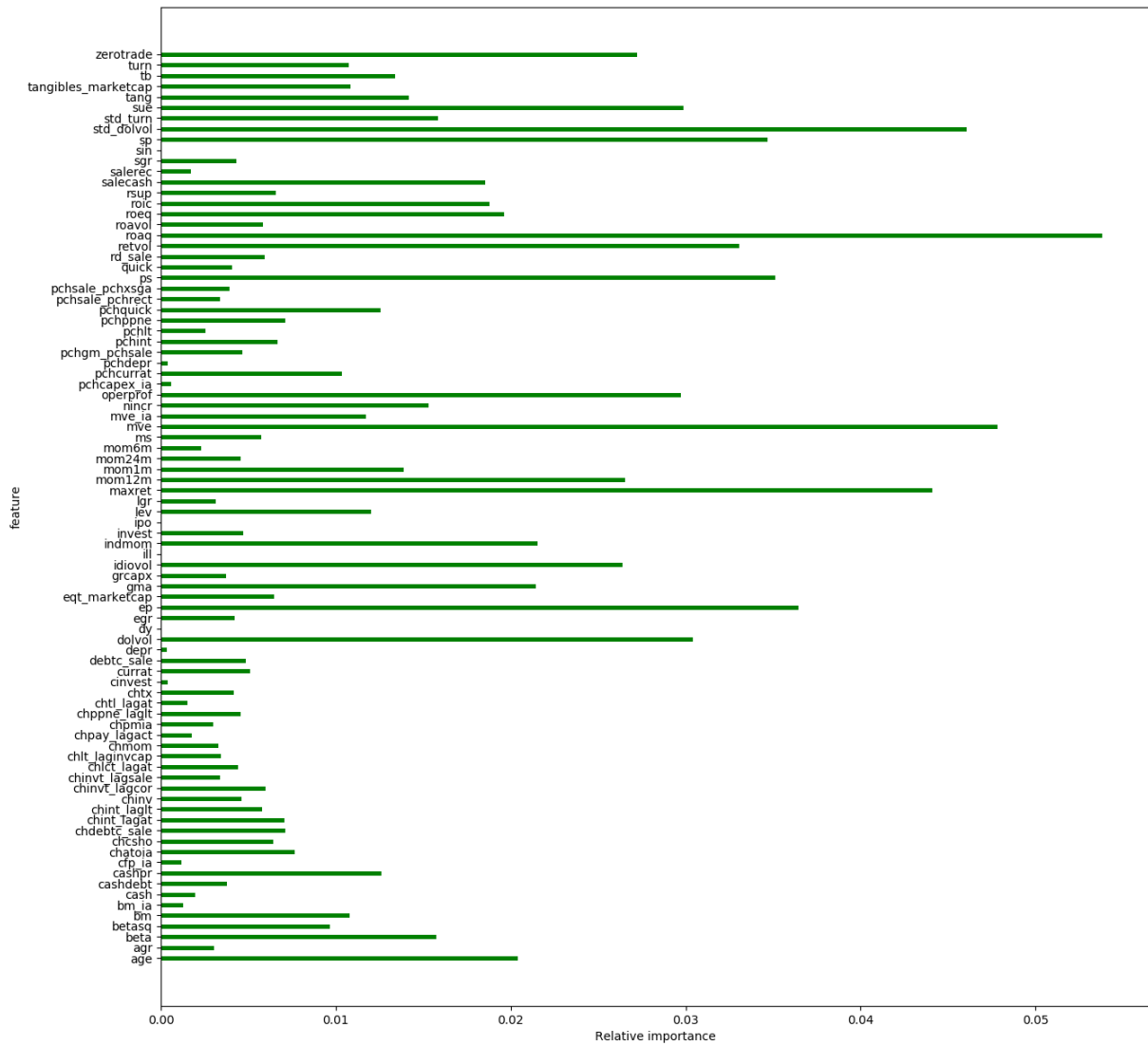


Figure 11: Relative Feature Importance - Random Forest ERP Sign Classification Model

Both models seem to find predictive power in both momentum and trading friction related features (mom1m, zerotrade, std_dolvol). Another common favorite is the natural log of market capitalization (mve). The RF Classifier relies on a broader set of predictors than the RF regressor. The RF classifier relies heavily, in addition to the aforementioned features, on return on assets (roaq), earnings surprises (sue), sales to price (sp), financial statement score (ps), maximum daily return (maxret) and earnings to price (ep). Some features are weighted much more in either the regressor or classifier. For example, 24-month momentum (mom24m) is heavily used by the regressor but not considered relevant by the classifier. Why this is the case is unknown, but some of it can probably be explained away by randomness.

4.2 Automated Trading System Performance

The following will go over the results of backtesting both an ML driven trading strategy and a strategy randomly picking stocks. First, results from side and certainty model training, and out-of-sample testing are presented.

4.2.1 Side and Certainty Classifier Results

Purged k-fold cross validation for the primary (side) classifier resulted in the selection of the following hyperparameters:

Table 15: Side Classifier - Best Hyperparameters

Parameter	Value
Max Features	5
Min Weight Fraction Leaf	20%
Number of Estimators	200

Table 16: Side Classifier - Cross-validation results given best hyperparameters

Statistic	Value
Training Folds	
Mean train Accuracy	0.5980
Mean train Balanced Accuracy	0.5983
Mean train F_1	0.6070
Mean train precision	0.6147
Mean train recall	0.6007
Testing Folds	
Mean test Accuracy	0.5195
Mean test Balanced Accuracy	0.5202
Mean test F_1	0.5117
Mean test precision	0.5353
Mean test recall	0.4926

The algorithm shows some tendency to overfit despite highly regularized trees. However, the degree of overfitting is less than that of the RF ERP sign classification model. The increased minimum weight fraction leaf parameter might explain this result.

Note that the certainty classifier was trained on the parameters found to be optimal for the side classifier.

The following are the out-of-sample results for both the side and certainty classifiers.

Table 17: Out-of-Sample Side and Certainty Classifier Results

Algorithm	Precision	Recall	F_1	Accuracy
Side Classifier	54.78%	62.81%	58.52%	54%04
Certainty Classifier	54.96%	61.21%	57.91%	53.09%

The side and certainty classifiers have certainly been able to learn some patterns from the training data that allow them to outperform pure chance significantly.

4.2.2 Backtest Results of the Machine Learning Driven ATS

To give insight into how the ATS performs under different configurations, backtesting was done with the ATS configured three different ways. These different configurations are, in essence, just tuning the number of concurrent trades (diversification) and strictness of stop-loss limits.

Selection bias and multiple testing are important issues in backtesting. This project largely avoided the problem by conducting model training and system tuning on in-sample data and avoid running backtests on out-of-sample data before the system was completed. The first backtest, not affected by multiple testing, is the one corresponding to configuration A.

Table 18: Automated Trading System Configurations

Configuration	A	B	C
Starting Capital	\$1,000,000	\$1,000,000	\$1,000,000
Max positions	30	20	10
Max position size	5%	8%	15%
Minimum balance	\$5,000	\$5000	\$5000
Max percent to invest each period	33%	33%	100%
Max orders per period	10	7	10
Min order size	\$10,000	\$10,000	\$10,000
Frequency of trading	Daily	Daily	Daily
Stop-loss limit	80% of -EWSD	50% of -EWSD	50% of -EWSD
Take-profit limit	100% of EWSD	100% of EWSD	100% of EWSD
Accepted age of signals	1 Week	5 Days	4 Days

Table 19: Backtest Results for ML driven Automated Trading System

Statistic	Configuration A	Configuration B	Configuration C
Start Value	\$1,000,000	\$1,000,000	\$1,000,000
End Value	\$2,182,537	2,156,126	\$2,174,992
Total Return	118.26%	115.61%	105.97%
Annualized Return	12.09%	11.87%	10.97%
Monthly Return Standard Deviation	0.03026	0.02961	0.03240
Correlation to Underlying	0.8473	0.8211	0.7137
S&P500 Total return	96.97%	96.97%	96.97%
S&P500 Annualized Return	10.42%	10.42%	10.42%
S&P500 Monthly Return Standard Deviation	0.03190	0.03190	0.03190
Statistical Significance of outperformance ($\alpha = 0.05$, Crit. val. = 1.6639)	Filed to reject H0 T-statistic=0.2961 P-value=0.3840	Failed to reject H0 T-statistic =0.3295 p-value=0.3712	Filed to reject H0 T-statistic=-0.0839 P-value=0.5333
Normality of returns (Shapiro-Wilk test, $\alpha = 0.05$)	Returns are non-normal W=0.948	Returns are non-normal W=0.958	Returns are non-normal W=0.957
Skew of Returns	-0.8484	-0.7886	-0.4946
Kurtosis of Returns	0.6274	0.5880	0.0566
Sharpe Ratio (Monthly Returns)	0.2583	0.2677	0.2169
Adjusted Sharpe Ratio	0.2484	0.2578	0.2120

