Authors: Frank Kornet and Harshad Hazariwala
Date: 26 March 2021

Introduction

Both authors have an interest in stock markets, and how machine learning can be used in trading stocks. One of the authors developed a trading machine learning system called "Stockie" (https://github.com/frkornet/Stockie). We intend to use this system and improve it using what we learned in Georgia Tech courses, specifically Computational Data Analysis (ISyE 6740), Machine Learning for Trading (CSE 7646), and Financial Modeling (MGT 8813).

Figure 1 shows how the main Stockie components interoperate:

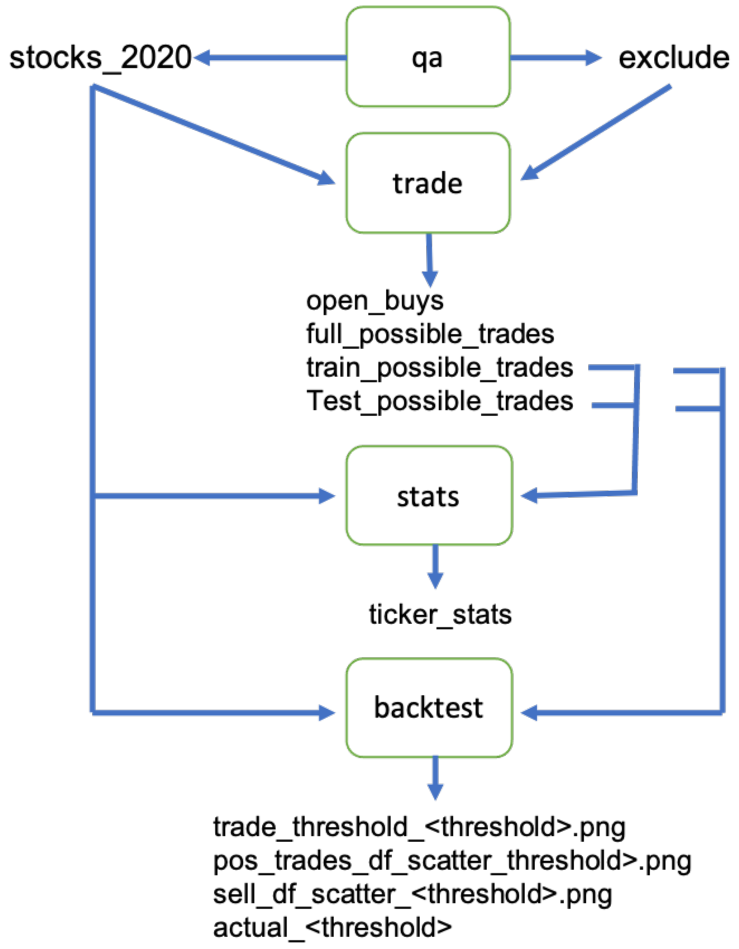

Figure 1: Main Components Stockie.

The files without extensions are CSV files. The key part of the machine learning trading system resides in trade.py where the possible trades are generated (a full buy and sell cycle). The program backtest.py uses heuristics to select trades to execute out of this pool of possible trades. To keep the scope manageable for the Computational Data Analysis manageable, we will focus on analyzing and improving trade.py.

1. Problem Statement

The trade.py program takes in the stock ticker symbols in stocks_2020.csv file. This file contains the list of stocks that Renaissance Technologies and Rockwell held as of May 2020. This list is compiled from the F-13 filings to the Stock Exchange Committee and was put together using web scraping. The assumption is that if those two large hedge funds consider it worthy to hold positions in these stocks, then these must be a good subset to start with. It avoids having to deal with all the stocks that are traded on the stock exchange in the US. It cuts down the subset from 13,000 symbols to 2,000. The exclude.csv contains stock symbols that need to be excluded for quality assurance reasons.

For each stock symbol, we build a smoothed price curve to reduce the noise that exists in the share price. This is then used to identify local minima and maxima using Numpy's argmin and argmax functions. These local minima and maxima are then used to train a logistic regression model to determine the buy and sell signals in the test data.