Ratio of Longs	1.0	1.0	1.0
Hit Raito	55.48%	44.44%	44.15%
PnL	\$1,182,573	\$1,156,126	\$1,059,731
PnL from short positions	\$0	\$0	\$0
PnL form long positions	\$1,241,367	\$1,216,701	1,131,510 \$
Average return from hits	4.70%	4.97%	4.93%
Average return from misses	-4.23%	-2.96%	-2.95%
Highest return from hit	33.36%	18.02%	13.16%
Lowest return from miss	-15.95%	-9.97%	-9.09%
Average holding period	19.35 Days	14.78 Days	14.67 Days
Frequency of bets (Trades/Month)	43.65	37.30	18.83
Maximum dollar position size	\$111,418	\$185,228	\$343,244
Average AUM	\$1,546,244	\$1,497,810	\$1,537,313
Broker fees per dollar	\$0.0002111	\$0.0002101	\$0.0002112
Broker fees per stock (\$/amount)	\$0.0095149	\$0.0095238	\$0.0095103
Trades closed by Take-Profit	1261	950	489
Trades closed by Stop-Loss	1220	1572	805
Trades closed by timeout	1142	572	269
Trades closed by bankruptcy	0	0	0
Trades closed by delisting	0	0	0
Total Commission	\$42,905	\$54,211	\$55,935
Total Charged to enter trades	\$203,464,514	\$ 258,120,063	\$264,796,785
Total Margin Interest	0	0	0
Total Interest Received	\$28,949	\$37,877	\$37,287
Total Dividends	\$156,812	\$144,584	\$159,388
Total Proceeds	\$204,218,546	\$259,048,248	\$265,715,778
Unique Stocks Invested In	700	595	461

The results show that the ML-driven ATS outperform the S&P500 index over the testing period. Over the 7-year testing period, the ML-driven ATS was able to deliver 118.26%, 115.61%, and 105.97% return (under configuration A, B, and C, respectively) compared to the 96.97% of the S&P500 index. In annualized terms, configuration A was able to deliver returns of 12.09% compared to 10.42% of the index.

Overall, the ML-driven ATS exhibit a strong correlation to the S&P500 index. Figure 13 shows the ML strategy's (configuration A) and S&P500's monthly returns over time. There is clearly a high correlation between the two series. For configuration A, B, and C, the correlation between the ML-driven ATS's returns and the S&P500's returns were 0.85, 0.82, and 0.71, respectively. As the number of concurrent trades is reduced (reduced diversification), the correlations to the market were unsurprisingly also reduced.

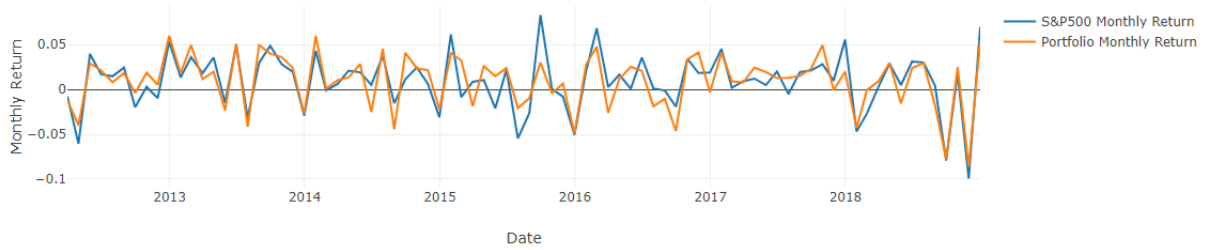


Figure 12: Monthly Returns of ML Strategy and S&P500 Index (Configuration A)

As explained in section 3.3, the signal and order generation algorithms rank trading signals based on the probability the secondary/certainty classifier assigns to class 1 (primary model is correct). The secondary classifier generally assigns a higher probability to class 1 when the primary model predicts the price will increase (sets side to long). This leads to a preference for the system to take long trades. This slight preference becomes very significant when the system is presented with over 1500 trading opportunities every day and has to select only the top 5 or 10. The system ends up never going short at all (ratio of longs: 1.0). During the system’s development, it was experimented with forcing some proportion of the capital to be reserved for short trades, but this was ultimately decided against in favor of blindly following the recommendations of the ML algorithms.

The hit ratio of the system was 55.48% for configuration A. When configuring the stop-loss limits more stringently (configuration B and C set stop-loss to 50% of $-EWSD_{t-24,t,i}$) it resulted in lower hit ratios (44.44% and 44.15%), but this also leads to higher average return from misses (from -4.23% to -2.96%, moving from configuration A to B). There is some unknown optimal compromise between limiting down-side and maximizing hit-ratio.

Each trade is held at most one month. Configuration A had an average holding period of 19.35 days. To maintain maximum degree of investment in the market an average frequency of bests of 43.65 trades per months was maintained by the ATS under configuration A. More stringent stop-loss limits resulted in shorter holding periods and thus higher frequency of bets.

For configuration A, the maximum dollar position size taken in a single trade was \$111,418. Which must have been towards the end of the backtest, because trade sizes were limited to 5% of total portfolio value. The size limitation and the frequency of bests should prevent the strategy’s results from being a result of a few spectacularly profitable trades.

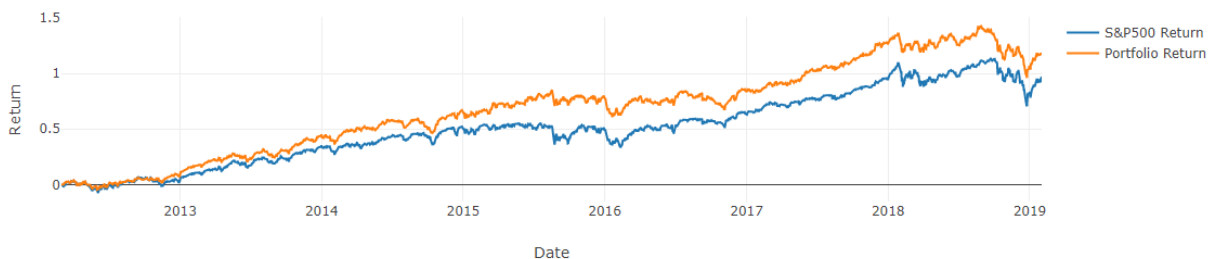


Figure 13: Total Return Relative to Initial Value of ML-driven ATS (Portfolio) and S&P500 Index (Configuration A)

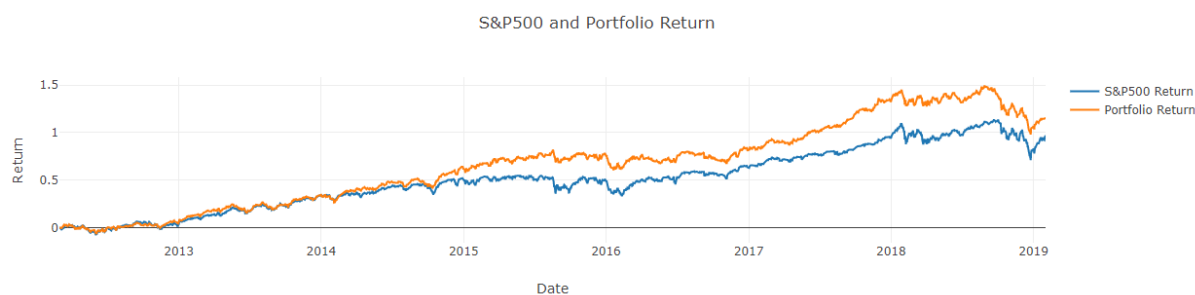


Figure 14: Total Return Relative to Initial Value of ML-driven ATS (Portfolio) and S&P500 Index (Configuration B)

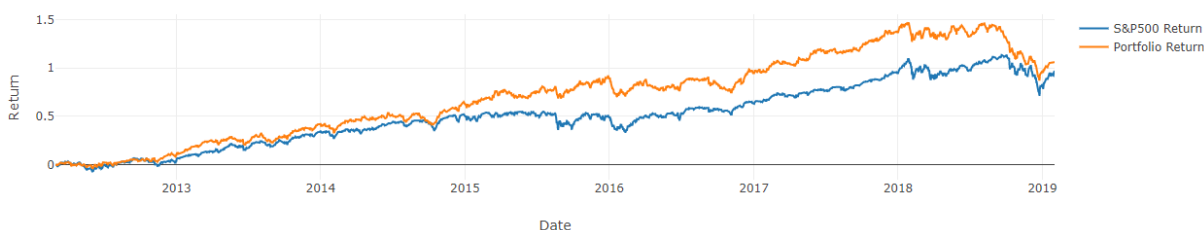


Figure 15: Total Return Relative to Initial Value of ML-driven ATS (Portfolio) and S&P500 Index (Configuration C)

Figures 13, 14 and 15 show total return over the test period relative to portfolio/S&P500 initial value for configurations A, B, and C. In all cases the portfolio's total returns are superior to the S&P500's. At no point over the backtested period, did the ML strategy clearly underperform the index. This indicates robustness. However, it is important to note that the U.S. economy experienced stable growth over the testing period and due to the system not being capable of detecting or adjust to different regimes, it is unlikely similar performance would be achieved under less favorable conditions. Backtests are at best hypothetical, and the measured performance is not proof, or even strong indication, that the system will perform similarly outside the backtested period.

The ML strategy had a high turnover, a total of \$203,464,514 was charged to enter trades over the 7-year testing period for configuration A. Which means that the original portfolio value of \$1,000,000 entered and exited the market approximately 28 times per year on average.

When it comes to commissions, the portfolio was charged \$42,905 in commissions and fees over the 7-year testing period (given configuration A). The system was charged on average \$0.0095149 per stock bought and sold. This fits nicely with the \$0.0035 per stock commission (fees come in addition), which the system was configured to charge. Only \$42,905 in commissions and fees, given that more than \$200 million moved in and out of the market, may seem on the low side. Due to the relatively high turnover of the system, the backtest's results are sensitive to commission rates.

The system was limited to buy or sell only 10% of the previous day's volume in any stock. Under this restriction, the slippage was assumed to be zero on all trades. This assumption may give the system a slight and unrealistic advantage, but the effect should be minimal given the amount of capital under management. Modeling slippage accurately is more important for systems managing larger pools of funds. Further, under real market conditions, the broker is not able to fill trades the very moment traders submit them. Buyers and seller must be matched, and the orders must be processed. This introduces some lag and prices can move during this time. This random jitter is assumed to have a normal distribution with a mean of zero and should therefore not produce an advantage or disadvantage for the system under testing. Modeling these "random" price movements were therefore omitted.

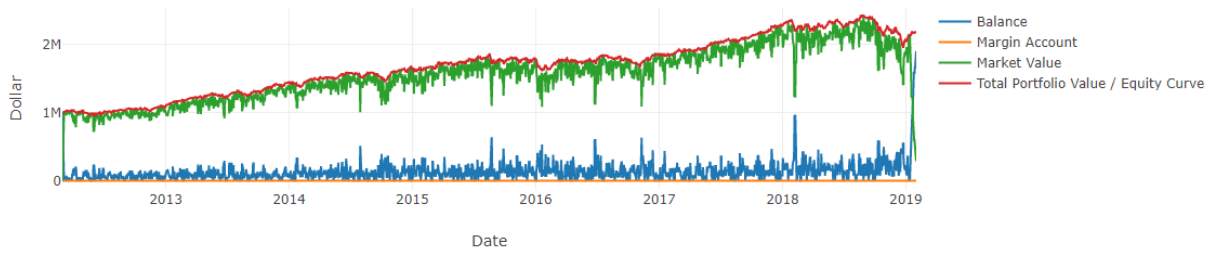


Figure 16: Allocation of value between different accounts – ML-driven ATS under configuration A (end of day values)

Figure 16 shows how the total portfolio value (equity curve) increases over the test period. The blue line shows the cash balance while the green line shows the market value of active long and short trades. The red line represents the total value of the portfolio or the combined values of all accounts. The system continually strives to minimize the cash balance in favor of having the money investing in stocks. The less money is placed in stocks, the less money is generating returns. The system aims to keep as much as possible invested to take full advantage of the predictive power of the ML models. Due to positions being exited randomly at any point during the week (as exit limits are reached), it was decided to enable trading every day. If the system were only able to trade once a week, the average size of the cash balance would be significantly higher.

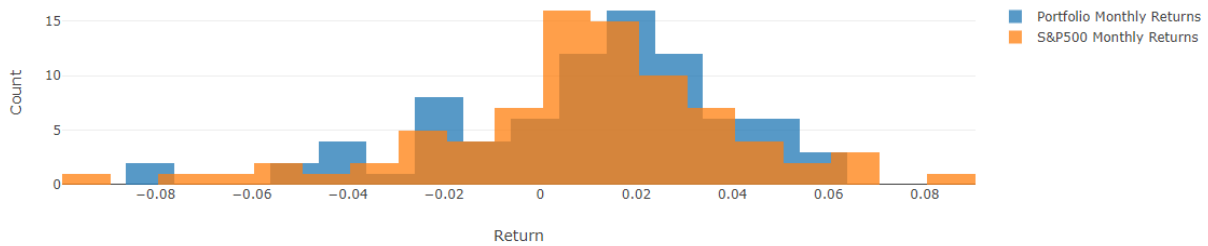


Figure 17: Histogram of Monthly Returns for ML-driven ATS and S&P500 Index (Configuration A)

Figure 17 shows that the ML strategy (configuration A) seems to have obtained monthly returns skewed more to the right than the S&P500. The mean monthly return is approximately 2%.

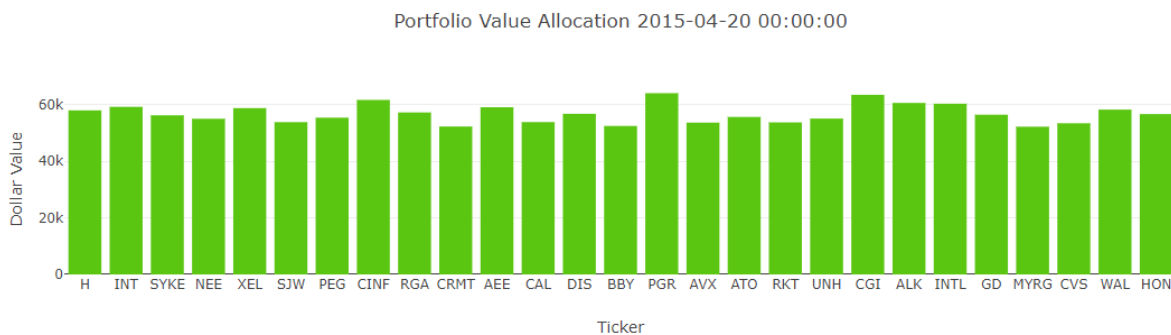


Figure 18: Example of Fund Allocation Between Concurrent Trades – ML-driven ATS (Configuration A)

Figure 14 shows that the fund allocation scheme causes the distribution of funds between trades to be approximately uniform. This is generally the case across time and different ML strategy configurations. This is due to the certainty level (probability of class 1) assigned by the certainty classifier for the top signals are very similar (around 0.53-0.55). This show that the ML models are unable to clearly differentiate between promising signals or have high confidence in any few signals/observations. This point to a problem with how the modeling problem was set up. Modeling was done with the aim of applying the same models to all stock, across all time, without regard for industry, market conditions, or other characteristics. Fitting a model on such a broad problem naturally leads to a model forced to generalize and lacking ability to take advantage of idiosyncrasies related to a smaller part of the U.S. equity universe.

4.2.3 Backtest Results of a Randomly Trading ATS

In addition to running backtests on the ATS powered by ML models, backtesting of an ATS which traded randomly was also performed. This randomly trading ATS was subject to the same restrictions and configured the same way as the ML-driven ATS, making results more comparable.

Table 20: Backtest Results for Random Automated Trading System

Statistic	Configuration A	Configuration B	Configuration C
Start Value	\$1,000,000	\$1,000,000	\$1,000,000
End Value	\$1,614,280	\$1,181,000	\$1,263,113
Total Return	61.43%	18.10%	26.31%
Annualized Return	7.16%	2.43%	3.43%
Monthly Return Standard Deviation	0.04053	0.04279	0.04580
Correlation to Underlying	0.8233	0.7386	0.72490
S&P500 Total return	96.97%	96.97%	96.97%
S&P500 Annualized Return	10.42%	10.42%	10.42%
S&P500 Monthly Return Standard Deviation	0.03190	0.03190	0.03190
Statistical Significance of outperformance ($\alpha = 0.05$, Crit. Val. = 1.6639)	Filed to reject H0 T-statistic=-1.1635 P-value=0.8760	Filed to reject H0 T-statistic=-2.0216 P-value=0.9767	Filed to reject H0 T-statistic=-1.5344 P-value=0.9356
Normality of returns (Shapiro-Wilk test, $\alpha = 0.05$)	Returns are normal W=0.971	Returns are non-normal W=0.959	Returns are normal W=0.988
Skew of Returns	-0.5099	-0.4155	0.2711
Kurtosis of Returns	0.7349	1.0457	0.0276
Sharpe Ratio (Monthly Returns)	0.1059	0.0190	0.0415
Adjusted Sharpe Ratio	0.10490	0.01898	0.04163
Ratio of Longs	1.0	1.0	1.0
Hit Raito	50.68%	39.86%	40.44%
PnL	\$614,280	\$180,999	\$263,113
PnL from short positions	0	0	0
PnL form long positions	\$747,378	-\$3838	\$363,467
Average return from hits	8.10%	8.71%	8.12%
Average return from misses	-7.10%	-5.36%	-5.39%
Highest return from hit	133.55%	92.04%	36.93%
Lowest return from miss	-60.22%	-42.92%	-28.26%
Average holding period	20.29 Days	15.30 Days	14.73 Days