The logistic regression model uses the following features:

- MACD_<window>
- PctDiff_<window>
- StdDev_<window>

Where <window> is 3, 5, 10, 15, 20, 30, 45, or 60 days. Resulting in a total of 24 features. Each stock will have its own logistic regression model. The specific logistic regression model used, is called a weighted balance of evidence score, and is typically used in credit scores to determine the default risk of applicants. The model is implemented in Python using sklearn (specifically, using the WOEEncoder and KBinsDiscretizer). A description of the model can be found at https://www.listendata.com/2015/03/weight-of-evidence-woe-and-information.html.

In Figure 2, you can see a summary of the performance of the logistic regression model. The performance shows the performance on training data and the performance on the test data. The training data is 7 years' worth of data and consists of 36,950 possible trades. The testing data consists of 3 years' worth of data and consists of 21,950 possible trades. The data spans a ten-year period (i.e., from 13 February 2014 to 13 February 2021). The possible trades are in 2,389 different stock symbols.

The charts in the top row of Figure 2 split trades into trades that have a positive return ("gain"), trades that have a negative return ("loss"), and trades that have a zero or close to zero return ("zero"). For each type we show the row count for each type.

The charts show that there are roughly an equal number of trades that have a positive or negative return in the training data. This is a problem as it potentially allows the logistic regression model to learn the “wrong” thing. So, as a minimum we need to eliminate the negative trades and use the positive trades to train the logistic regression model for each stock.

The top row in Figure 2 for testing data, shows a different pattern compared to that of the training data. The trades that have a positive return outweigh those that have a negative return, esp. up to around 50 trading days. This suggests that the logistic regression model is able to pick up signals for positive trades to a degree that is higher than what you would expect if the efficient market hypothesis were true.

Note that financial data suffers from common pitfalls: 1) small, biased data sets (i.e. only data for what happened and not for what could have happened), 2) low signal-to-noise ratio (the markets are efficient to a fair degree as it would otherwise be easy to consistently make money in the stock market and outperform the market), and 3) we cannot conduct experiments where we test out all possibilities and determine the precise functional relationships between independent variables and the dependent variable.