Frequency of bets (Trades/Month)	40.82	34.92	18.14
Maximum dollar position size	\$95,467	\$111,194	\$241,902
Average AUM	\$1,445,271	\$991,955	\$1,081,495
Broker fees per dollar	\$0.0007113	\$0.0007546	\$0.0006057
Broker fees per stock (\$/amount)	\$0.0070750	\$0.0067056	\$0.0058023
Trades closed by Take-Profit	1004	744	391
Trades closed by Stop-Loss	1106	1531	798
Trades closed by timeout	1278	622	317
Trades closed by bankruptcy	0	0	0
Trades closed by delisting	0	1	0
Total Commission	\$127,342	\$122,811	\$112,174
Total Charged to enter trades	\$179,240,378	\$162,792,963	\$185,209,007
Total Margin Interest	0	0	0
Total Interest Received	\$28,260	\$28,679	\$39,843
Total Dividends	\$141,444	\$75,756	\$84,057
Total Proceeds	\$179,506,197	\$162,925,373	\$185,460,399
Unique Stocks Invested In	2264	2013	1238

Below are some illustrations relating to the performance of the random ATS.

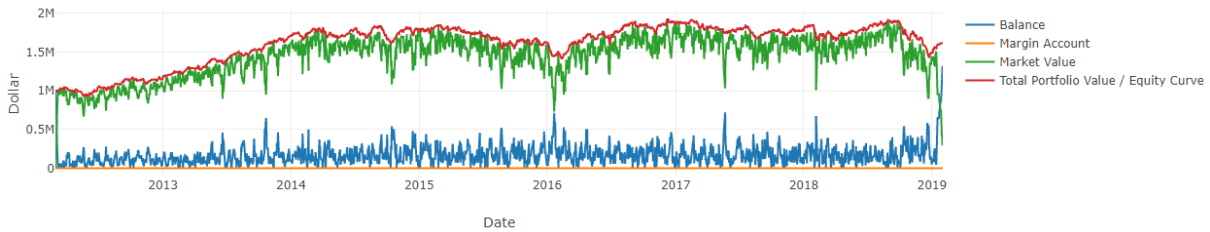


Figure 19: Allocation of value between different accounts - Randomly Trading ATS under configuration A (end of day values)

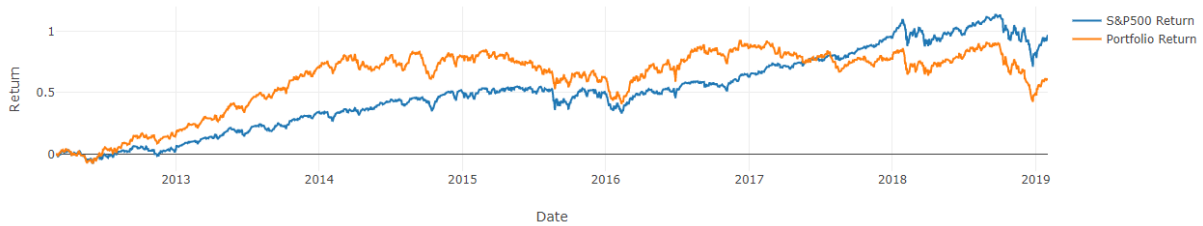


Figure 20: Total Return Relative to Initial Value of Randomly Trading ATS (Portfolio) and S&P500 Index (Configuration A)

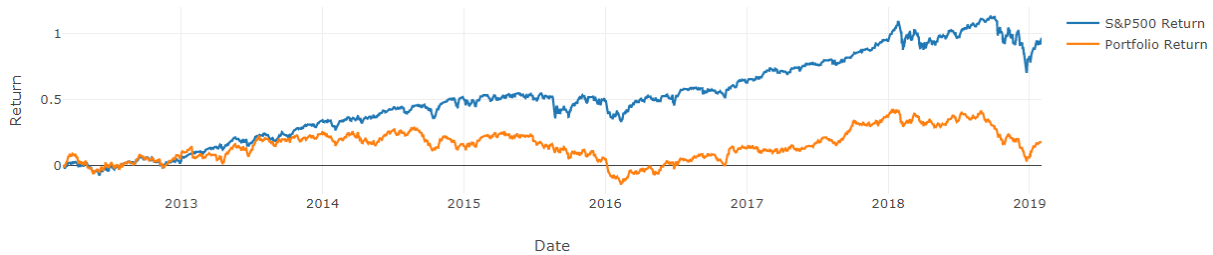


Figure 21: Total Return Relative to Initial Value of Randomly Trading ATS (Portfolio) and S&P500 Index (Configuration B)

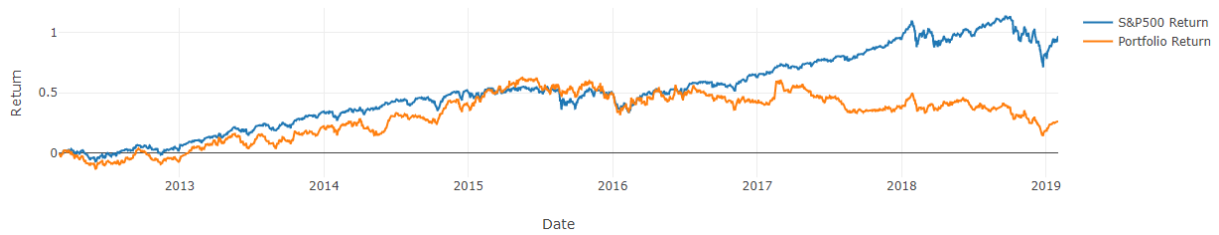


Figure 22: Total Return Relative to Initial Value of Randomly Trading ATS (Portfolio) and S&P500 Index (Configuration C)

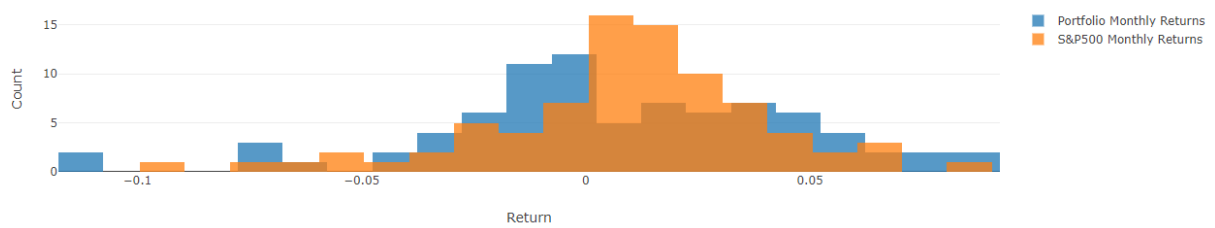


Figure 23: Histogram of Monthly Returns for the Randomly Trading ATS and S&P500 Index (Configuration A)

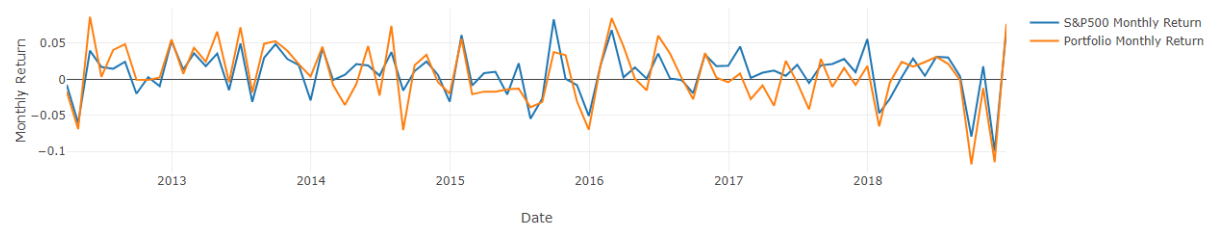


Figure 24: Monthly Returns of Randomly Trading ATS and S&P500 Index (Configuration A)

The returns of the randomly trading ATS is unable to outperform the index by a significant margin under all configurations. Configuration A was most successful, with a total return of 61.43% (7.16% annualized). Monthly returns were negatively skewed for both configuration A and B but exhibited higher kurtosis than the ML-driven ATS. Combining the higher variance in monthly returns with a lower mean, resulted in adjusted Sharpe ratios significantly lower than those of the ML-driven ATS. Adjusted Sharpe ratios were 0.1049, 0.01898, and 0.04163 for configurations A, B, and C, respectively.

The ML-driven ATS traded 700, 595 and 461 unique stocks while the random ATS traded 2264, 2013 and 1238 unique stocks under configurations A, B, and C respectively. This indicates that the ML-driven ATS expresses preferences regarding what stocks it selects.

The consistently poor results over all three backtests indicate that randomly selecting stocks is not a favorable strategy and that the ML-driven ATS is a much better bet. Random stock selection seems unlikely to produce a superior performance to either the index or the ML-driven ATS. Note that the S&P500 index is a highly selected group of large and successful companies. Even keeping up with the index seems to require a strategy with some stock picking ability. The fact that the randomly trading

ATS significantly underperformed the index indicates that the ML-driven ATS, in fact, learned patterns from training data that transferred to the test set and that the system was able to translate this ability to predict correctly into returns.

4.2.4 Note on the Primary Research Question

The aim of the project was primarily to build an automated trading system using machine learning algorithms to inform investment decisions towards the end of producing higher risk-adjusted returns. There is an innumerable amount of ways to integrate machine learning into an ATS. Due to time and resource limitations, the project had to focus on a single approach. It was decided to follow the recommendations of Marco López de Prado in “Advances in Financial Machine Learning” (2018). Most notably, the triple barrier method and meta-labeling were used to label the dataset and train the ML models. The certainty classifier (trained on “meta-labels”) achieved an out-of-sample accuracy of 53.09%. A satisfying result is that the hit-ratio (can be interpreted as accuracy) was as high as 55.48% for the ML-driven ATS under configuration A. With symmetric stop-loss and take-profit limits, this ratio might be even higher. This shows that following only the top predictions of the certainty classifier yields a higher accuracy. A promising path forward is to train the certainty classifier to achieve even higher precision, in order to improve the hit-ratio even further, which should result in higher risk-adjusted returns.

Although the project cannot speak to the merit of TBM and meta-labeling compared to other schemes when training ML models to inform investment decisions, the results certainly indicate that these methods hold great promise. With more data and more specialized models, it is likely that even better results can be obtained using these methods.

More efforts should have been devoted when it comes to designing the system to achieve higher risk-adjusted returns or allowing the investor to tweak the system to better fit their risk-profile. The system is able to mitigate risk through its level of diversification and works to increase returns by following only the top recommendations of the certainty classifier, but it has no ability to control the level of investment in the markets, the ratio of long to short trades or any other characteristic to adapt to evolving market conditions. Further work on the system should investigate this type of functionality.

The system is highly correlated with the market as a whole (proxied by the S&P500), this reduces its diversification value when employed in a broader portfolio together with systems/indexes that also capture the trends of the U.S. stock market.

4.2.5 Note on Risk and Risk Adjusted Performance

Configuration A of the ML driven ATS delivered a skew and kurtosis adjusted Sharpe ratio of 0.2484, adjusting the Sharpe ratio was appropriate given the Shapiro-Wilk test of normality concluded monthly returns were non-normal (although by a small margin). The return distributions of the ML driven ATS generally exhibited negative skew, which is a desirable property, but also a higher kurtosis (above 0, fatter tails than a Gaussian distribution), which ultimately lead to a downward adjustment of the Sharpe ratio for all ATS configurations.

The standard deviation of monthly returns of the ML-driven ATS (configuration A) was 3.03% compared to 3.19% of the S&P500. This reduction in return variability indicates the ATS is at least not riskier than the S&P500 over the test period (in terms of return volatility). The standard deviation of monthly returns is increased as diversification is reduced, e.g., configuration C for the ML-driven ATS yields a standard deviation of 3.24%, but the effect is not substantial.

How the system would respond under less favorable market conditions is unknown. One could conduct backtesting through cross-validation or use synthetic data to get a deeper understanding of the system's performance under different regimes. However, it is unlikely that the system would produce results much different from that of the market, due to their high correlation.

Looking at the last half of 2018, the market took a dip. The ML driven ATS was unable to avoid following the market on its way down. The losses were, however, no more severe than the market's (proxied by the S&P500). This is a situation where having a more market neutral strategy could be beneficial, as it is possible that the negative returns from long trades would be offset by positive returns from short trades. Another technique to limit this type of losses could be to adjust the level of investment in the market by measuring market volatility and/or trend. Losses would likely be less severe if more capital were kept in cash when the market is unstable and/or bearish.

4.2.6 Note on Statistical Significance of ATS Outperformance

Despite outperformance over the 7-year period and year-to-year, the ML-driven ATS's monthly excess returns did not contain sufficient evidence to conclude the ATS produced returns superior to the index on a monthly basis (p-value of 0.3840 for configuration A). Looking at the monthly excess returns on the randomly trading ATS (p-value of 0.8760 for configuration A), it is undoubtedly more evidence of outperformance for the ML-driven ATS.

It proved to be challenging to produce clear outperformance on a monthly basis. The systems high correlation with the S&P500 result in similar returns. Returns could be differentiated more by, for example, allocating some portion of funds to short trades.

5 Conclusion

This project investigated how machine learning could be applied in automated trading systems, how such systems should be backtested and ran regressions on monthly equity risk premiums. The regression results indicate that more advanced ML algorithms (random forests, deep neural nets), which allows complex and non-linear interaction among predictor variables, are more successful in predicting monthly equity risk premiums compared to multiple linear regression, principal component regression, and a historical mean forecast. However, at the highest level, the results are rather unimpressive and fail to demonstrate the added benefit of more complex ML algorithms in a convincing fashion. The low signal to noise ratio and high variance in the datasets together with model -and variable instability are likely some of the culprits of the poor performance.

A random forest classification model for the sign of equity risk premiums was also developed, which produced an out-of-sample accuracy of 53.33%. This is a clear indication of predictive ability exceeding pure chance (confirmed by t-test with a p-value of 0.00).

The project developed an automated trading system driven by two ML models — one to set the side of trades and one to set the size of trades. The models were developed using the triple barrier method and meta-labeling together with a custom dataset, including 86 predictors from the literature. The system was able to outperform the S&P500 index over the 7-year testing period in terms of return as well as adjusted Sharpe ratio, although in a modest fashion. Backtests were also performed on a randomly trading ATS. The ML-driven ATS clearly outperformed it, which indicate that the ML-driven ATS expresses stock picking ability that greatly exceeds random selection. However, the ML driven ATS failed to deliver monthly returns that exceeded those of the S&P500 index with high statistical significance (p-value of 0.3840).

6 Further Work

There are countless ways the system could be improved. For example:

- Produce better models by:
 - o Using more data covering a longer timeline and including more stocks
 - o Including other types of data, like market microstructural data and alternative data
 - o Build a more information dense dataset, through better feature selection and feature engineering processes as well as more advanced sampling schemes that better deal with the variable information arrival rates of financial markets
 - o Adjusting time series better for overlapping information and undesirable statistical properties
 - o Optimizing the training process of the machine learning algorithms
 - o Build models for specific market conditions and events
- Improve the automated trading system by
 - o Having the system adjust its behavior based on market volatility/instability
 - o Updating the capital allocation algorithm
 - o Intelligently allocating capital between different asset classes for a more balanced portfolio
 - o Conduct dynamic reallocation of capital between trades as events occur
 - o Scale up the system by incorporating concepts from high-performance computing
 - o Incorporate more advanced risk management strategies
- Build a backtester that can simulate a wide range of scenarios by using the concepts of cross-validation and synthetic data when simulating history
- Enable backtesting at a larger scale, where multiple strategies can be tested under multiple different scenarios concurrently. This introduces the problem of multiple testing and selection bias, and it is therefore essential to properly adjust statistics for this effect. See “The Sharpe Ratio Efficient Frontier” (2012) and “The Deflated Sharpe Ratio: Correcting for Selection Bias, Backtest Overfitting and Non-Normality” (2014) by Baily and López de Prado.

A particularly exciting direction the project could take is to broaden its aim, such that the goal no longer is to produce a single investment strategy but rather develop an assembly line for investment strategies. Use teamwork and the scientific method to produce discoveries and new strategies at a predictable rate. Have each strategy go through a lifecycle of development, testing, paper-trading, live trading, and decommission. It is necessary to continually evolve and innovate in order to stay ahead in the investing game.

7 References

Books

Marcos López de Prado (2018). *Advances in Financial Machine Learning*. New Jersey: Wiley.

Allen Timmermann and Graham Elliot (2013). *Handbook of Economic Forecasting – Chapter 6: Forecasting Stock Returns*. Amsterdam: North Holland.

Tom Mitchell (1997). *Machine Learning*, New York City: McGraw Hill.

Aurélien Géron (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. California: O'Reilly Media Inc.

Walpole, Myers, Myers, and Ye (2014). *Probability and Statistics for Scientists and Engineers*. Edinburgh Gate: Pearson.

Pratap Dangeti (2017). *Statistics for Machine Learning*. Birmingham: Packt Publishing.

Khanna and Awad (2015). *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. New York: Apress

University of Stavanger (2018). *Compendium for IND640 – part 2 on Portfolio theory and Value-at-Risk (VaR)*. Stavanger: University of Stavanger.