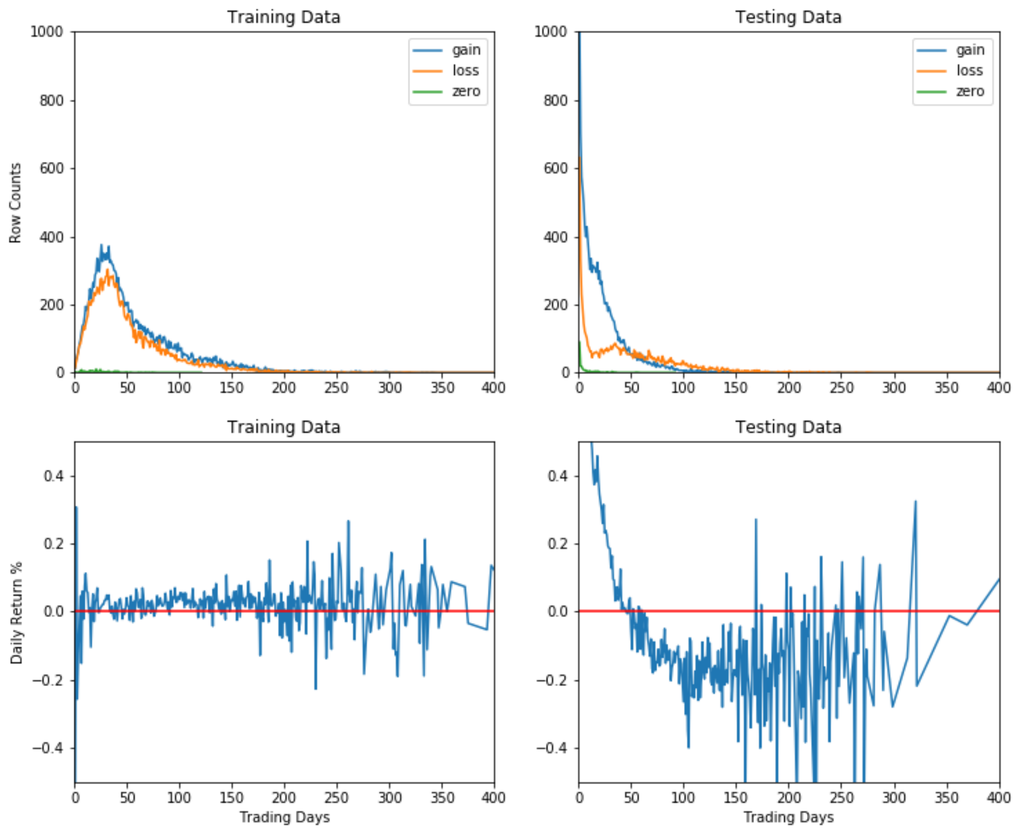

Figure 2: Performance of Logistic Regression Model

The bottom row of Figure 2 shows the overall daily return for each trading day. It executes all the trades for each trading day and then calculates an overall daily return for that particular trading day. This is done for each trading day and then plotted. This allows us to see how the daily return changes as the number of trading days for a trade increases. The plot on the left for training data is what you would expect if the number of possible trades have about an equal number of positive and negative returns.

The plot on the right for testing data shows a different picture and reflects the fact that the number of positive trades outweighs the negative trades up to about 50 trading days. Up to 50 trading days, the logistic regression model does better than on the training data. Beyond 50 trading days the logistic regression model does worse than on the training data since the daily returns seems to fluctuate around -0.2 % daily return (since there around 252 trading days per year this equates to an annual loss of 40 %).

The current implementation of trade.py uses yfinance (https://aroussi.com/post/python-yahoo-finance) to access and retrieve historical price data. The interface does not implement fundamental trading or technical trading indicators, so they are missing in the current model implementation. This is a limitation that we want to try to address in our project using open data sources on the web and to avoid having to pay for proprietary historical data. As good proprietary data can cost millions of dollars and we simply don't have that money available. If we had, we'd prefer to use it to trade and make money that way! The other reason is that a lot of the good proprietary data (models) are used by the major stock market players making them more likely to have the similar models. We believe that having simple and unique models will allow us to outperform the market in the long run. It will require hard work to build these models.

The authors want to improve the current model in three ways:

1) add fundamental and technical indicators to see if and to what degree they improve the performance of the logistic regression model,
2) implement and try out other classification models that we learned in CDA to see if they improve performance, and
3) implement and try out an ensemble of different classification models to see if they improve the performance.

Since it is unclear which of these will improve our trading model the most, we will use an iterative process. The focus will be on identifying which models and features (possibly using an ensemble) work best. Since we only have a limited time available for the project 4 – 6 weeks, we will focus on what can be done in the available time and continue the project afterwards with what has been left out. As part of the final project report, we will provide a set of next steps to continue the work on improving the model and integrating the results back into Stockie.

2. Data Source

As mentioned in the Problem Statement, we are using yfinance to retrieve historical price and volume data. It used to provide daily historical data as it does not provide intraday historical volume and price data. It does provide intraday volume and price data for the ongoing trading day (i.e., as off the time of the call). It also does provide limited fundamental data (up to four years for annual reports and up to four quarters of the current finance year).

Since yfinance only provides limited fundamental data, we want to extend it using data stored in Edgar (SEC) using the XBRL interface (for more information see https://www.sec.gov/structureddata/osd-inline-xbrl.html). The XBRL interface goes back to financial documents filed since 2009. We will use this to store a local copy of the financial documents filed at SEC (K-10, Q-10, and so on). We will then use these files to implement a rolling four quarters price-to-earnings ratio for each stock in our trading universe. Once this works properly, we will extend it to price-to-sales ratio and price-to-book ratio.

We will also implement the Bollinger Band, RSI, MFI, and Williams % R technical indicators using the solutions we implemented for Machine Learning for Trading (Project 6) and converting the code to work on data retrieved using yfinance.