Articles

Gu, Kelly, and Xiu. *Empirical Asset Pricing via Machine Learning*. Chicago Booth Research Paper No. 18-04; 31st Australasian Finance and Banking Conference 2018. (2018)

McLean, R. D., & Pontiff, J. *Does Academic Research Destroy Stock Return Predictability?*. Journal of Finance. (2016)

Hou, Xue, and Zhang. *Replicating Anomalies*. Fisher College of Business Working Paper No. 2017-03-010; 28th Annual Conference on Financial Economics and Accounting; Charles A. Dice Center Working Paper No. 2017-10, p. 1-21., 2017.

Yan and Zheng. *Fundamental Analysis and the Cross-Section of Stock Returns: A Data-Mining Approach*. The Review of Financial Studies, Volume 30, Issue 4, p. 1382–1423, 2017.

Baily and López de Prado. *The Deflated Sharpe Ratio: Correction for Selection Bias, Backtest Overfitting and Non-Normality*. Journal of Portfolio Management, Volume 40, Issue 5, p. 94-107, 2014.

Baily and López de Prado. *The Sharpe Ratio Efficient Frontier*. Journal of Risk, Volume 15, Issue 2, 2012.

Campbell and Thompson. *Predicting excess stock returns out of sample: Can anything beat the historical average?* The Review of Financial Studies, Volume 21, Issue 4, p. 1509-1531, 2008.

Goyal and Welch. *Predicting the Equity Premium with Dividend Ratios*. Management Science, Volume 49, Issue 5, p. 639-654, 2003.

Goyal and Welch. *A Comprehensive Look at The Empirical Performance of Equity Premium Prediction*. Review of Financial Studies, Oxford University Press for Society for Financial Studies, Volume 21, Issue 4, p. 1455-1508, 2008.

Fama and French. *Dissecting Anomalies*. Journal of Finance, Volume 63, Issue 4, 1653-1678, 2008.

Lewellen. *The Cross-section of Expected Stock Returns*. Critical Finance Review, 4: 1–44, 2015.

Pesaran and Timmermann. Predictability of Stock Returns: Robustness and Economic Significance. Journal of Finance, Volume 50, Issue 4, p. 1201-28, 1995.

Green, Hand, and Zhang. *The Remarkable Multidimensionality in the Cross-Section of Expected U.S. Stock Returns*. (2013)

Online Resources

Board of Governors of the Federal Reserve System (US), 3-Month Treasury Bill: Secondary Market Rate [DTB3], retrieved from FRED, Federal Reserve Bank of St. Louis.

Available at: <https://fred.stlouisfed.org/series/DTB3> (read 25.04.2019)

Interactive Brokers. (2019). Commissions – Stocks, ETFs (ETPs) Warrants - Tierd Pricing Structure – North America

Available at: <https://www.interactivebrokers.co.uk/en/index.php?f=39753&p=stocks2> (read 09.04.2019)

Scikit-Learn. (2019). Scikit-Learn – sklearn.ensemble.RandomForestClassifier

Available at: [https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

[learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) (read 25.05.2019)

Scikit-Learn. (2019). Scikit-Learn – Decision Trees

Available at: <https://scikit-learn.org/stable/modules/tree.html> (read 20.05.2019).

Relevant Online Documentation

Scikit-Learn. (2019). Scikit-Learn (v. 0.20.3) Documentation

Available at: <https://scikit-learn.org/0.20/documentation.html>

TensorFlow. (2019). Tensorflow (v. 1.13.1) Documentation

Available at: https://www.tensorflow.org/api_docs/python/tf

Quandl. (2019). Sharadar's Core U.S. Equities Bundle

Available at: <https://www.quandl.com/databases/SFA/data>

Project Source Code

Didrik Nyhus Fleischer. (2019). Github - Automated Trading System

Available at: <https://github.com/DidrikF/automated-trading-system>

8 Appendices

8.1 Appendix A – Dataset Features

Below is a table of features using in the project to train ML algorithms. The table includes the formulas implemented in software to calculate the feature's values.

A value from a source dataset in the “Core US Equities Bundle” delivered by Sharadar Co. takes the following general form: `table_name[variable_name]time_index`

The formulas have the following notation:

- SF1 refers to the Core US Fundamentals Data table.
- SEP refers to Sharadar's Equity Prices table.
- `variable_name` refers to the column name in the respective source dataset.
- `time_index` explanations:
 - o t-1: most recent yearly value
 - o t-2: one year old (relative to the most recently available value) yearly value
 - o q-1: most recent quarterly value
 - o q-2: one quarter old (relative to the most recently available value) quarterly value

Several feature descriptions were gathered from “The Remarkable Multidimensionality in the Cross-Section of Expected US Stock Returns” (Green, Hand, Zhang, 2013).

Note that variable calculations may deviate some from their original description, due to data availability or other limitations.

Table 21: Information about Dataset Features

Acronym	Name	Paper's author(s)	Date, Journal	Written description	Implementation Formula
age	Nr. years since going public	Jiang, Lee & Zhang	2005, RAS	Number of years since first Compustat coverage.	$SF1[datekey]t-1 - TICKERS[firstpricedate]$
agr	Asset growth	Cooper, Gulen & Schill	2008, JF	Annual percent change in total assets.	$(SF1[assets]t-1 / SF1[assets]t-2) - 1$
beta	Beta	Fama & MacBeth	1973, JPE	Estimated market beta from weekly returns and equal weighted market returns for the past two years.	$Cov(R_i, R_m) / Var(R_m)$, where R_i, R_m is weekly returns for a stock and the market respectively. Returns are sampled weekly from the preceding two years of data.
betasq	Beta Squared	Fama & MacBeth	1973, JPE	Market beta squared.	$beta^2$
bm	Book-to-market	Rosenberg, Reid & Lanstein	1985, JPM	Book value of equity divided by end of fiscal-year-end market capitalization.	$SF1[equityusd]t-1 / SF1[marketcap]t-1$
bm_ia	Industry-adjusted book-to-market	Asness, Porter & Stevens	2000, WP	Industry adjusted book-to-market ratio.	$bm - industry_mean(bm)$
cash	Cash holdings	Palazzo	2012, JFE	Cash and cash equivalents divided by average total assets.	$SF1[cashneqsd]t-1 / SF1[assetsavg]t-1$
cashdebt	Cash flow to debt	Ou & Penman	1989, JAE	Earnings before depreciation and extraordinary items divided by avg. total liabilities.	$(SF1[revenueusd]t-1 + SF1[depamor]t-1) / ((SF1[liabilites]t-1 - SF1[liabilites]t-2) / 2)$
cashpr	Cash productivity	Chandrashekar & Rao	2009, WP	Fiscal year end market capitalization plus long term debt minus total assets divided by cash and equivalents.	$(SF1[marketcap]t-1 + SF1[debtnc]t-1 - SF1[assets]t-1) / SF1[cashneq]t-1$
cfp_ia	Industry-adjusted cash flow to price ratio	Asness, Porter & Stevens	2000, WP	Industry adjusted (using Sharadar's GICS approximation) cash flow to price ratio.	$cfp - industry_mean(cfp)$
chatoia	Industry-adjusted change in asset turnover	Soliman	2008, TAR	Sharadar's approximation to GISC fiscal-year mean adjusted change in sales divided by average total assets.	$((SF1[revenueusd]t-1 - SF1[revenueusd]t-2) / SF1[assetsavg]t-1) - industry_mean((SF1[revenueusd]t-1 - SF1[revenueusd]t-2) / SF1[assetsavg]t-1)$
chcsho	Change in shares outstanding	Pontiff & Woodgate	2008, JF	Annual percent change in shares outstanding.	$(SF1[sharesbas]t-1 / SF1[sharesbas]t-2) - 1$
chdebtc_sale	Change in current debt over sales	Yan & Zheng	2017, RFS	Annual change in current debt over sales	$(SF1[debtct]t-1 - SF1[debtct]t-2) / SF1[revenueusd]t-1$
chint_lagat	Change in interest expenses over lagged total assets	Yan & Zheng	2017, RFS	Annual change in interest expenses over lagged assets	$(SF1[intexp]t-1 - SF1[intexp]t-2) / SF1[assets]t-2$
chint_laglt	Change in interest expenses over lagged total liabilities	Yan & Zheng	2017, RFS	Annual change in interest expenses over lagged total liabilities	$(SF1[intexp]t-1 - SF1[intexp]t-2) / SF1[liabilities]t-2$
chinv	Change in inventory	Thomas & Zhang	2002, RAS	Change in inventory scaled by average total assets.	$(SF1[inventory]t-1 - SF1[inventory]t-2) / SF1[assetsavg]t-1$
chinvt_lagcor	Change in inventory over lagged cost of revenue	Yan & Zheng	2017, RFS	Change in inventory the past year divided by last years cost of revenue	$(SF1[inventory]t-1 - SF1[inventory]t-2) / SF1[cor]t-2$
chinvt_lagsale	Change in inventory over lagged revenue	Yan & Zheng	2017, RFS	Change in inventory the past year divided by last year's revenue.	$(SF1[inventory]t-1 - SF1[inventory]t-2) / SF1[revenueusd]t-2$
chlct_lagat	Change in current liabilities over lagged assets	Yan & Zheng	2017, RFS	Change in current liabilities over the past year divided by last year's invested capital.	$(SF1[liabilitiesc]t-1 - SF1[liabilitiesc]t-2) / SF1[assets]t-2$
chlt_laginvcap	Change in liabilities over lagged invested capital	Yan & Zheng	2017, RFS	Change in liabilities over the past year divided by last year's invested capital	$(SF1[liabilities]t-1 - SF1[liabilities]t-2) / SF1[invcap]t-2$
chmom	Change in 6-month momentum	Gettleman & Marks	2006, WP	Cumulative returns from months $t-6$ to $t-1$ minus months $t-12$ to $t-7$.	$((SEP[close]u-1 / SEP[close]u-6) - 1) - ((SEP[close]u-7 / SEP[close]u-12) - 1)$
chpay_lagact	Change in payables over lagged current assets	Yan & Zheng	2017, RFS	Change in payables over the past year divided by last year's current assets.	$(SF1[payables]t-1 - SF1[payables]t-2) / SF1[assetct]t-2$
chpmia	Industry-adjusted change in profit margin	Soliman	2008, TAR	Sharadar's approximation to GISC fiscal-year mean adjusted change in income before extraordinary items divided by sales.	$(SF1[netinc]t-1 / SF1[revenueusd]t-1) - (SF1[netinc]t-2 / SF1[revenueusd]t-2) - industry_mean((SF1[netinc]t-1 / SF1[revenueusd]t-1) - (SF1[netinc]t-2 / SF1[revenueusd]t-2))$

Acronym	Name	Paper's author(s)	Date, Journal	Written description	Implementation Formula
chppne_laglt	Change in PP&E over lagged liabilities	Yan & Zheng	2017, RFS	Change in PP&E over the past year divided by last year's liabilities.	$(SF1[ppnet]t-1 - SF1[ppnet]t-2) / SF1[liabilities]t-2$
chtl_lagat	Change in liabilities over lagged assets	Yan & Zheng	2017, RFS	Change in liabilities over the past year divided by last year's assets.	$(SF1[liabilities]t-1 - SF1[liabilities]t-2) / SF1[assets]t-2$
chtx	Change in tax expense	Thomas & Zhang	2011, JAR	Percent change in total taxes from quarter t-4 to t.	$(SF1[taxexp]q-1 / SF1[taxexp]q-5) - 1$
cinvest	Corporate investment	Titman, Wei & Xie	2004, JFQA	Change over one quarter in net PP&E divided by sales minus the average of this variable for prior 3 quarters; if saleq = 0, then scale by 0.01.	$(SF1[ppnet]q-1 - SF1[ppnet]q-2) / SF1[revenueusd]q-1 - \text{avg}((SF1[ppnet]q-i - SF1[ppnet]q-i-1) / SF1[revenueusd]q-i, i=[2,3,4])$ NB: if sales is zero scale change in ppenet by 0.01
currat	Current ratio	Ou & Penman	1989, JAE	Current assets / current liabilities.	$SF1[assetst]t-1 / SF1[liabilitest]t-1$
debtc_sale	Current debt over sales	Yan & Zheng	2017, RFS	Current debt divided by sales	$SF1[debtct]t-1 / SF1[revenueusd]t-1$
depr	Depreciation over PP&E	Holthausen & Larcker	1992, JAE	Depreciation divided by PP&E.	$SF1[depamor]t-1 / SF1[ppnet]t-1$
dovol	Dollar trading volume	Chordia, Subrahmanyam & Anshuman	2001, JFE	Natural log of trading volume times price per share from month t-2.	$\ln(\text{sum}(SEP[close]*SEP[volume]))$ using daily data for the preceding two months
dy	Dividend to price	Litzenberger & Ramaswamy	1982, JF	Total dividends divided by market capitalization at fiscal year-end.	$(\text{sum}(SEP[dividend]) \text{ the past year at } t-1) / SF1[marketcap]t-1$
egr	Growth in common shareholder equity	Richardson, Sloan, Soliman & Tuna	2005, JAE	Annual percent change in book value of equity.	$(SF1[equityusd]t-1 / SF1[equityusd]t-2) - 1$
ep	Earnings to price	Basu	1977, JF	Annual income before extraordinary items divided by end of fiscal year market cap.	$SF1[netinc]t-1 / SF1[marketcap]t-1$
eqt_marketcap	Equity to market capitalization	Yan & Zheng	2017, RFS	Equity divided by market capitalization.	$(SF1[equity]t-1 - SF1[intangibles]t-1) / SF1[marketcap]t-1$
gma	Gros profitability	Novy-Marx	2013, JFE	Revenues minus cost of goods sold divided by lagged total assets.	$(SF1[revenueusd]t-1 - SF1[cor]t-1) / SF1[assets]t-2$
grcapx	Growth in capital expenditure	Anderson & Garcia-Feijoo	2006, JF	Percent change in capital expenditures from year t-2 to year t.	$(SF1[capex]t-1 / SF1[capex]t-2) - 1$
idiovol	Idiosyncratic return volatility	Ali, Hwang & Trombley	2003, JFE	Standard deviation of residuals of weekly returns on weekly equal weighted market returns for 2 years.	$\text{std}(SEP[weekly_return], \text{Equal weighted weekly market returns})$ using 2 years of data
ill	Illiquidity	Amihud	2002, JFM	Average of daily (absolute return / dollar volume).	$\text{avg}(SEP[close]-SEP[open] / ((SEP[close]+SEP[open]) / 2)*SEP[volume])$ for the past year
indmom	Industry momentum	Moskowitz & Grinblatt	1999, JF	Equal weighted average industry 12-month returns.	$\text{equal_weight_industry_avg}(SEP[close]m-1 / SEP[close]m-12 - 1)$
invest	Capital expenditures and inventory	Chen & Zhang	2010, JF	Annual change in gross property, plant, and equipment + annual change in inventories all scaled by lagged total assets.	$((SF1[ppnet]t-1 - SF1[ppnet]t-2) + (SF1[inventory]t-1 - SF1[inventory]t-2)) / SF1[assetst]t-2$
ipo	Initial public offering	Loughran, Ritter & Ritter	1995, JF	An indicator variable equal to 1 if first year available on CRSP monthly stock file.	$\text{if } (SF1[datekey]t-1 - \text{TICKERS}[firstpricedate]) \leq 1 \text{ year: } 1; \text{ else: } 0$
lev	Leverage	Bhandari	1988, JF	Total liabilities divided by fiscal year end market capitalization.	$SF1[liabilites]t-1 / SF1[marketcap]t-1$
lgr	Growth in long-term debt	Richardson, Sloan, Soliman & Tuna	2005, JAE	Annual percent change in total liabilities.	$(SF1[liabilites]t-1 / SF1[liabilites]t-2) - 1$
maxret	Maximum daily return	Bali, Cakici & Whitelaw	2011, JFE	Maximum daily return from returns during calendar month t-1.	$\text{max}(SEP[open]d - SEP[close]d)$ last 1 month
mom12m	12-month momentum	Jegadeesh	1990, JF	11-month cumulative returns ending one month before month end.	$(SEP[close]m-1 / SEP[close]m-12) - 1$
mom1m	1-month momentum	Jegadeesh & Titman	1993, JF	1-month cumulative return.	$(SEP[close]m / SEP[close]m-1) - 1$
mom24m	24-month momentum	Not applicable	Not applicable	24-month cumulative return	$(SEP[close]m / SEP[close]m-24) - 1$
mom6m	6-month momentum	Jegadeesh & Titman	1993, JF	5-month cumulative returns ending one month before month end.	$(SEP[close]m-1 / SEP[close]m-6) - 1$