The above intentions depend on how hard it turns out to be to implement these extra sources for data.

3. Methodology

As mentioned above, the efficient market hypothesis (EMH), postulates that it is impossible to predict the stock market using fundamental data, technical data, and insider trading. It basically states that the stock market is a random walk and cannot be predicted. There are three levels recognized in the EMH: strong, semi-strong, and weak. Table 1 below explains what data does not allow prediction of the stock market. The strong EMH basically states that it is not possible to predict the stock market (no matter what data one has available). The semi-strong EMH recognizes that you can predict using insider trading data (illegal in most cases). The weak EMH implies that we can use fundamental and insider trading to predict the stock market. The authors believe that the strong and semi-strong EMH are wrong, and that it is possible to predict the market using a mixture of technical and fundamental data. Lasse Heje Pedersen calls it “Efficiently Inefficient” in his latest book.

	Strong	**Semi-Strong**	**Weak**
Technical	Unable	Unable	Unable
Fundamental	Unable	Unable	
Insider Trading	Unable		

Table 1: Prediction Ability for Different Levels of Efficient Market Hypothesis

Please note, we believe that fundamental data will be of limited value compared to technical data since it deals with data that have little variance (i.e., do not change much since they are published only quarterly or annually). More signal is expected to be available in technical data.

Although the EMH may not be true, it doesn't mean that the stock market is not efficient. To a large degree it is efficient. As a result, it is not easy to build a set of features and models that can predict the stock market. So, our attempt to improve the model in this project may not succeed. So, there will be an element of trial and error that we cannot fully control.

First and foremost, we will rely on the material we learned in Computational Data Analysis (ISyE 6740), Machine Learning for Trading (CSE 7646), and Financial Modeling (MGT 8813). Esp. the material that was covered in CDA will guide what we do.

The authors will rely, furthermore, on ideas expressed in the following books to inform their experiments:

1) Advances in Financial Machine Learning by Marcos Lopez de Prado (2018). John Wiley & Sons, Hoboken, New Jersey, USA.
2) Efficiently Inefficient, How Smart Money Invests and Market Prices are Determined by Lasse Heje Pedersen (2015). Princeton University Press, Princeton, New Jersey, USA.
3) Machine Learning for Algorithmic Treading by Stefan Jansen (2020, 2nd ed.). Packt, Livery Place, Birmingham, UK.

As eluded above, we will use an iterative approach to improve the existing model in small steps. This way we get a sense of what works and what each change adds to the final model.

The criteria we use to determine which solution is better is correlation between the training daily return and the testing daily return after calculating the total return for training and testing data. The total return is then used to calculate the daily return. This is done for each stock in the trading universe. So, we get 2,389 points: (training daily return, testing daily return). We then use Numpy's corr function to calculate the correlation matrix. The current corr score is 0.23.

The underlying assumption here is that if the classification model for recognizing buy and sell signals works well, it should achieve similar results on both the training and testing data. This is, of course, true only to a limited degree since there is a fair amount of randomness in the data since the stock markets are efficient to a degree. The other reason why you will not necessarily get a perfect score is that the training data covers seven years and testing data only covers three years. Despite these obvious limitations, it is a reasonable measure to use. It is, for example, used by numer.ai.

Numer.ai (https://numer.ai) is a platform that publishes cleaned, normalized, regularized and obfuscated data that can be used by data scientists/machine learning experts to develop and train prediction models and publish their predictions using the data. The data provides over 300 features for each stock for an "era" (most likely 4 weeks). Numer.ai then uses these predictions to make their own bets in the stock market. It labels itself as an open-source hedge fund.

If submitted predictions are correct, you can earn money (provided you stake your predictions with hard cold cash using the numerai cryptocurrency). You can consider the stake a bet, and the higher the stakes the more money you can earn. The amount you can earn is determined by three factors:

1) correlation,
2) meta model contribution, and
3) feature neutral correlation.

The following formula is used to determine how much will be paid out

payout = stake_value * payout_factor * (corr * corr_multiplier + mmc * mmc_multiplier)

The payout factor drops from 1.0 to 0.2 depending on the total amount of money staked by all prediction models for a particular period, i.e., max(min(1, 300,000/total amount staked), 0.2). You select the corr_multiplier (1.0), and mmc_multiplier (0.0, 0.5, 1.0 or 2.0).

You can earn or lose up to twenty-five per cent per round (twenty trading days) depending on the performance of your model. The payout is aligned with the interests of numer.ai since it wants a high correlation between performance on training and test data, so it can use training data performance for predicting which models perform best in the stock market the coming four weeks.

Note that if you lose money, your numerai cryptocurrency is destroyed. Resulting in a smaller amount of numerai. There are currently around 11 million numerai in circulation. This reduction will slowly push up the valuation of the numerai currency over time. The numerai currency is built on top of the Ethereum smart contracts and Ether cryptocurrency.

The meta model contribution is a way to promote models that are different. This was recently added to stimulate more diverse models. This is no doubt driven by the desire to get a wider range of models to better model what is going to happen in the stock market in the coming four weeks.

The feature neutral correlation is intended to penalize models that only rely on a small number of features. Numer.ai prefers a larger range of features to ensure that models have a better chance to work in the long term, esp. when the regime of the stock market changes. The factor does not appear in the formula numer.ai publishes on their website, but it is no doubt used in some form or shape.

For this project we will, therefore, use correlation as a measure of performance since it is used by a real hedge funds to payout contributors, and it makes logical sense. Beyond the CDA project, the authors will investigate whether and how to incorporate the feature neutral correlation concept into a good performance measure.

A final note, backtesting is typically prone to overfitting and hence measures need to be put in place to reduce this potential and very real danger. Given the limited time available, for this project, we will ignore this risk. To some degree we will manage this by forward testing the performance of the final model(s) before investing money in the implemented strategies.

High-level breakdown structure of required tasks for the CDA project is as follows:

1. Eliminate trades that have a negative return
2. Implement technical indicators
3. Implement fundamental indicators
4. Implement alternative classifiers
5. Implement ensemble of classifiers
6. Write final report

A simple high-level schedule of the work breakdown structure is shown in Figure 3.

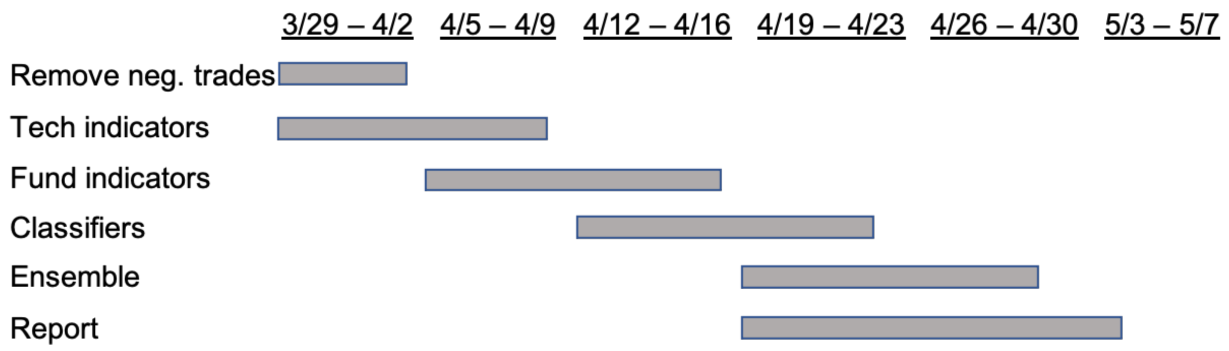

Figure 3: High-level Schedule for CDA Project.

Please note that the above schedule depends on things going well, and homework 6 not taking too much time for the authors to complete. If we are unable to execute the plan above, we will explain what happened and why, as well as what steps we took to mitigate the impact. The risk will be managed by reviewing the progress weekly and adjusting plan as needed.

<u>4. Evaluation and Final Results</u>

Not yet possible to provide. Will be done as we execute the project.