Acronym	Name	Paper's author(s)	Date, Journal	Written description	Implementation Formula
ms	Financial statement score (G-score)	Mohanram	2005, RAS	Sum of 8 indicator variables for fundamental performance.	See source code for implementation (deviates some from original specification)
mve	Size	Banz	1981, JFE	Natural log of market capitalization at end of month t-1.	$\ln(SEP[close]_{m-1} * SF1[sharefactor]_{t-1} * SF1[sharesbas]_{t-1})$
mve_ia	Industry-adjusted size	Asness, Porter & Stevens	2000, WP	Industry-adjusted fiscal year-end market capitalization (using Sharadar's approximation to GISC).	$SF1[marketcap]_{t-1} - industry_mean(SF1[marketcap]_{t-1})$
nincr	Number of earnings increases	Barth, Elliott & Finn	1999, JAR	Number of consecutive quarters (up to eight quarters) with an increase in earnings over same quarter in the prior year.	See source code for implementation
operprof	Operating profitability	Fama & French	2015, JFE	Revenue minus cost of goods sold - SG&A expense - interest expense divided by lagged common shareholders' equity.	$(SF1[revenueusd]_{t-1} - SF1[cor]_{t-1} - SF1[sgna]_{t-1} - SF1[intexp]_{t-1}) / SF1[equityusd]_{t-2}$
pchcapex_ia	Industry-adjusted % change in capital expenditure	Abarbanell & Bushee	1998, TAR	Sharadar's approximation to GISC - fiscal-year mean adjusted percent change in capital expenditures.	$((SF1[capex]_{t-1} / SF1[capex]_{2-1}) - 1) - industry_mean((SF1[capex]_{t-1} / SF1[capex]_{2-1}))$
pchcurrat	% change in current ratio	Ou & Penman	1989, JAE	Percent change in <i>currat</i> .	$(SF1[assetsc]_{t-1} / SF1[liabilitesc]_{t-1}) / (SF1[assetsc]_{t-2} / SF1[liabilitesc]_{t-2}) - 1$
pchdepr	% change in depreciation	Holthausen & Larcker	1992, JAE	Percent change in <i>depr</i> .	$(SF1[depamor]_{t-1} / SF1[ppnet]_{t-1}) / (SF1[depamor]_{t-2} / SF1[ppnet]_{t-2}) - 1$
pchgm_pchsale	% change in gross margin - % change in sales	Abarbanell & Bushee	1998, TAR	Percent change in gross margin minus percent change in sales.	$((SF1[gross_margin]_{t-1} / SF1[gross_margin]_{t-2}) - 1) - ((SF1[revenueusd]_{t-1} / SF1[revenueusd]_{t-2}) - 1)$
pchint	Percent change in interest expense	Yan & Zheng	2017, RFS	Percent change in interest expense over the past 1 year.	$(SF1[intexp]_{t-1} - SF1[intexp]_{t-2}) - 1$
pchlt	Percent change in liabilities	Yan & Zheng	2017, RFS	Percent change in liabilities over the past 1 year.	$(SF1[liabilities]_{t-1} / SF1[liabilities]_{t-2}) - 1$
pchppne	Percent change in PP&E	Yan & Zheng	2017, RFS	Percent change in PP&E over the past 1 year.	$(SF1[ppnet]_{t-1} / SF1[ppnet]_{t-2}) - 1$
pchquick	% change in quick ratio	Ou & Penman	1989, JAE	Percent change in quick ratio.	$((quick_ratio]_{t-1} / [quick_ratio]_{t-2}) - 1$
pchsale_pchrect	% change in sales - % change in A/R	Abarbanell & Bushee	1998, TAR	Annual percent change in sales minus annual percent change in receivables.	$((SF1[revenueusd]_{t-1} / SF1[revenueusd]_{t-2}) - 1) - ((SF1[receivables]_{t-1} / SF1[receivables]_{t-2}) - 1)$
pchsale_pchxsga	% change in sales - % change in SG&A	Abarbanell & Bushee	1998, TAR	Annual percent change in sales minus annual percent change in SG&A.	$((SF1[revenueusd]_{t-1} / SF1[revenueusd]_{t-2}) - 1) - ((SF1[sgna]_{t-1} / SF1[sgna]_{t-2}) - 1)$
ps	Financial statements score (F-Score)	Piotroski	2000, JAR	Sum of 9 indicator variables to form fundamental health score.	See source code for impementation
quick	Quick ratio	Ou & Penman	1989, JAE	(current assets - inventory) / current liabilities.	$(SF1[assetsc]_{t-1} - SF1[inventory]_{t-1}) / SF1[liabilitesc]_{t-1}$
rd_sale	R&D to sales	Guo, Lev & Shi	2006, JBFA	R&D expense divided by sales.	$SF1[rd]_{t-1} / SF1[revenueusd]_{t-1}$
retvol	Return volatility	Ang, Hodrick, Xing & Zhang	2006, JF	Standard deviation of daily returns from month t-1.	$std(sep[close] / SEP[open] - 1)$ using one month of daily data.
roaq	Return on assets	Balakrishnan, Bartov & Faurel	2010, JAE	Income before extraordinary items divided by one quarter lagged total assets.	$SF1[netinc]_{q-1} / SF1[assets]_{q-2}$
roavol	Earnings volatility	Francis, LaFond, Olsson & Schipper	2004, TAR	Standard deviation for 16 quarters of income before extraordinary items divided by average total assets.	$std(SF1[netinc]_{q-1} / avg(SF1[assets]_{q-1}))$ using data from the last 8 10-Q filings
roeq	Return on equity	Hou, Xue & Zhang	2014, RFS	Earnings before extraordinary items divided by lagged common shareholders' equity.	$SF1[netinc]_{t-1} / SF1[equity]_{t-2}$
roic	Return on invested capital	Brown & Rowe	2007, WP	Annual earnings before interest and taxes minus non-operating income divided by non-cash enterprise value.	$(SF1[ebit]_{t-1} - [nopinc]_{t-1}) / (SF1[equity]_{t-1} + SF1[liabilities]_{t-1} + SF1[cashneq]_{t-1} - SF1[investmentsc]_{t-1})$
rsup	Revenue surprise	Kama	2009, JBFA	Sales from quarter t minus sales from quarter t-4 divided by fiscal-quarter end market capitalization.	$(SF1[revenueusd]_{q-1} - SF1[revenueusd]_{q-5}) / SF1[marketcap]_{q-1}$
salecash	Sales to cash	Ou & Penman	1989, JAE	Annual sales divided by cash and cash equivalents.	$SF1[revenueusd]_{t-1} / SF1[cashneq]_{t-1}$
salerec	Sales to receivables	Ou & Penman	1989, JAE	Annual sales divided by accounts receivable.	$SF1[revenueusd]_{t-1} / SF1[receivables]_{t-1}$
sgr	Sales growth	Lakonishok, Shleifer & Vishny	1994, JF	Annual percent change in sales.	$(SF1[revenueusd]_{t-1} / SF1[revenueusd]_{t-2}) - 1$
sin	Sin stocks	Hong & Kacperczyk	2009, JFE	An indicator variable equal to 1 if a company's primary industry classification is in smoke or tobacco, beer or alcohol, or gaming.	if TICKER[industry] is in ["Beverages - Brewers", "Beverages - Wineries & Distilleries", "Electronic Gaming & Multimedia", "Gambling", "Tobacco"]; 1; else: 0
sp	Sales to price	Barbee, Mukherji, & Raines	1996, FAJ	Annual revenue divided by fiscal-year-end market capitalization.	$SF1[revenueusd]_{t-1} / SF1[marketcap]_{t-1}$

Acronym	Name	Paper's author(s)	Date, Journal	Written description	Implementation Formula
std_dolvol	Volatility of liquidity (dollar trading volume)	Chordia, Subrahmanyam & Anshuman	2001, JFE	Monthly standard deviation of daily dollar trading volume.	$std((SEP[open]d - SEP[close]d)/2 * SEP[volume]d) * \sqrt{22}$, using daily data from the leading month
std_turn	Volatility of liquidity (share turnover)	Chordia, Subrahmanyam, & Anshuman	2001, JFE	Monthly standard deviation of daily share turnover.	$std(SEP[volume]d/SF1[sharesbas]t-1) * \sqrt{22}$, using daily data from the leading month
sue	Earnings Surprise	Rendelman, Jones & Latane	1982, JFE	The seasonally differenced quarterly earnings before extraordinary items from Compustat quarterly file.	$(SF1[netinc]q-1 - SF1[netinc]q-5) / SF1[marketcap]q-1$
tang	Debt capacity/firm tangibility	Almeida & Campello	2007, RFS	Cash holdings + 0.715 × receivables + 0.547 × inventory + 0.535 × PP&E/ total assets.	$SF1[cashnequsd]t-1 + 0.715 * SF1[recievables]t-1 + 0.547 * SF1[inventory]t-1 + 0.535 * (SF1[ppnetet]t-1 / SF1[assets]t-1)$
tangibles_marketcap	Tangibles over marketcap	Yan & Zheng	2017, RFS	Tangibles divided by market capitalization.	$SF1[tangibles]t-1 / SF1[marketcap]t-1$
tb	Tax income to book income	Lev & Nissim	2004, TAR	Tax income, calculated from current tax expense divided by maximum federal tax rate, divided by income before extraordinary items.	$(SF1[taxexp]t-1 / 0.21) / SF1[netinc]t-1$
turn	Share turnover	Datar, Naik & Radcliffe	1998, JFM	Average monthly trading volume for most recent 3 months scaled by number of shares outstanding in current month.	$avg(SEP[volume]m-1, SEP[volume]m-2, SEP[volume]m-3) / SF1[sharesbas]t-1$
zerotrade	Zero trading days	Liu	2006, JFE	Turnover weighted number of zero trading days for most recent 1 month.	See source code for implementation